Recoll user manual

Jean-Francois Dockes

COLLABORATORS

	TITLE : Recoll user manual	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Jean-Francois Dockes	December 14, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

List of Tables

Abstract

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be found at the following location: GNU web site.

This document introduces full text search notions and describes the installation and use of the Recoll application. This version describes Recoll 1.40.

Chapter 1

Introduction

This document introduces full text search notions and describes the installation and use of the Recoll application. It is updated for Recoll 1.40.

Recoll was for a long time dedicated to Unix-like systems. It was only lately (2015) ported to MS-Windows. Many references in this manual, especially file locations, are specific to Unix, and not valid on Windows, where some described features are also not available. The manual will be progressively updated. Until this happens, on Windows, most references to shared files can be translated by looking under the Recoll installation directory (Typically `C:/Program Files (x86)/Recoll`). Especially, anything referenced inside `/usr/share` in this document will be found in the `Share` subdirectory of the installation). The user configuration is stored by default under `AppData/Local/Recoll` inside the user directory, along with the index itself.

1.1 Giving it a try

If you do not like reading manuals (who does?) but wish to give Recoll a try, just install the application and start the **recoll** graphical user interface (GUI), which will ask permission to index your home directory, 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. From the **recoll** GUI go to: Preferences → Indexing configuration, then adjust the Start folders section (or Top directories in older Recoll versions), which defines the directories from which the filesystem exploration starts.

By default, the indexer process writes all errors to its `stderr` output, which may be lost if you started the GUI from the desktop. You may find it useful to assign a file name to the Indexer log file name entry on the above indexing preferences screen. With the default level of 3, this will list all processed documents, and all errors (lines beginning with `:2:`).

On Unix-like systems, you may need to install the appropriate supporting applications for document types that need them (for example antiword for Microsoft Word files). The Windows package is self-contained and includes most useful auxiliary programs. After the indexing ran, the **recoll** GUI Tools → Missing helpers menu entry will show a list of missing supporting applications for the documents found in the indexed area.

1.2 Full text search

Recoll is a full text search application, which means that it finds your data by content rather than by external attributes (like the file name). You specify words (terms) which should or should not appear in the text you are looking for, and receive in 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.

Full text search applications try 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. Recoll relies on the Xapian probabilistic information retrieval library to determine relevance.

In many cases, you are looking for all the forms of a word, including plurals, different tenses for a verb, or terms derived from the same root or *stem* (example: `floor, floors, floored, flooring...`). Queries are usually automatically expanded to all such related terms (words that reduce to the same stem). This can be prevented for searching for a specific form.

Stemming, by itself, does not accommodate for misspellings or phonetic searches. A full text search application may also support this form of approximation. For example, a search for `aliterattion` returning no result might propose `alliteration, alteration, alterations, or altercation` as possible replacement terms. Recoll bases its suggestions on the actual index contents, so that suggestions may be made for words which would not appear in a standard dictionary.

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.

The Xapian library manages an index database which describes where terms appear in your document files. It efficiently processes the complex queries which are produced by the Recoll query expansion mechanism, and is in charge of the all-important relevance computation task.

Recoll provides the mechanisms and interface to get data into and out of the index. This includes translating the many possible document formats into pure text, handling term variations (using Xapian stemmers), and spelling approximations (using the aspell speller), interpreting user queries and presenting results.

In a shorter way, Recoll does the dirty footwork, Xapian deals with the intelligent parts of the process.

The Xapian index can be big (roughly the size of the original document set), but it is not a document archive. Recoll can only fully display documents that still exist at the place from which they were indexed. However, recent Recoll version do store the plain text from all indexed documents.

Recoll stores all internal data in Unicode UTF-8 format, and it can index many types of files with different character sets, encodings, and languages into the same index. It can process documents embedded inside other documents (for example a PDF document stored inside a Zip archive sent as an email attachment...), down to an arbitrary depth.

By default. Recoll processes east asian texts by generating terms as arbitrary sequences of consecutive characters (n-grams). However, it has provisions to integrate with language-aware text segmenters for Chinese and Korean which will produce a smaller index and improved search.

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.

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

Recoll can optionally store the raw terms, without accent stripping or case conversion. In this configuration, default searches will behave as before, but it is possible to perform searches sensitive to case and diacritics. This is described in more detail in the section about index case and diacritics sensitivity.

Recoll uses many parameters to 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/share/recoll/examples`) during installation. The default values set by the configuration files in this directory may be overridden by values set inside your personal configuration. With the default configuration, Recoll will index your home directory with generic parameters. Most common parameters can be set by using configuration menus in the **recoll** GUI. Some less common parameters can only be set by editing the text files.

The indexing process is started automatically (after asking permission), the first time you execute the **recoll** GUI. Indexing can also be performed by executing the **recollindex** command. Recoll indexing is multithreaded by default when appropriate hardware resources are available, and can perform multiple tasks in parallel for text extraction, segmentation and index updates.

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 query the index:

- A command line interface.
- A Recoll WebUI.
- A Gnome Shell Search Provider .
- A Python programming interface
- A KDE KIO slave module.
- A Ubuntu Unity Scope 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 since the last run. 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`).

recollindex skips files which caused an error during a previous pass. This is a performance optimization, and the command line option `-k` can be set to retry failed files, for example after updating an input handler.

When a file has been deleted, **recollindex** removes the corresponding data from the index. The exact moment when this happens depends on the indexing mode. There are provisions to avoid deleting data for an umounted removable volume.

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

Depending on your data, temporary files may be needed during indexing, some of them possibly quite big. You can use the `RECOLL_TMPDIR` or `TMPDIR` environment variables to determine where they are created (the default is to use `/tmp`). Using `TMPDIR` has the nice property that it may also be taken into account by auxiliary commands executed by **recollindex**.

2.1.1 Indexing modes

Recoll indexing can be performed along two main modes:

- **Periodic (or batch) indexing recollindex** is executed at discrete times. On Unix-like systems, the typical usage is to have a nightly run programmed into your **cron** file. On Windows, the Task Scheduler can be used to run indexing. In both cases, the Recoll GUI includes a simplified interface to configure the system scheduler.
- **Real time indexing recollindex** runs permanently as a daemon and uses a file system alteration monitor (e.g. inotify on Unix-like systems) to detect file changes. New or updated files are indexed at once. Monitoring a big file system tree can consume significant system resources.

2.1.1.1 Choosing an indexing mode

The choice between the two methods is mostly a matter of preference, and they can be combined by setting up multiple indexes (e.g.: use periodic indexing on a big documentation directory, and real time indexing on a small home directory), or by configuring the index so that only a subset of the tree will be monitored.

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

2.1.2 Configurations, multiple indexes

Recoll supports defining multiple indexes, each defined by its own configuration directory. A configuration directory contains several files which describe what should be indexed and how.

When **recoll** or **recollindex** is first executed, it creates a default configuration directory. This configuration is the one used for indexing and querying when no specific configuration is specified. It is located in `$HOME/.recoll/` for Unix-like systems and `%LOCALAPPDATA%/Recoll` on Windows (typically `C:/Users/[me]/Appdata/Local/Recoll`).

All configuration parameters have defaults, defined in system-wide files. Without further customisation, the default configuration will process your complete home directory, with a reasonable set of defaults. It can be adjusted to process a different area of the file system, select files in different ways, and many other things.

In some cases, it may be useful to create additional configuration directories, for example, to separate personal and shared indexes, or to take advantage of the organization of your data to improve search precision.

In order to do this, you would create an empty directory in a location of your choice, and then instruct **recoll** or **recollindex** to use it by setting either a command line option (`-c` *`/some/directory`*), or an environment variable (`RECOLL_CONFDIR=`*`/some/directo`*
Any modification performed by the commands (e.g. configuration customisation or searches by **recoll** or index creation by **recollindex**) would then apply to the new directory and not to the default one.

Once multiple indexes are created, you can use each of them separately by setting the `-c` option or the `RECOLL_CONFDIR` environment variable when starting a command, to select the desired index.

It is also possible to instruct one configuration to query one or several other indexes in addition to its own, by using the External index function in the **recoll** GUI, or some equivalent in the command line and programming tools.

A plausible 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. for example, 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 by using a directory filter clause in a search, but multiple indexes may have better performance and may be worth the trouble in some cases.

A more advanced use case would be to use multiple indexes to improve indexing performance, by updating several indexes in parallel (using multiple CPU cores and disks, or possibly several machines), and then merging them, or querying them in parallel.

See the section about configuring multiple indexes for more detail

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...

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

Other file types (e.g.: 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.

After installing a missing handler, you may need to tell **recollindex** to retry the failed files, by adding option `-k` to the command line, or by using the GUI File → Special indexing menu. This is because **recollindex**, in its default operation mode, will not retry files which caused an error during an earlier pass. In special cases, it may be useful to reset the data for a category of files before indexing. See the **recollindex** manual page. If your index is not too big, it may be simpler to just reset it.

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 define a positive list of types to be indexed. In the latter case, any type not in the list will be ignored.

Excluding files by name can be done by adding wildcard name patterns to the skippedNames list, which can be done from the GUI Index configuration menu. Excluding by type can be done by setting the excludedmimetypes list in the configuration file. This can be redefined for subdirectories.

You can also define an exclusive list of MIME types to be indexed (no others will be indexed), by setting 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).

`excludedmimetypes` or `indexedmimetypes`, can be set either by editing the configuration file (recoll.conf) for the index, or by using the GUI index configuration tool.

Note about MIME types

When editing the `indexedmimetypes` or `excludedmimetypes` lists, you should use the MIME values listed in the `mimemap` file or in Recoll result lists in preference to `file -i` output: there are a number of differences. The `file -i` output should only be used for files without extensions, or for which the extension is not listed in `mimemap`

2.1.4 Indexing failures

Indexing may fail for some documents, for a number of reasons: a helper program may be missing, the document may be corrupt, we may fail to uncompress a file because no file system space is available, etc.

The Recoll indexer in versions 1.21 and later does not retry failed files by default, because some indexing failures can be quite costly (for example failing to uncompress a big file because of insufficient disk space). Retrying will only occur if an explicit option (`-k`) is set on the **recollindex** command line, or if a script executed when **recollindex** starts up says so. The script is defined by a configuration variable (`checkneedretryindexscript`), and makes a rather lame attempt at deciding if a helper command may have been installed, by checking if any of the common `bin` directories have changed.

2.1.5 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. The difference between the two methods is that the second will not change the current index format, which may be undesirable if a newer format is supported by the Xapian version.

2.2 Index storage

The default index location is the `xapiandb` subdirectory of the Recoll configuration directory, typically `$HOME/.recoll/xapiandb/` on Unix-like systems or `C:/Users/[me]/Appdata/Local/Recoll/xapiandb` on Windows. This can be changed via two different methods (with different purposes):

1. 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). Use this method to keep the configuration directory in its default location, and use another location for the index, typically because of disk occupation or performance reasons.
2. 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 command:

```
recoll -c ~/.indexes-email
```

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.

There are quite a few more parameters which can be set in the configuration file itself for tailoring Recoll data storage. They are described in a section of the configuration chapter.

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 in most cases, the space used by the index will be negligible compared to 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 (typically `~/.recoll/xapiandb`), then run a normal indexing command. Using **recollindex** option `-z` would not work in this situation.

2.2.2 Security aspects

The Recoll index does not hold complete copies of the indexed documents (it almost does after version 1.24). 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 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.

2.2.3 Special considerations for big indexes

This only needs concern you if your index is going to be bigger than around 5 GBytes. Beyond 10 GBytes, it becomes a serious issue. Most people have much smaller indexes. For reference, 5 GBytes would be around 2000 bibles, a lot of text. If you have a huge text dataset (remember: images don't count, the text content of PDFs is typically less than 5% of the file size), read on.

The amount of writing performed by Xapian during index creation is not linear with the index size (it is somewhere between linear and quadratic). For big indexes this becomes a performance issue, and may even be an SSD disk wear issue.

The problem can be mitigated by using the following approaches:

- Partition the data set and create several indexes of reasonable size rather than a huge one. These indexes can then be queried in parallel (using the Recoll external indexes facility), or merged using **xapian-compact**.

- Have a lot of RAM available and set the `idxflushmb` Recoll configuration parameter as high as you can without swapping (experimentation will be needed). 200 would be a minimum in this context.
- Use Xapian 1.4.10 or newer, as this version brought a significant improvement in the amount of writes.

Recoll versions 1.38 and newer have an option to use multiple temporary indexes and a final merge internally. This was designed as a CPU performance optimization (increasing parallelism), but it may also provide a simple solution for the index size issue, though it may not give enough control over the temporary indexes physical placement for really huge datasets.

2.3 Index configuration

Variables stored inside the Recoll configuration files control which areas of the file system are indexed, and how files are processed. The values can be set by editing the text files. Most of the more commonly used ones can also be adjusted 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 default configuration directory with empty configuration files, which you can then edit.

The configuration is documented inside the installation chapter of this document, or in the recoll.conf(5) manual page. Both documents are automatically generated from the comments inside the configuration file.

The most immediately useful variable is topdirs, which lists the subtrees and files to be indexed. The variable name is a bit misleading for native English speakers, so the corresponding GUI label is Start folders.

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

There are two incompatible types of Recoll indexes, depending on the treatment of character case and diacritics. A further section describes the two types in more detail. The default type is appropriate in most cases.

2.3.1 The index configuration GUI

Most index configuration parameters can be set from the **recoll** GUI (set `RECOLL_CONFDIR` or use the `-c` option to affect a non-default index.)

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

The Global parameters tab allows setting global variables, like the lists of top/start 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 may 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...

2.3.2 Multiple indexes

Multiple Recoll indexes can be created by using several configuration directories which would typically be set to index different areas of the file system.

A specific configuration can be selected by setting the RECOLL_CONFDIR environment variable or giving the -c option to **recoll** and **recollindex**.

The **recollindex** program, used for creating or updating indexes, always works on a single index. The different configurations are entirely independent (no parameters are ever shared between configurations when indexing).

All the search interfaces (**recoll**, **recollq**, the Python API, etc.) operate with a main configuration, from which both configuration and index data are used, and can also query data from multiple additional indexes. Only the index data from additional indexes is used, their configuration parameters are ignored. This implies that some parameters should be consistent among index configurations which are to be used together.

When searching, the current 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.

Index configuration parameters can be set either by using a text editor on the files, or, for most parameters, by using the recoll index configuration GUI. In the latter case, the configuration directory for which parameters are modified is the one which was selected by RECOLL_CONFDIR or the -c parameter, and there is no way to switch configurations within the GUI.

See the configuration section for a detailed description of the parameters

Some configuration parameters must be consistent among a set of multiple indexes used together for searches. 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 affect the term generation.

Using multiple configurations implies a small level of command line or file manager usage. The user must explicitly create additional configuration directories, the GUI will not do it. This is to avoid mistakenly creating additional directories when an argument is mistyped. Also, the GUI or the indexer must be launched with a specific option or environment to work on the right configuration.

2.3.2.1 Creating and using an additional index: Linux example

The following applies to Unix-like systems

Initially creating the configuration and index:

```
mkdir /path/to/my/new/config
```

Configuring the new index can be done from the **recoll** GUI, launched from the command line to pass the -c option (you could create a desktop file to do it for you), and then using the GUI index configuration tool to set up the index.

```
recoll -c /path/to/my/new/config
```

Alternatively, you can just start a text editor on the main configuration file:

```
someEditor /path/to/my/new/config/recoll.conf
```

Creating and updating the index can be done from the command line:

```
recollindex -c /path/to/my/new/config
```

or from the File menu of a GUI launched with the same option (**recoll**, see above).

The same GUI would also let you set up batch indexing for the new index. Real time indexing can only be set up from the GUI for the default index (the menu entry will be inactive if the GUI was started with a non-default -c option).

The new index can be queried alone with:

```
recoll -c /path/to/my/new/config
```

Or, in parallel with the default index, by starting **recoll** without a -c option, and using the External Indexes tab in the preferences dialog, which can be reached either trough: Preferences → GUI Configuration → External Index Dialog or Query → External index dialog. See the GUI external indexes section for more details.

2.3.2.2 Creating an alternate index: Windows example

When running Recoll under Windows, the simplest approach for using separate indexes is to start the GUI from different desktop icons. The following approach can be used:

1. Create an empty folder somewhere for holding the new configuration and index.
2. Select the Recoll icon on the desktop and Copy/Paste it. If no desktop icon was created during installation, you can right-drag the recoll.exe program from `C:\Program Files (X86)\Recoll` to the desktop and select Create shortcuts here to create one.
3. Right-click the new shortcut and go to the Properties->shortcut tab
4. Modify the Target value from the original `C:\Program Files (x86)\Recoll\recoll.exe` to something like:

   ```
   "C:\Program Files (x86)\Recoll\recoll.exe" -c C:\Path\To\My\New\Directory
   ```

   Use double quotes around the directory path is it contains spaces.
5. Then save the new Icon by clicking ok, and double click it to start a Recoll GUI for the new configuration. You should be presented with the initial configuration dialog.

Any other method for running the GUI or **recollindex** program with a `-c` option or a RECOLL_CONFDIR value in the environment would work too.

2.3.3 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 using case and diacritics to discriminate between terms, e.g., 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).

Recoll creates a stripped index by default if `indexStripChars` is not set.

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 relatively little used, 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.

2.4 Indexing performance and resource usage

2.4.1 Indexing threads configuration (Unix-like systems)

Note: you don't probably don't need to read this. The default automatic configuration is fine is most cases. Only the part about disabling multithreading may be more commonly useful, so I'll prepend it here. In `recoll.conf`:

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

The Recoll indexing process **recollindex** can use multithreading to speed up indexing on multiprocessor systems. This is currently enabled on MacOS systems and Unix-like systems systems, but not under Windows.

The data processing used 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 anything else in this case, thrTCounts is not used) - 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).

Note
If the first value in `thrQSizes` is 0, `thrTCounts` is ignored.

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
```

2.4.2 Using multiple temporary indexes to improve indexing time

Note
The underlying code is buggy between 1.38 and 1.41.0, fixed in 1.41.1. The bug affects the storing of document texts inside the index, so it only affects snippets generation inside result lists. If this result lists snippets are important to you, do not use the function with an affected release.

In some cases, either when the input documents are simple and require little processing (e.g. HTML files), or possibly with a high number of available cores, the single-threaded Xapian index updates can become the performance bottleneck for indexing.

In this case, it is possible to configure the indexer (Recoll 1.38 and later) for using multiple temporary indexes which are merged at the end of the operation. This can provide a huge gain in performance, but, as opposed to multithreading for document preparation, it can also have a (slight) negative impact in some cases, so that it is not enabled by default.

In most cases, this should also be turned off after the initial index creation is done, because it is extremely detrimental to the speed of small incremental updates.

The parameter which controls the number of temporary indexes in `recoll.conf` is named `thrTmpDbCnt`. The default value is 0, meaning that no temporary indexes are used.

If your document set is big, and you are using a processor with many cores for indexing, especially if the input documents are simple, it may be worth it to experiment with the value. For example, with a partial Wikipedia dump (many HTML small files), indexing times could be divided almost by three, by using four temporary indexes on a quad-core machine. More detail in this article on the Recoll WEB site.

All the tests were performed on SSDs, it is quite probable that this approach would not work well on spinning disks, at least not in its current form.

2.4.3 Quieting down the indexing process

The Recoll indexing process, **recollindex** is usually configured to have very low priority and not disturb other activity on the machine. Still, on an idle system, even with multithreading disabled, it will use 100% of one core if needed and available. This may be enough to get a laptop fan to spin up in some cases. To prevent this, we want to limit the CPU utilisation for every short time quanta (e.g. not more than 20 mS for every 100 mS).

This would be extremely difficult to do from inside the indexing process, because of the many places where intensive CPU usage takes place, some not under our control (Xapian or external helpers). Happily enough, on Linux systems, you can use the cgroup facility to throttle the process CPU usage. This is further documented on the Recoll WEB site

2.5 Index update scheduling

2.5.1 Periodic indexing

2.5.1.1 Running the indexer

The **recollindex** program performs index updates. You can start it 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 GUI File menu has entries to start or stop the current indexing operation. When indexing is not currently running, you have a choice between Update Index or Rebuild Index. The first choice only processes changed files, the second one erases the index before starting so that all files are processed.

On Linux and Windows, the GUI can be used to manage the indexing operation. Stopping the indexer can be done from the **recoll** GUI File → Stop Indexing menu entry.

On Linux, the **recollindex** indexing process can be interrupted by sending an interrupt (Ctrl-C, SIGINT) or terminate (SIGTERM) signal.

When stopped, some time may elapse before **recollindex** exits, because it needs to properly flush and close the index.

After 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 inexistent 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).

2.5.1.2 recollindex command line

recollindex has many options which are listed in its manual 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).

Option `-k` will force retrying files which previously failed to be indexed, for example because of a missing helper program.

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.

2.5.1.3 Linux: 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"
```

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. Entries created via the tool are marked with a `RCLCRON_RCLINDEX=` marker so that the tool knows which entries belong to it. As a side effect, this sets an environment variable for the process, but it's not actually used, this is just a marker.

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.

2.5.2 Real time indexing

Real time monitoring/indexing is performed by starting the **recollindex** `-m` command. With this option, **recollindex** will permanently monitor file changes and update the index.

On Windows systems, the monitoring process is started from the **recoll** GUI File menu. On Unix-like systems, there are other possibilities, see the following sections.

When this is in use, the **recoll** GUI File menu makes two operations available: Stop and Trigger incremental pass.

Trigger incremental pass has the same effect as restarting the indexer, and will cause a complete walk of the indexed area, processing the changed files, then switch to monitoring. This is only marginally useful, maybe in cases where the indexer is configured to delay updates, or to force an immediate rebuild of the stemming and phonetic data, which are only processed at intervals by the real time indexer.

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.

As of Recoll 1.24, you can set the monitordirs configuration variable to specify that only a subset of your indexed files will be monitored for instant indexing. In this situation, an incremental pass on the full tree can be triggered by either restarting the indexer, or just running **recollindex**, which will notify the running process. The **recoll** GUI also has a menu entry for this.

2.5.2.1 Unix-like systems: automatic daemon start with systemd

The installation contains two example files (in `share/recoll/examples`) for starting the indexing daemon with systemd.

`recollindex.service` would be used for starting **recollindex** as a user service. The indexer will start when the user logs in and run while there is a session open for them.

`recollindex@.service` is a template service which would be used for starting the indexer at boot time, running as a specific user. It can be useful when running the text search as a shared service (e.g. when users access it through the WEB UI).

If configured to do so, the unit files should have been installed in your system's default systemd paths (usually `/usr/lib/systemd/system/` and `/usr/lib/systemd/user/`). If not, you may need to copy the files there before starting the service.

With the unit files installed in the proper location, the user unit can be started with the following commands:

```
systemctl --user daemon-reload
systemctl --user enable --now recollindex.service
```

The system unit file can be enabled for a particular user by running, as root:

```
systemctl daemon-reload
systemctl enable --now recollindex@username.service
```

(A valid user name should be substituted for `username`, of course.)

2.5.2.2 Unix-like systems: automatic daemon start from the desktop session

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, a good old xdm-based session could have 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).

2.5.2.3 Miscellaneous details

Logging By default, the messages from the indexing daemon will be sent to the same file as those from the interactive commands (`logfilename`). 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.

Unix-like systems: increasing resources for inotify On Linux systems, monitoring a big tree may need increasing the resources available to inotify, which are normally defined in `/etc/sysctl.conf`.

```
### inotify
#
# cat  /proc/sys/fs/inotify/max_queued_events   - 16384
# cat  /proc/sys/fs/inotify/max_user_instances  - 128
# cat  /proc/sys/fs/inotify/max_user_watches    - 16384
#
# -- Change to:
#
fs.inotify.max_queued_events=32768
fs.inotify.max_user_instances=256
fs.inotify.max_user_watches=32768
```

Especially, you will need to trim your tree or adjust the `max_user_watches` value if indexing exits with a message about errno `ENOSPC` (28) from `inotify_add_watch`.

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.

2.6 Fields and metadata

Apart from the main text content, documents usually aggregate other data elements, such as an author names, dates, abstracts, etc. These are usually called metadata elements because they qualify or describe the data rather than being part of it. Recoll has a slightly more general notion of `field` to mean any named piece of data associated with a document.

Fields are extracted by the document handlers when processing a document and further used by Recoll for searching or displaying results.

Some fields, like e.g. a file modification time, have a strict and predefined usage. For most fields though, the processing is entirely configurable and defined in the fields configuration file

Fields have two main processing options (at least one of which will be set if they are processed at all):

- Their content can be indexed. This makes them searchable.

- Their content can be stored in the index as document attribute data. This makes them displayable as part of a result list entry.

These options are preset in the default `fields` file for common elements like a title or an author name.

The terms from indexed fields are stored in the inverted index with a specific prefix, which makes them searchable by specifying the field name (e.g. `author:`*`Balzac`*). The terms can optionally also be used for the main index section to provide hits for non-prefixed searches. This is decided by an attribute in the `fields` file.

In most cases, field data is provided by the document itself, for example, by HTML `<meta>` elements. They can also be obtained from other sources, this is described in the following section.

2.6.1 Incorporating external metadata

2.6.1.1 Unix-like systems and MacOS systems: using extended attributes

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

Recoll processes all extended attributes as document fields. Note that most fields are not indexed by default, you need to activate them by defining a prefix in the fields configuration file.

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 of the same name.

On Linux, the `user` prefix is removed from the name.

The name translation can be configured more precisely, inside the fields configuration file.

Setting the document modification/creation date

Some documents have an internal date attribute (e.g. emails), but most get their date from the file modification time. It is possible to set a document date different from the file's by setting a specific extended attribute. For obscure and uninteresting reasons, the harcoded name of the attribute is `modificationdate`. Its contents should be the ASCII representation of a decimal integer representing the Unix time (seconds since the epoch). An example Linux command line for setting this particular field follow. The substituted **date** prints the example date parameter in Unix time format (seconds since the epoch).

```
setfattr -n user.modificationdate -v `date -d '2022-09-30 08:30:00' +%s` /some/ ↩
    file
```

The date substitution will then be automatic, you do not need to customize the `fields` file.

2.6.1.2 Using a command for importing external metadata

During indexing, it is possible to import metadata for each file by executing commands. This allows, for example, extracting tag data from an external application and storing it in a field for indexing.

See the section about the metadatacmds field in the main configuration chapter for a description of the configuration syntax.

For example, if you would want Recoll to use tags managed by tmsu in a field named *`tags`*, you would add the following to the configuration file:

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

Note the initial semi-colon after the equal sign.

You may want to restrict this processing to a subset of the directory tree, because it may slow down indexing a bit (`[some/area/of/th`

In the example above, the output of **tmsu** is used to set a field named *tags*. The field name is arbitrary and could be *tmsu* or *myfield* just the same, but *tags* is an alias for the standard Recoll `keywords` field, and the **tmsu** output will just augment its contents. This will avoid the need to extend the field configuration.

Note
Depending on the tmsu version, you may need/want to add options like `--database=/some/db`.

After setting or updating the parameter, you will need to tell Recoll to reindex the affected files. Just reset the index or see **recollindex** options `-e` or `-r`.

You will then be able to search the field from the query language: *tags:some/alternate/values* or *tags:all,these,values*.

Tags changes will not be detected by the indexer if the file itself did not change. One possible workaround would be to update the file `ctime` when you modify the tags, which would be consistent with how extended attributes function. A pair of **chmod** commands could accomplish this, or a `touch -a`. Alternatively, just couple the tag update with a `recollindex -e -i` */path/to/the/file*.

2.7 Miscellaneous indexing notes

2.7.1 Indexing punctuation characters (1.39)

By default, the Recoll indexer only uses most non-alphanumeric characters as separators, treating them as white space, so that inputs like *all words*, and *all,words* produce the same terms.

It may sometimes be useful to index some of these characters so that they can be used as discriminants for searches. This can be done by setting the `indexedpunctuation` configuration parameter. The value is an UTF-8 string, for example, setting:

```
indexedpunctuation = %€
```

would allow searching separately *100%* or *100€*.

The affected characters are indexed as terms with their own term positions, and they are their own separators, so that *100%* and *100 %* would be equivalent inputs.

2.7.2 The PDF input handler

The PDF format is very important for scientific and technical documentation, and document archival. It has extensive facilities for storing metadata along with the document, and these facilities are actually used in the real world.

In consequence, the **rclpdf.py** PDF input handler has more complex capabilities than most others, and it is also more configurable, because some extra features need executing external commands, so that they are not enabled by default. Specifically, **rclpdf.py** has the following optional features:

- It can extract PDF outlines and bookmarks.
- It can be configured to extract specific metadata tags from an XMP packet.
- It can extract PDF attachments.
- It can automatically perform OCR if the document text is empty. This is done by executing an external program and is now described in a separate section, because the OCR framework can also be used with non-PDF image files.

2.7.2.1 Extracting PDF outlines and bookmarks

These data elements will be extracted if `pdfoutline` is set in the configuration file and the **pdftohtml** command (from poppler-tools is available). Executing the command takes extra time, which is why the feature is not enabled by default.

2.7.2.2 XMP fields extraction

The `rclpdf.py` script in Recoll version 1.23.2 and later can extract XMP metadata fields by executing the **pdfinfo** command (usually found with poppler-utils). This is controlled by the pdfextrameta configuration variable, which specifies which tags to extract and, possibly, how to rename them.

The pdfextrametafix variable can be used to designate a file with Python code to edit the metadata fields (available for Recoll 1.23.3 and later. 1.23.2 has equivalent code inside the handler script). Example:

```
import sys
import re

class MetaFixer(object):
def __init__(self):
pass

def metafix(self, nm, txt):
if nm == 'bibtex:pages':
txt = re.sub(r'--', '-', txt)
elif nm == 'someothername':
# do something else
pass
elif nm == 'stillanother':
# etc.
pass

return txt
def wrapup(self, metaheaders):
pass
```

If the 'metafix()' method is defined, it is called for each metadata field. A new MetaFixer object is created for each PDF document (so the object can keep state for, for example, eliminating duplicate values). If the 'wrapup()' method is defined, it is called at the end of XMP fields processing with the whole metadata as parameter, as an array of '(nm, val)' pairs, allowing an alternate approach for editing or adding/deleting fields.

See this page for a more detailed discussion about indexing PDF XMP properties.

2.7.2.3 PDF attachment indexing

If pdftk is installed, and if the the pdfattach configuration variable is set, the PDF input handler will try to extract PDF attachments for indexing as sub-documents of the PDF file. This is disabled by default, because it slows down PDF indexing a bit even if not one attachment is ever found (PDF attachments are uncommon in my experience).

2.7.3 Running OCR on image documents

The Recoll PDF handler has the ability to call an external OCR program if the processed file has no text content. The OCR data is stored in a cache of separate files, avoiding any modification of the originals.

It must be noted that, if modifying the files (or a copy) is acceptable, then running something like OCRmyPDF to add a text layer to the PDF itself is a better solution (e.g. allowing Recoll to position the PDF viewer on the search target when opening the document, and permitting secondary search in the native tool).

To enable the Recoll OCR feature, you need to install one of the supported OCR applications (tesseract or ABBYY), enable OCR in the PDF handler (by setting `pdfocr` to `1` in the index configuration file), and tell Recoll how to run the OCR by setting

configuration variables. All parameters can be localized in subdirectories through the usual main configuration mechanism (path sections).

Example configuration fragment in `recoll.conf`:

```
pdfocr = 1
ocrprogs = tesseract
pdfocrlang = eng
```

This facility got a major update in Recoll 1.26.5. Older versions had a more limited, non-caching capability to execute an external OCR program in the PDF handler. The new function has the following features:

- The OCR output is cached, stored as separate files. The caching is ultimately based on a hash value of the original file contents, so that it is immune to file renames. A first path-based layer ensures fast operation for unchanged (unmoved files), and the data hash (which is still orders of magnitude faster than OCR) is only re-computed if the file has moved. OCR is only performed if the file was not previously processed or if it changed.
- The support for a specific program is implemented in a simple Python module. It should be straightforward to add support for any OCR engine with a capability to run from the command line.
- Modules initially exist for tesseract (Linux and Windows), and ABBYY FineReader (Linux, tested with version 11). ABBYY FineReader is a commercial closed source program, but it sometimes perform better than tesseract.
- The OCR is currently only called from the PDF handler, but there should be no problem using it for other image types.

2.7.4 Running a speech to text program on audio files

If the `OpenAI` Whisper program is available and the appropriate parameters set in the configuration files, the Recoll audio file handler will run speech to text recognition on audio files and the resulting text will be indexed. See the the FAQ entry for more details.

The results of the speech recognition will be cached in the same manner as the results of image OCR.

2.7.5 Removable volumes

Recoll used to have no support for indexing removable volumes (portable disks, USB keys, etc.). Recent versions have improved the situation and support indexing removable volumes in two different ways:

- By indexing the volume in the main, fixed, index, and ensuring that the volume data is not purged if the indexing runs while the volume is mounted. (since Recoll 1.25.2).
- By storing a volume index on the volume itself (since Recoll 1.24).

2.7.5.1 Indexing removable volumes in the main index

As of version 1.25.2, Recoll provides a simple way to ensure that the index data for an absent volume will not be purged. Two conditions must be met:

- The volume mount point must be a member of the `topdirs` list.
- The mount directory must be empty (when the volume is not mounted).

If **recollindex** finds that one of the `topdirs` is empty when starting up, any existing data for the tree will be preserved by the indexing pass (no purge for this area).

2.7.5.2 Self contained volumes

As of Recoll 1.24, it has become possible to build self-contained datasets including a Recoll configuration directory and index together with the indexed documents, and to move such a dataset around (for example copying it to an USB drive), without having to adjust the configuration for querying the index.

Note
This is a query-time feature only. The index must only be updated in its original location. If an update is necessary in a different location, the index must be reset.

The principle of operation is that the configuration stores the location of the original configuration directory, which must reside on the movable volume. If the volume is later mounted elsewhere, Recoll adjusts the paths stored inside the index by the difference between the original and current locations of the configuration directory.

To make a long story short, here follows a script to create a Recoll configuration and index under a given directory (given as single parameter). The resulting data set (files + recoll directory) can later to be moved to a CDROM or thumb drive. Longer explanations come after the script.

```
#!/bin/sh

fatal()
{
echo $*;exit 1
}
usage()
{
fatal "Usage: init-recoll-volume.sh <top-directory>"
}

test $# = 1 || usage
topdir=$1
test -d "$topdir" || fatal $topdir should be a directory

confdir="$topdir/recoll-config"
test ! -d "$confdir" || fatal $confdir should not exist

mkdir "$confdir"
cd "$topdir"
topdir=`pwd`
cd "$confdir"
confdir=`pwd`

(echo topdirs = '"'$topdir'"'; \
echo orgidxconfdir = $topdir/recoll-config) > "$confdir/recoll.conf"

recollindex -c "$confdir"
```

The examples below will assume that you have a dataset under `/home/me/mydata/`, with the index configuration and data stored inside `/home/me/mydata/recoll-confdir`.

In order to be able to run queries after the dataset has been moved, you must ensure the following:

- The main configuration file must define the orgidxconfdir variable to be the original location of the configuration directory (`orgidxconfdir=/home/me/mydata/recoll-confdir` must be set inside `/home/me/mydata/recoll-confdir/recoll.conf` in the example above).
- The configuration directory must exist with the documents, somewhere under the directory which will be moved. E.g. if you are moving `/home/me/mydata` around, the configuration directory must exist somewhere below this point, for example `/home/me/mydata/recoll-confdir`, or `/home/me/mydata/sub/recoll-confdir`.

- You should keep the default locations for the index elements which are relative to the configuration directory by default (principally `dbdir`). Only the paths referring to the documents themselves (e.g. `topdirs` values) should be absolute (in general, they are only used when indexing anyway).

Only the first point needs an explicit user action, the Recoll defaults are compatible with the third one, and the second is natural.

If, after the move, the configuration directory needs to be copied out of the dataset (for example because the thumb drive is too slow), you can set the curidxconfdir, variable inside the copied configuration to define the location of the moved one. For example if `/home/me/mydata` is now mounted onto `/media/me/somelabel`, but the configuration directory and index has been copied to `/tmp/tempconfig`, you would set `curidxconfdir` to `/media/me/somelabel/recoll-confdir` inside `/tmp/tempconfig/recoll.conf`. `orgidxconfdir` would still be `/home/me/mydata/recoll-confdir` in the original and the copy.

If you are regularly copying the configuration out of the dataset, it will be useful to write a script to automate the procedure. This can't really be done inside Recoll because there are probably many possible variants. One example would be to copy the configuration to make it writable, but keep the index data on the medium because it is too big - in this case, the script would also need to set `dbdir` in the copied configuration.

The same set of modifications (Recoll 1.24) has also made it possible to run queries from a readonly configuration directory (with slightly reduced function of course, such as not recording the query history).

2.7.6 Unix-like systems: indexing visited Web pages

With the help of a Firefox extension, Recoll can index the Internet pages that you visit. The extension has a long history: it was initially designed for the Beagle indexer, then adapted to Recoll and the Firefox XUL API. The current version of the extension is located in the Mozilla add-ons repository uses the WebExtensions API, and works with current Firefox versions.

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

The local cache is not an archive

As mentioned above, a copy of the indexed Web pages is retained by Recoll in a local cache (from which data is fetched for previews, or when resetting the index). The cache is not changed by an index reset, just read for indexing. The cache has a maximum size, which can be adjusted from the Index configuration / Web history panel (`webcachemaxmbs` parameter in `recoll.conf`). Once the maximum size is reached, old pages are erased to make room for new ones. The pages which you want to keep indefinitely need to be explicitly archived elsewhere. Using a very high value for the cache size can avoid data erasure, but see the above 'Howto' page for more details and gotchas.

The visited Web pages indexing feature can be enabled on the Recoll side from the GUI Index configuration panel, or by editing the configuration file (set `processwebqueue` to 1).

The Recoll GUI has a tool to list and edit the contents of the Web cache. (Tools → Webcache editor)

The **recollindex** command has two options to help manage the Web cache:

- `--webcache-compact` will recover the space from erased entries. It may need to use twice the disk space currently needed for the Web cache.
- `--webcache-burst` *`destdir`* will extract all current entries into pairs of metadata and data files created inside *`destdir`*

You can find more details on Web indexing, its usage and configuration in a Recoll 'Howto' entry.

Chapter 3

Searching

3.1 Introduction

Getting answers to specific queries is of course the whole point of Recoll. The multiple provided interfaces always understand simple queries made of one or several words, and return appropriate results in most cases.

In order to make the most of Recoll though, it may be worthwhile to understand how it processes your input. Five different modes exist:

- In `All Terms` mode, Recoll looks for documents containing all your input terms.
- The `Query Language` mode behaves like `All Terms` in the absence of special input, but it can also do much more. This is the best mode for getting the most of Recoll. It is usable from all possible interfaces (GUI, command line, WEB UI, ...), and is described here.
- In `Any Term` mode, Recoll looks for documents containing any your input terms, preferring those which contain more.
- In `File Name` mode, Recoll will only match file names, not content. Using a small subset of the index allows things like left-hand wildcards without performance issues, and may sometimes be useful.
- The GUI `Advanced Search` mode is actually not more powerful than the query language, but it helps you build complex queries without having to remember the language, and avoids any interpretation ambiguity, as it bypasses the user input parser.

These five input modes are supported by the different user interfaces which are described in the following sections.

3.2 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 interfaces:

- Simple search (the default, on the main screen) has a single entry field where you can enter multiple words or a query language query.
- 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.

The Advanced Search tool is easier to use, but not actually more powerful, than the Simple Search in query language mode. Its name is historical, but Assisted Search would probably have been a better designation.

In most text areas, you can enter the terms as you think them, even if they contain embedded punctuation or other non-textual characters (e.g. Recoll can handle things like email addresses).

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).

Some searches can be quite complex, and you may want to re-use them later, perhaps with some tweaking. Recoll can save and restore searches. See Saving and restoring queries.

3.2.1 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.

Any term will search for documents where at least one of the terms appear.

File name will exclusively look for file names, not contents

All search modes allow terms to be expanded with wildcards characters (`*`, `?`, `[]`). See the section about wildcards for more details.

In all modes except File name, you can search for exact phrases (adjacent words in a given order) by enclosing the input inside double quotes. Ex: `"virtual reality"`.

The Query Language features are described in a separate section.

When using a stripped index (the default), character case has no influence on search, except that you can disable stem expansion for any term by capitalizing it. E.g.: 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 directly access the search history by clicking the clock button on the right of the search entry, while the latter is empty. Otherwise, the history is used for entry completion (see next). Only the search texts are remembered, not the mode (all/any/file name).

While text is entered in the search area, **recoll** will display possible completions, filtered from the history and the index search terms. This can be disabled with a GUI Preferences option.

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.

The File name search mode will specifically look for file names. The point of having a separate file name search is that wildcard expansion can be performed more efficiently on a small subset of the index (allowing wildcards on the left of terms without excessive cost). 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, independently of the type of index.
- An entry without any wildcard character and not capitalized will be prepended and appended with '*' (e.g.: *etc* -> **etc**, but *Etc* -> *etc*).
- If you have a big index (many files), excessively generic fragments may result in inefficient searches.

3.2.2 The result list

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

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

Each result is displayed as a structured text paragraph. The standard format is typically adequate, but the content and presentation are entirely customisable.

Most results will contain `Preview` and `Open` clickable links.

Clicking the `Preview` link 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. See further for customizing the applications.

The `Preview` and `Open` links may not be present for all entries. They are only available, respectively, for documents with MIME types that Recoll can extract text from, and for documents that have a configured viewer. However, you can modify the configuration to adjust this behavior. In more detail:

- The `Preview` link will appear for documents with a MIME type present in the `[index]` section of the mimeconf file, and, only if the `textunknownasplain` configuration variable is set, for all types identified as a subtype of `text` (`text/*`).
- The `Open` link will appear for documents with a MIME type present in the `[view]` section of the mimeview configuration file. If `textunknownasplain` is set and no specific viewer is found for a subtype of `text`, the viewer for `text/plain` will be used.

You can click on the `Query details` link at the top of the results page to see the actual Xapian query, 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. You can change the page size in the preferences. Use the arrow buttons in the toolbar or the links at the bottom of the page to browse the results.

3.2.2.1 Customising the viewers

By default Recoll lets the desktop choose what application should be used to open a given document, with exceptions.

The details of this behaviour can be customized with the Preferences → GUI configuration → User interface → Choose editor applications dialog or by editing the mimeview configuration file.

When Use desktop preferences, at the top of the dialog, is checked, the desktop default is generally used, but there is a small default list of exceptions, for MIME types where the Recoll choice should override the desktop one. These are applications which are well integrated with Recoll, for example, on Linux, 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. You can add or remove document types to the exceptions by using the dialog.

If you prefer to completely customize the choice of applications, you can uncheck Use desktop preferences, in which case the Recoll predefined applications will be used, and can be changed for each document type. This is probably not the most convenient approach in most cases.

In all cases, the applications choice dialog accepts multiple selections of MIME types in the top section, and lets you define how they are processed in the bottom one. In most cases, you will be using `%f` as a place holder to be replaced by the file name in the application command line.

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

Under Unix-like systems, each result list entry also has a right-click menu with an Open With entry. This lets you choose an application from the list of those which registered with the desktop for the document MIME type, on a case by case basis.

3.2.2.2 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.

3.2.2.3 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
- Open With
- Run Script
- 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.

Open With (Unix-like systems) lets you open the document with one of the applications claiming to be able to handle its MIME type (the information comes from the `.desktop` files in `/usr/share/applications`).

Run Script allows starting an arbitrary command on the result file. It will only appear for results which are top-level files. See further for a more detailed description.

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 (e.g.: an email attachment). It is especially useful to extract attachments with no associated editor.

The Open/Preview Parent document entries allow working with the higher level document (e.g. the email message an attachment comes from). Recoll is sometimes not totally accurate as to what it can or can't do in this area. For example the Parent entry will also appear for an email which is part of an mbox folder file, but you can't actually visualize the mbox (there will be an error dialog if you try).

If the document is a top-level file, Open Parent will start the default file manager on the enclosing filesystem directory.

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. I can't remember a single instance where this function was actually useful to me...

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.

3.2.3 The result table

As an alternative to the result list, the results can also 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.

Using Shift-click on a row will display the document extracted text (somewhat like a preview) instead of the document details. The functions of Click and Shift-Click can be reversed in the GUI preferences.

3.2.4 The filters panel

By default, the GUI displays the filters panel on the left of the results area. This is new in version 1.32. You can adjust the width of the panel, and hide it by squeezing it completely. The width will be memorized for the next session.

The panel currently has two areas, for filtering the results by dates, or by filesystem location.

The panel is only active in Query Language search mode, and its effect is to add `date:` and `dir:` clauses to the actual search.

The dates filter can be activated by clicking the checkbox. It has two assisted date entry widgets, for the minimum and maximum dates of the search period.

The directory filter displays a subset of the filesystem directories, reduced to the indexed area, as defined by the `topdirs` list and the name exclusion parameters. Some directories may not be shown at all, depending on their (lack of) indexable content and other indexing parameters. The depth of the displayed tree is limited at 2 levels under the start directories by default. You can change this in the GUI Preferences, User interface panel.

You can independantly select and deselect directories by clicking them. Note that selecting a directory will activate the whole subtree for searching, there is no need to select the subdirectories, and no way to exclude some of them (use Query language `dir:` clauses if this is needed).

3.2.5 Running arbitrary commands on result files

Apart from the Open and Open With operations, which allow starting an application on a result document (or a temporary copy), based on its MIME type, it is also possible to run arbitrary commands on results which are top-level files, using the Run Script entry in the results pop-up menu.

The commands which will appear in the Run Script submenu must be defined by `.desktop` files inside the `scripts` subdirectory of the current configuration directory.

Here follows an example of a `.desktop` file, which could be named for example, `~/.recoll/scripts/myscript.desktop` (the exact file name inside the directory is irrelevant):

```
[Desktop Entry]
Type=Application
Name=MyFirstScript
Exec=/home/me/bin/tryscript %F
MimeType=*/*
```

The `Name` attribute defines the label which will appear inside the Run Script menu. The `Exec` attribute defines the program to be run, which does not need to actually be a script, of course. The `MimeType` attribute is not used, but needs to exist.

The commands defined this way can also be used from links inside the result paragraph.

As an example, it might make sense to write a script which would move the document to the trash and purge it from the Recoll index.

3.2.6 Unix-like systems: 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 in the Recoll FAQ.

3.2.7 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, or using the window manager button in the top of the frame will also close the window.

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 (e.g.: 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.

3.2.7.1 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 occurrences 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.

3.2.8 The Query Fragments window

The Query Fragments window can be used to control filtering query language elements modifying the current query, simply by clicking a button. This can be useful to save typing, or avoid memorizing, simple clauses of common usage (e.g. selecting only standalone documents or attachments, or filtering out WEB results, selecting a file system subtree, a file type, etc.).

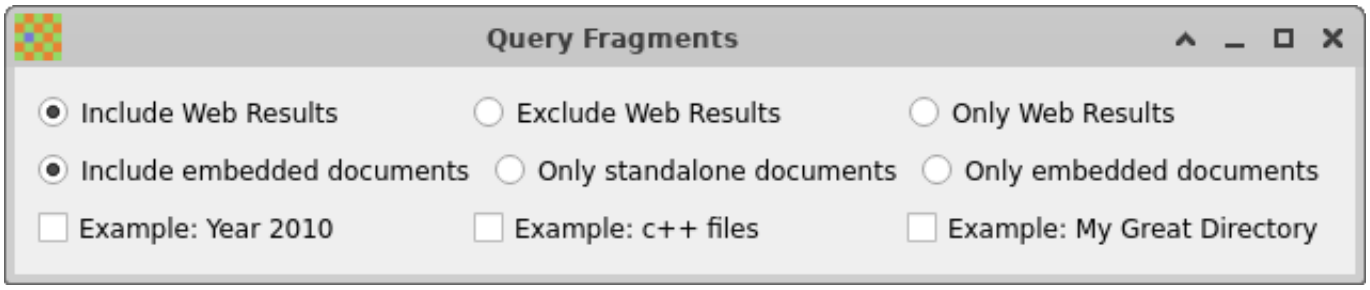
Selecting the Tools → Query Fragments menu entry will open the dialog.

The contents of the window are entirely customizable, and defined by the contents of a XML text file, named `fragment-buttons.xml` and which will be looked for in the current index configuration directory. The sample file distributed with Recoll contains a number of example filters. This will be automatically copied to the configuration directory if the file does not exist in there (e.g. `~/.recoll/fragment-buttons.xml` under Linux and MacOS, `$HOME/AppData/Local/Recoll/fragment-buttons.xml` for Windows). Editing the copy will allow you to configure the tool for your needs .

Note

The `fragment-buttons.xml` file was named `fragbuts.xml` up to Recoll version 1.31.0. This was deemed too close to offensive for native English speakers, so that the file was renamed. An existing `fragbuts.xml` will still be used if `fragment-buttons.xml` does not exist. No automatic renaming will be performed.

Here follows an example window:



And the corresponding configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<fragbuttons version="1.0">

  <radiobuttons>
    <!-- Toggle WEB queue results inclusion -->
    <fragbutton>
      <label>Include Web Results</label>
      <frag></frag>
    </fragbutton>
    <fragbutton>
      <label>Exclude Web Results</label>
      <frag>-rclbes:BGL</frag>
    </fragbutton>
    <fragbutton>
      <label>Only Web Results</label>
      <frag>rclbes:BGL</frag>
    </fragbutton>
  </radiobuttons>

  <radiobuttons>
    <!-- Standalone vs embedded switch -->
    <fragbutton>
      <label>Include embedded documents</label>
      <frag></frag>
    </fragbutton>
    <fragbutton>
      <label>Only standalone documents</label>
```

```
      <frag>issub:0</frag>
    </fragbutton>
    <fragbutton>
      <label>Only embedded documents</label>
      <frag>issub:1</frag>
    </fragbutton>
  </radiobuttons>

  <buttons>
    <fragbutton>
      <label>Example: Year 2010</label>
      <frag>date:2010-01-01/2010-12-31</frag>
    </fragbutton>
    <fragbutton>
      <label>Example: c++ files</label>
      <frag>ext:cpp OR ext:cxx</frag>
    </fragbutton>
    <fragbutton>
      <label>Example: My Great Directory</label>
      <frag>dir:/my/great/directory</frag>
    </fragbutton>
  </buttons>
</fragbuttons>
```

There are two types of groupings `radiobuttons` and `buttons`, each defining a line of checkbuttons or radiobuttons inside the window. Any number of `buttons` can be selected, but the `radiobuttons` in a line are exclusive.

Buttons are defined by a `fragbutton` section, which provides the label for a button, and the `Query Language` fragment which will be added (as an AND filter) before performing the query if the button is active.

```
    <fragbutton>
      <label>Example: My Great Directory</label>
      <frag>dir:/my/great/directory</frag>
    </fragbutton>
```

It is also possible to add `message` elements inside the groups, for documenting the behaviour. `message` elements have a `label` but no `frag` element. Example:

```
  <buttons>
    <message>
      <label>This is a message</label>
    </message>
  </buttons>
```

The `label` contents are interpreted as HTML. Take care to replace opening < characters with the `<` entity if you use tags.

The only thing that you need to know about XML for editing this file is that any opening tag like *<label>* needs to be matched by a closing tag after the value: *</label>*.

You will normally edit the file with a regular text editor, like, e.g. **vi** or **notepad**. Double-clicking the file in a file manager may not work, because this usually opens it in a WEB browser, which will not let you modify the contents.

3.2.9 Assisted Complex Search (A.K.A. "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.

Recoll keeps a history of searches. See Advanced search history.

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 allows filtering 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.

3.2.9.1 Advanced search: the "find" tab

This part of the dialog lets you construct 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.

3.2.9.1.1 Phrase and Proximity searches

These two clauses look for a group of terms in specified relative positions. They differ in the sense that the order of input terms is significant for `phrase` searches, but not for `proximity` searches. The latter 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. For `phrase` searches, the default count is zero (exact match). For `proximity` searches 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`.

The slack can be adjusted with the counter to the left of the input area

3.2.9.2 Advanced 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 10E3, 10E6, 10E9, 10E12 respectively.

- The next section allows filtering the results by their MIME types, or MIME categories (e.g.: 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. E.g., entering `dirA/dirB` would match either `/dir1/dirA/dirB/myfile1` or `/dir2/dirA/dirB/someother/myfile2`.

3.2.9.3 Advanced 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.

3.2.10 The term explorer tool

Recoll automatically manages the expansion of search terms to their derivatives (e.g.: 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 search will only propose replacement terms with spelling variations when no matching document were found. In some cases, both proper spellings and mispellings are present in the index, and it may be interesting to look for them explicitly.

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, or (later addition), display some statistics or other index information. It has several modes of operations:

Wildcard In this mode of operation, you can enter a search string with shell-like wildcards (*, ?, []). e.g.: *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 implicitly anchored at the beginning. E.g.: *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.

Show index statistics This will print a long list of boring numbers about the index

List files which could not be indexed This will show the files which caused errors, usually because **recollindex** could not translate their format into text.

Note that in cases where Recoll does not know the beginning of the string to search for (e.g. 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).

3.2.11 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 main index, which is the one to be updated when requested from the File menu, but it can use any number of external Recoll indexes for searching. The external indexes can be selected through the External Indexes tab in the preferences dialog, which can be reached either trough: Preferences → GUI Configuration → External Index Dialog or Query → External index 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.

If the external index is actually located on a volume mounted from another machine, and references remote files, there may be a need to adjust the result paths so that they match the locally mounted ones (for opening documents). This can be done by using the path translation facility.

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, e.g.:

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

On Windows, use semi-colons (`;`) as separators instead of colons.

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**. Unreachable indexes will automatically be deactivated when starting up.

3.2.12 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.

3.2.13 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 e.g., 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.

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.

3.2.14 Keyboard shortcuts

A number of common actions within the graphical interface can be triggered through keyboard shortcuts. As of Recoll 1.29, many of the shortcut values can be customised from a screen in the GUI preferences. Most shortcuts are specific to a given context (e.g. within a preview window, within the result table).

Most shortcuts can be changed to a preferred value by using the GUI shortcut editor: Preferences → GUI configuration → Shortcuts. In order to change a shortcut, just click the corresponding cell in the Shortcut column, and type the desired sequence.

3.2.15 Search tips

3.2.15.1 Terms and search expansion

Term completion While typing into the simple search entry, a popup menu will appear and show completions for the current string. Values preceded by a clock icon come from the history, those preceded by a magnifier icon come from the index terms. This can be disabled in the preferences.

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 `gardening` 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.

3.2.15.2 Working with phrases and proximity

Phrases searches A phrase can be looked for by enclosing a number of terms in double quotes. Example: `"user manual"` will look only for occurrences of `user` immediately followed by `manual`. You can use the "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 "Phrase".

Proximity searches A proximity search differs from a phrase search in that it does not impose an order on the terms. Proximity searches can be entered by specifying the "Proximity" type in the advanced search, or by postfixing a phrase search with a 'p'. Example: "user manual"p would also match "manual user". Also see the modifier section from the query language documentation.

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. E.g.: 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 slow down a query if most of the terms in the phrase are common. If the `autophrase` option is on, very common terms will be removed from the automatically constructed phrase. The removal threshold can be adjusted from the search preferences.

Phrases and abbreviations Dotted abbreviations like `I.B.M.` are also automatically indexed as a word without the dots: `IBM`. Searching for the word inside a phrase (e.g.: `"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 occurrences of the word will be matched normally.

Description	Default value
Context: almost everywhere	
Program exit	Ctrl+Q
Context: advanced search	
Load the next entry from the search history	Up
Load the previous entry from the search history	Down
Context: main window	
Clear search. This will move the keyboard cursor to the simple search entry and erase the current text	Ctrl+S
Move the keyboard cursor to the search entry area without erasing the current text	Ctrl+L
Move the keyboard cursor to the search entry area without erasing the current text	Ctrl+Shift+S
Toggle displaying the current results as a table or as a list	Ctrl+T
Context: main window, when showing the results as a table	
Move the keyboard cursor to currently the selected row in the table, or to the first one if none is selected	Ctrl+R
Jump to row 0-9 or a-z in the table	Ctrl+[0-9] or Ctrl+Shift+[a-z]
Cancel the current selection	Esc
Context: preview window	
Close the preview window	Esc
Close the current tab	Ctrl+W
Open a print dialog for the current tab contents	Ctrl+P
Load the next result from the list to the current tab	Shift+Down
Load the previous result from the list to the current tab	Shift+Up
Context: result table	
Copy the text contained in the selected document to the clipboard	Ctrl+G
Copy the text contained in the selected document to the clipboard, then exit recoll	Ctrl+Alt+Shift+G
Open the current document	Ctrl+O
Open the current document and exit Recoll	Ctrl+Alst+Shift+O
Show a full preview for the current document	Ctrl+D
Toggle showing the column names	Ctrl+H
Show a snippets (keyword in context) list for the current document	Ctrl+E
Toggle showing the row letters/numbers	Ctrl+V
Context: snippets window	
Close the snippets window	Esc
Find in the snippets list (method #1)	Ctrl+F
Find in the snippets list (method #2)	/
Find the next instance of the search term	F3
Find the previous instance of the search term	Shift+F3

Table 3.1: Keyboard shortcuts

3.2.15.3 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:helpfulgui`

Adjusting 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.

Changing the GUI geometry It is possible to configure the GUI in wide form factor by dragging the toolbars to one of the sides (their location is remembered between sessions), and moving the category filters to a menu (can be set in the Preferences → GUI configuration → User interface panel).

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 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.

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.

3.2.16 Saving and restoring queries

Both simple and advanced query dialogs save recent history, but the amount is limited: old queries will eventually be forgotten. Also, important queries may be difficult to find among others. This is why both types of queries can also be explicitly saved to files, from the GUI menus: File → Save last query / Load last query

The default location for saved queries is a subdirectory of the current configuration directory, but saved queries are ordinary files and can be written or moved anywhere.

Some of the saved query parameters are part of the preferences (e.g. `autophrase` or the active external indexes), and may differ when the query is loaded from the time it was saved. In this case, Recoll will warn of the differences, but will not change the user preferences.

3.2.17 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 (e.g. `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.

- Activate links in preview if set, Recoll will turn HTTP links found inside plain text into proper HTML anchors, and clicking a link inside a preview window will start the default browser on the link target.
- 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.
- Choose editor application: this opens a dialog which allows you to select the application to be used to open each MIME type. The default is to use the **xdg-open** utility, but you can use this dialog to override it, setting exceptions for 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.
- Disable Qt autocompletion in search entry: this will disable the completion popup. Il will only appear, and display the full history, either if you enter only white space in the search area, or if you click the clock button on the right of the area.
- Document filter choice style: this will let you choose if the document categories are displayed as a list or a set of buttons, or a menu.
- Start with simple search mode: this lets you choose the value of the simple search type on program startup. Either a fixed value (e.g. `Query Language`, or the value in use when the program last exited.
- 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 (e.g.: `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 (e.g.: `/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,

3.2.17.1 The result list format

Recoll normally uses a full function HTML processor to display the result list and the snippets window. Depending on the version, this may be based on either Qt WebKit or Qt WebEngine. It is then possible to completely customise the result list with full support for CSS and Javascript.

It is also possible to build Recoll to use a simpler Qt QTextBrowser widget to display the HTML, which may be necessary if the ones above are not ported on the system, or to reduce the application size and dependencies. There are limits to what you can do in this case, but it is still possible to decide what data each result will contain, and how it will be displayed.

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

- The paragraph format
- HTML code inside the header section. This is also used for the snippets window.

The paragraph format and the header fragment can be edited from the Result list tab of the GUI configuration.

The header fragment is used both for the result list and the snippets window. The snippets list is a table and has a `snippets` class attribute. Each paragraph in the result list is a table, with class `respar`, but this can be changed by editing the paragraph format.

There are a few examples on the page about customising the result list on the Recoll Web site.

3.2.17.1.1 The paragraph format

This is an arbitrary HTML string which will be transformed by printf-like `%` substitutions to show the results.

Note
Any literal `%` character in the input must be quoted as `%%`. E.g. `<table style="width: 100%;">` should be entered as `<table style="width: 100%%;">`.

The following substitutions will be performed:

%A Abstract. If `%s` is not present, this will be either the document `abstract` attribute if one is present, or the synthetic snippets abstract. If `%s` is present, this will be the document abstract or empty.

%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.

%L Precooked Preview, Edit, and possibly Snippets links.

%M MIME type.

%N result Number inside the result page.

%P Parent folder Url. In the case of an embedded document, this is the parent folder for the top level container file.

%R Relevance percentage.

%S Size information.

%s Synthetic "snippets" abstract (selected text around search terms found in the document.

%T Title if this is set, else Filename.

%t Title or empty.

%(filename) File name.

%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 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 input handlers.

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).

A link target defined as `"F%N"` will open the document corresponding to the `%P` parent folder expansion, usually creating a file manager window on the folder where the container file resides. E.g.:

```
<a href="F%N">%P</a>
```

A link target defined as `R%N|`*scriptname* will run the corresponding script on the result file (if the document is embedded, the script will be started on the top-level parent). See the section about defining scripts. Note that *scriptname* value should be the value of the `Name` field of the desktop file, and not the desktop file name.

The default value for the paragraph format string is:

```
"<table class=\"respar\">\n"
"<tr>\n"
"<td><a href='%U'><img src='%I' width='64'></a></td>\n"
"<td>%L  <i>%S</i>   <b>%T</b><br>\n"
"<span style='white-space:nowrap'><i>%M</i> %D</span>    <i ↩
    >%U</i> %i<br>\n"
"%s %A %K</td>\n"
"</tr></table>\n"
```

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.

3.2.18 The recoll GUI command line options

The **recoll** command has a number of useful command line options.

`-c` *`configdir`* specifies a non-default configuration directory.

`-L` *`lang`* can be used to use a different language for the GUI labels than the one which would be chosen according to the system locale. Some translations are quite incomplete and you may prefer to see the English messages, even if your machine is generally setup for, e.g. Spanish. Example:

```
recoll -L en
```

`-q` *`query`* specifies a query to be run when the program starts. It takes a single argument, which must be quoted if it contains white space.

`-o/-l/-f/-a` specify the type of query. The default is to interpret the `-q` argument as a query language string. You can use these options to interpret the argument as an `Any Term`, `File Name` or `All Terms` query instead.

The `-t` option will tell the program to behave exactly like the **recollq** command, printing the results to the standard output (terminal) instead of starting a graphical window.

3.3 Searching with the KDE KIO slave

The Recoll KIO slave allows performing a Recoll search by entering an appropriate URL in a KDE open dialog, or a **Dolphin** URL. The results are displayed as directory entries.

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.

3.4 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, or by calling it as **recollq** (through a link).
- By using the actual **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 always 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. Apart from being easily customised, **recollq** is only really useful on systems where the Qt libraries are not available, else it is redundant with `recoll -t`.

recollq has a man page. The Usage string 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 elements:
   * 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.
 -C : collapse duplicates
 -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.
    -p <cnt> : show <cnt> snippets, with page numbers instead of the concatenated abstract.
    -g <cnt> : show <cnt> snippets, with line numbers instead of the concatenated abstract.
 -S fld : sort by field <fld>.
   -D : sort descending.
 -s stemlang : set stemming language to use (must exist in index...).
    Use -s "" to turn off stem expansion.
 -T <synonyms file>: use the parameter (Thesaurus) for word expansion.
 -i <dbdir> : additional index, several can be given.
 -e use url encoding (%xx) for urls.
 -E use exact result count instead of lower bound estimate.
 -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. Use -F '' to output all fields, but you probably
    also want option -N in this case.
   -N : with -F, print the (plain text) field names before the field values.
 --extract_to <filepath> : extract the first result to filepath, which must not exist.
      Use a -n option with an offset to select the appropriate result.
```

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/dockes/projets/bateaux/ilur/comptes.html]      [comptes.html ↩
    ]  18593   bytes
```

```
text/html       [file:///Users/dockes/projets/nautique/webnautique/articles/ilur1/index. ↩
    html] [Constructio...
text/html       [file:///Users/dockes/projets/pagepers/index.html]    [psxtcl/writemime/ ↩
    recoll]...
text/html       [file:///Users/dockes/projets/bateaux/ilur/factEtCie/recu-chasse-maree....
```

3.5 The query language

The Recoll query language was based on the now defunct Xesam user search language specification. It allows defining general boolean searches within the main body text or specific fields, and has many additional features, broadly equivalent to those provided by *complex search* interface in the GUI.

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 from the command line search, the KIO slave, or the WEB UI.

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.

3.5.1 General syntax

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, e.g.: 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). Examples:

- *Eugenie*
- *author:balzac*
- *dc:title:grandet*
- *dc:title:"eugenie 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*.

You can use parentheses to group elements (from version 1.21), which will sometimes make things clearer, and may allow expressing combinations which would have been difficult otherwise.

An element preceded by a − specifies a term that should *not* appear.

By default, words inside double-quotes define a `phrase` search (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. This can be changed by using modifiers.

Words inside phrases and capitalized words are not stem-expanded. Wildcards may be used anywhere inside a term. Specifying a wildcard 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.

To save you some typing, Recoll versions 1.20 and later interpret a field value given as a comma-separated list of terms as an AND list and a slash-separated list as an OR list. No white space is allowed. So

```
author:john,lennon
```

will search for documents with `john` and `lennon` inside the `author` field (in any order), and

```
author:john/ringo
```

would search for `john` or `ringo`. This behaviour is only triggered by a field prefix: without it, comma- or slash- separated input will produce a phrase search. However, you can use a `text` field name to search the main text this way, as an alternate to using an explicit `OR`, e.g. `text:napoleon/bonaparte` would generate a search for *napoleon* or *bonaparte* in the main text body.

Modifiers can be set on a double-quote value, for example to specify a proximity search (unordered). See the modifier section. No space must separate the final double-quote and the modifiers value, e.g. *"two one"po10*

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. You can use the shorter `fn` alias. This value is not set for all documents: internal documents contained inside a compound one (for example an EPUB section) do not inherit the container file name any more, this was replaced by an explicit field (see next). Sub-documents can still have a `filename`, if it is implied by the document format, for example the attachment file name for an email attachment.
- `containerfilename`, aliased as `cfn`. This is set for all documents, both top-level and contained sub-documents, and is always the name of the filesystem file which contains the data. The terms from this field can only be matched by an explicit field specification (as opposed to terms from `filename` which are also indexed as general document content). This avoids getting matches for all the sub-documents when searching for the container file name.
- `ext` specifies the file name extension (Ex: `ext:html`).
- `rclmd5` the MD5 checksum for the document. This is used for displaying the duplicates of a search result (when querying with the option to collapse duplicate results). Incidentally, this could be used to find the duplicates of any given file by computing its MD5 checksum and executing a query with just the `rclmd5` value.

You can define aliases for field names, in order to use your preferred denomination or to save typing (e.g. the predefined `fn` and `cfn` aliases defined for `filename` and `containerfilename`). See the section about the fields file.

The document input handlers 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.

3.5.2 Special field-like specifiers

The field syntax also supports a few field-like, but special, criteria, for which the values are interpreted differently. Regular processing does not apply (for example the slash- or comma- separated lists don't work). A list follows.

- `dir` for filtering the results on file location. For example, `dir:/home/me/somedir` will restrict the search to results found anywhere under the */home/me/somedir* directory (including subdirectories).

  Tilde expansion will be performed as usual. Wildcards will be expanded, but please have a look at an important limitation of wildcards in path filters.

  You can also use relative paths. For example, `dir:share/doc` would match either `/usr/share/doc` or `/usr/local/share/doc`.

  `-dir` will find results *not* in the specified location.

  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.

On Unix-like systems, 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.

The shortcut syntax to define OR or AND lists within fields with commas or slash characters is not available.

- `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 `P`*`n`*`Y`*`n`*`M`*`n`*`D`. 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 (e.g.: p2y).
- `mime` or `format` for specifying the MIME type. These clauses are processed apart from the normal Boolean logic of the search: multiple values will be OR'ed (instead of the normal AND). You can specify types to be excluded, with the usual `-`, and use wildcards. Example: *`mime:text/* -mime:text/plain`*. Specifying an explicit boolean operator before a `mime` specification is not supported and will produce strange results.
- `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, and can be negated with `-`.
- `issub` for specifying that only standalone (`issub:0`) or only embedded (`issub:1`) documents should be returned as results.

Note

`mime`, `rclcat`, `size`, `issub` and `date` criteria always affect the whole query (they are applied as a final filter), even if set with other terms inside a parenthesis.

Note

`mime` (or the equivalent `rclcat`) is the *only* field with an `OR` default. You do need to use `OR` with `ext` terms for example.

3.5.3 Range clauses

Recoll 1.24 and later support range clauses on fields which have been configured to support it. No default field uses them currently, so this paragraph is only interesting if you modified the fields configuration and possibly use a custom input handler.

A range clause looks like one of the following:

```
myfield:small..big
myfield:small..
myfield:..big
```

The nature of the clause is indicated by the two dots `..`, and the effect is to filter the results for which the *myfield* value is in the possibly open-ended interval.

See the section about the fields configuration file for the details of configuring a field for range searches (list them in the [values] section).

3.5.4 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, but see also `x` further down).
- `o` can be used to specify a "slack" for both `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. The default slack (with no `o`) is 0 for `phrase` searches and 10 for `proximity` searches.
- `p` can be used to turn an ordered `phrase` search into an unordered `proximity` one. Example: `"order any in"p`. You can find a little more detail about `phrase` and `proximity` searches here.
- `s` (1.22) can be used to turn off synonym expansion, if a synonyms file is in place.
- `x` (1.33.2) will enable the expansion of terms inside a phrase search (the default is for phrases to be searched verbatim). Also see the stemexpandphrases in the configuration section, for changing the default behaviour.
- A weight can be specified for a query element by specifying a decimal value at the start of the modifiers. Example: `"Important"2.5`

The following only make sense on indexes which are capable of case and diacritics sensitivity (not the default):

- `C` will turn on case sensitivity.
- `D` will turn on diacritics sensitivity (if the index supports it).
- `e` (explicit) will turn on diacritics sensitivity and case sensitivity, and prevent stem expansion.

3.6 Wildcards and anchored searches

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.

3.6.1 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).

3.6.1.1 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 containing 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).

3.6.2 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, for example for looking for matches inside the abstract or the list of authors (which occur at the top of the document).

3.7 Using Synonyms (1.22)

Term synonyms and text search: in general, there are two main ways to use term synonyms for searching text:

- At index creation time, they can be used to alter the indexed terms, either increasing or decreasing their number, by expanding the original terms to all synonyms, or by reducing all synonym terms to a canonical one.
- At query time, they can be used to match texts containing terms which are synonyms of the ones specified by the user, either by expanding the query for all synonyms, or by reducing the user entry to canonical terms (the latter only works if the corresponding processing has been performed while creating the index).

With one exception, Recoll only uses synonyms at query time. A user query term which part of a synonym group will be optionally expanded into an `OR` query for all terms in the group.

The one exception is that if the idxsynonyms parameter is set during indexing, and if the file contains multi-word synonyms, a multi-word single term will be emitted for every occurrence found in the text. If the same file is in use at query time, this will allow phrase and proximity searches to work for the multi-word synonyms.

Synonym groups are defined inside ordinary text files. Each line in the file defines a group.

Example:

```
hi hello "good morning"

# not sure about "au revoir" though. Is this english ?
bye goodbye "see you" \
"au revoir"
```

As usual, lines beginning with a # are comments, empty lines are ignored, and lines can be continued by ending them with a backslash.

Multi-word synonyms are supported, but be aware that these will generate phrase queries, which may degrade performance and will disable stemming expansion for the phrase terms.

The contents of the synonyms file must be casefolded (not only lowercased), because this is what expected at the point in the query processing where it is used. There are a few cases where this makes a difference, for example, German sharp s should be expressed as `ss`, Greek final sigma as sigma. For reference, Python3 has an easy way to casefold words (str.casefold()).

The synonyms file can be specified in the Search parameters tab of the GUI configuration Preferences menu entry, or as an option for command-line searches.

Once the file is defined, the use of synonyms can be enabled or disabled directly from the Preferences menu.

The synonyms are searched for matches with user terms after the latter are stem-expanded, but the contents of the synonyms file itself is not subjected to stem expansion. This means that a match will not be found if the form present in the synonyms file is not present anywhere in the document set (same with accents when using a raw index).

The synonyms function is probably not going to help you find your letters to Mr. Smith. It is best used for domain-specific searches. For example, it was initially suggested by a user performing searches among historical documents: the synonyms file would contains nicknames and aliases for each of the persons of interest.

3.8 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 example 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. In some case, the path adjustments can be automated.
- As a last example, one could imagine that a big directory has been moved, but that it is currently inconvenient to run the indexer.

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 with a plain text editor or by using the GUI external indexes configuration dialog: Preferences → External index dialog, then click the Paths translations button on the right below the index list: translations will be set for the main index if no external index is currently selected in the list, or else for the currently selected index.

Example entry from a ptrans file:

```
[/path/to/external/xapiandb]
/some/index/path = /some/local/path
```

This would decide that, for the index stored in `/path/to/external/xapiandb`, any occurence of `/some/index/path` should be replaced with `/some/local/path` when presenting a result.

Windows note

At the moment, the path comparisons done for path translation under MS Windows are case sensitive (this will be fixed at some point). Use the natural character case as displayed in the file explorer. Example:

```
[Z:/some/mounted/xapiandb]
C: = Z:
```

3.9 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 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 entered without upper-case or accented characters 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 (they were documented but actually did nothing until Recoll 1.22):

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 explicitly 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.

3.10 Desktop integration

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

- Users of recent Ubuntu-derived distributions, or any other Gnome desktop systems (e.g. Fedora) can install the Recoll GSSP (Gnome Shell Search Provider).
- The KDE KIO Slave was described in a previous section. It can provide search results inside **Dolphin**.
- If you use an oldish version of Ubuntu Linux, you may find the Ubuntu Unity Lens module useful.
- There is also an independently developed Krunner plugin. It is now integrated in the Recoll source.
- 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.
- 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 input handlers for new types of documents.

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

4.1 Writing a document input handler

Terminology
The small programs or pieces of code which handle the processing of the different document types for Recoll used to be called `filters`, which is still reflected in the name of the directory which holds them and many configuration variables. They were named this way because one of their primary functions is to filter out the formatting directives and keep the text content. However these modules may have other behaviours, and the term `input handler` is now progressively substituted in the documentation. `filter` is still used in many places though.

Recoll input handlers 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 is plain text (in many cases the processing pipeline has an intermediary HTML step, which may be used for better previewing presentation). Most input handlers are executable programs or scripts. A few handlers are coded in C++ and live inside **recollindex**. This latter kind will not be described here.

There are two kinds of external executable input handlers:

- Simple `exec` handlers 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 plain text or HTML. HTML is usually preferred because it can store metadata fields and it allows preserving some of the formatting for the GUI preview. However, these handlers have limitations:
  - They can only process one document per file.
  - The output MIME type must be known and fixed.
  - The character encoding, if relevant, must be known and fixed (or possibly just depending on location).
- Multiple `execm` handlers can process multiple files (sparing the process startup time which can be very significant), or multiple documents per file (e.g.: for archives or multi-chapter publications). They communicate with the indexer through a simple protocol, but are nevertheless a bit more complicated than the older kind. Most of the new handlers are written in Python (exception: **rclimg** which is written in Perl because `exiftool` has no real Python equivalent). The Python handlers use common modules to factor out the boilerplate, which can make them very simple in favorable cases. The subdocuments output by these handlers can be directly indexable (text or HTML), or they can be other simple or compound documents that will need to be processed by another handler.

In both cases, handlers 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.

A simple handler returning a document in `text/plain` format, can transfer no metadata to the indexer. Generic metadata, like document size or modification date, will be gathered and stored by the indexer.

Handlers 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 handlers 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` 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. These handlers can also return metadata either as HTML `meta` tags, or as named data through the communication protocol.

The following section describes the simple handlers, and the next one gives a few explanations about the `execm` ones. You could conceivably write a simple handler 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 input handlers

Recoll simple handlers 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 handler. In some cases the output of the translating program is completely appropriate, and no intermediate shell-script is needed.

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

When writing a handler, 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 handler if the operation is for indexing or previewing. Some handlers use this to output a slightly different format, for example stripping uninteresting repeated keywords (e.g.: `Subject:` for email) when indexing. This is not essential.

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

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

4.1.2 "Multiple" handlers

If you can program and want to write an `execm` handler, it should not be too difficult to make sense of one of the existing handlers.

The existing handlers differ in the amount of helper code which they are using:

- `rclimg` is written in Perl and handles the execm protocol all by itself (showing how trivial it is).
- All the Python handlers share at least the `rclexecm.py` module, which handles the communication. Have a look at, for example, `rclzip.py` for a handler which uses `rclexecm.py` directly.
- Most Python handlers which process single-document files by executing another command are further abstracted by using the `rclexec1.py` module. See for example `rclrtf.py` for a simple one, or `rcldoc.py` for a slightly more complicated one (possibly executing several commands).
- Handlers which extract text from an XML document by using an XSLT style sheet are now executed inside **recollindex**, with only the style sheet stored in the `filters/` directory. These can use a single style sheet (e.g. `abiword.xsl`), or two sheets for the data and metadata (e.g. `opendoc-body.xsl` and `opendoc-meta.xsl`). The `mimeconf` configuration file defines how the sheets are used, have a look. Before the C++ import, the xsl-based handlers used a common module `rclgenxslt.py`, it is still around but unused at the moment. The handler for OpenXML presentations is still the Python version because the format did not fit with what the C++ code does. It would be a good base for another similar issue.

There is a sample trivial handler based on `rclexecm.py`, with many comments, not actually used by Recoll. It would index a text file as one document per line. Look for `rcltxtlines.py` in the `src/filters` directory in the online Recoll Git repository (the sample not in the distributed release at the moment).

You can also have a look at the slightly more complex **rclzip.py** which uses Zip file paths as identifiers (`ipath`).

`execm` handlers 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 handler will be escaped, but would be a bad choice as a handler-specific separator (mostly, again, for debugging issues).

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

`execm` handlers 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 handler

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

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, recent Recoll will use libmagic. Older versions or specially built ones may try to execute a system command (typically **file -i** or **xdg-mime**).

The second element is the association of MIME types to handlers in the mimeconf file. A sample will probably be better 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.py
```

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 persistent handler. This is determined by the `execm` keyword.

4.1.4 Input handler output

Both the simple and persistent input handlers can return any MIME type to Recoll, which will further process the data according to the MIME configuration.

Most input filters filters produce either `text/plain` or `text/html` data. There are exceptions, for example, filters which process archive file (`zip`, `tar`, etc.) will usually return the documents as they are found, without processing them further.

There is nothing to say about `text/plain` output, except that its character encoding should be consistent with what is specified in the `mimeconf` file.

For filters producing HTML, the output 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 external helper 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">
```

Input handlers 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.

Persistent filters can use another, probably simpler, method to produce metadata, by calling the `setfield()` helper method. This avoids the necessity to produce HTML, and any issue with HTML quoting. See, for example, `rclaudio.py` in Recoll 1.23 and later for an example of handler which outputs `text/plain` and uses `setfield()` to produce metadata.

4.1.5 Page numbers

The indexer will interpret ^L characters in the handler 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 handlers 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 input handlers 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 handlers, 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.

Some fields may also designated as supporting range queries, meaning that the results may be selected for an interval of its values. See the configuration section for more details.

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 independent 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 in the FAQs area, detailing how one could add a *page count* field to pdf documents for displaying inside result lists.

4.3 Python API

4.3.1 Introduction

The Recoll Python programming interface can be used both for searching and for creating/updating an index with a program run by the Python3 interpreter. It is available on all platforms (Unix-like systems, MS Windows, MacOS).

The search interface is used in a number of active projects: the Recoll Gnome Shell Search Provider, the Recoll Web UI, and the upmpdcli UPnP Media Server, in addition to many small scripts.

The index updating part of the API can be used to create and update Recoll indexes. Up to Recoll 1.37 these needed to use separate configurations (but could be queried in conjunction with the regular index). As of Recoll 1.37, an external indexer based on the Python extension can update the main index. For example the Recoll indexer for the Joplin notes application is using this method.

The search API is modeled along the Python database API version 2.0 specification (early versions used the version 1.0 spec).

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 at query time to access document data. This can be used, for example, for extracting embedded documents into standalone files.

There is a good chance that your system repository has packages for the Recoll Python API, sometimes in a package separate from the main one (maybe named something like python3-recoll). Else refer to the Building from source chapter.

As an introduction sample, the following small program will run a query and list the title and url for each of the results. The `python/samples` source directory contains several examples of Python programming with Recoll, exercising the extension more completely, and especially its data extraction features.

```
#!/usr/bin/python3

from recoll import recoll

db = recoll.connect()
query = db.query()
nres = query.execute("some query")
results = query.fetchmany(20)
for doc in results:
    print("%s %s" % (doc.url, doc.title))
```

You can also take a look at the source for (in order of complexity) the Recoll Gnome Shell Search Provider or WebUI, and the upmpdcli local media server.

4.3.2 Interface elements

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

ipath An `ipath` identifies an embedded document inside a standalone one (designated by an URL). The value, if needed, is stored along with the URL, but not indexed. It is accessible or set as a field in the Doc object.

`ipaths` are opaque values for the lower index layers (Doc objects producers or consumers), and their use is up to the specific indexer. For example, the Recoll file system indexer uses the `ipath` to store the part of the document access path internal to (possibly imbricated) container documents. `ipath` in this case is a vector of access elements (e.g, the first part could be a path inside a zip file to an archive member which happens to be an mbox file, the second element would be the message sequential number inside the mbox etc.). The index itself has no knowledge of this hierarchical structure.

At the moment, only the filesystem indexer uses hierarchical `ipaths` (neither the Web nor the Joplin one do), and there are some assumptions in the upper software layers about their structure. For example, the Recoll GUI knows about using an FS indexer `ipath` for such functions as opening the immediate parent of a given document.

`url` and `ipath` are returned in every search result and define the access to the original document. `ipath` is empty for top-level document/files (e.g. a PDF document which is a filesystem file).

udi An `udi` (unique document identifier) identifies a document. Because of limitations inside the index engine, it is restricted in length (to 200 bytes). 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 a maximum length, the suppressed part being replaced by a hash value to retain practical unicity.

To rephrase, and hopefully clarify: the filesystem indexer can't use the URL+ipath as a unique document-identifying term because this may be too big: it derives a shorter `udi` from URL+ipath. Another indexer could use a completely different method. For example, the Joplin indexer uses the note ID.

parent_udi If this attribute is set on a document when entering it in the index, it designates its physical container document. In a multilevel hierarchy, this may not be the immediate parent. If the indexer uses the `purge()` method, then the use of `parent_udi` is mandatory for subdocuments. Else it is optional, but its use by an indexer may simplify index maintenance, as Recoll will automatically delete all children defined by `parent_udi == udi` when the document designated by `udi` is destroyed. e.g. if a `Zip` archive contains entries which are themselves containers, like `mbox` files, all the subdocuments inside the `Zip` file (mbox, messages, message attachments, etc.) would have the same `parent_udi`, matching the `udi` for the `Zip` file, and all would be destroyed when the `Zip` file (identified by its `udi`) is removed from the index.

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. Apart from a few standard/internal fields, only the `stored` fields are retrievable through the Python search interface.

4.3.3 Log messages for Python scripts

Two specific configuration variables: `pyloglevel` and `pylogfilename` allow overriding the generic values for Python programs. Set `pyloglevel` to 2 to suppress default startup messages (printed at level 3).

4.3.4 Python search interface

4.3.4.1 The recoll module

4.3.4.1.1 connect(confdir=None, extra_dbs=None, writable = False)

The `connect()` function connects to one or several Recoll index(es) and returns a `Db` object.

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

- *`confdir`* designates the main index configuration directory. The usual system-dependant defaults apply if the value is empty.
- *`extra_dbs`* is a list of additional external indexes (Xapian directories). These will be queried, but supply no configuration values.
- *`writable`* decides if we can index new data through this connection.

Example:

```
from recoll import recoll

# Opening the default db
db = recoll.connect()

# Opening the default db and a pair of additional indexes
db = recoll.connect(extra_dbs=["/home/me/.someconfdir/xapiandb", "/data/otherconf/xapiandb ↩
    "])
```

4.3.4.1.2 The Db class

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

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

Db.getdoc(udi, idxidx=0) Retrieve a document given its unique document identifier, and its index if external indexes are in use. The main index is always index 0. The *udi* value could have been obtained from an earlier query as doc.rcludi, or would be known because the application is the indexer and generates the values.

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 one of `wildcard`, `regexp` or `stem`. *field*, if set, restricts the matches to the contents of the specified metadata field. Returns a list of terms expanded from the input expression.

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.close() Closes the connection. You can't do anything with the `Db` object after this. If the index was opened as writable, this commits any pending change.

Db.setSynonymsFile(path) Set the synonyms file used when querying.

4.3.4.1.3 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.

Query.sortby(fieldname, ascending=True) Set the sorting order for future searches to using *fieldname*, in ascending or descending order. Must be called before executing the search.

Query.execute(query_string, stemming=1, stemlang="english", fetchtext=False, collapseduplicates=False) Start a search for *query_string*, a Recoll search language string. If the index stores the documents texts and *fetchtext* is True, the `Doc` objects in the query result will store the document extracted text in doc.text. Else, the doc.text fields will be empty. If *collapseduplicates* is true, only one of multiple identical documents (defined by having the same MD5 hash) will appear in the result list.

Query.executesd(SearchData, fetchtext=False, collapseduplicates=False) Starts a search for the query defined by the `SearchData` object. See above for a description of the other parameters.

Query.fetchmany(size=query.arraysize) Fetch the next `Doc` objects from the current search result list, and return them as an array of the required size, which is by default the value of the arraysize data member.

Query.fetchone() Fetch the next `Doc` object from the current search result list. Generates a `StopIteration` exception if there are no results left.

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

```
from recoll import recoll

db = recoll.connect()
q = db.query()
nres = q.execute("some query")
for doc in q:
    print("%s" % doc.title)
```

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

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

Query.getgroups() Retrieve 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 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">`, and `</span>` 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 having methods `startMatch(i)` and `endMatch()` which will be called for each match and should return a begin and end tag. Example:

```
class MyHighlighter:
    def startMatch(self, idx):
        return "<span style='color:red;background:yellow;'>"
    def endMatch(self):
        return "</span>"
```

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.getsnippets(doc, maxoccs = -1, ctxwords = -1, sortbypage=False, methods=object) Return a list of extracts from the result document by selecting text around the match terms. Each entry in the result list is a triple: page number, term, text. By default, the most relevants snippets appear first in the list. Set *sortbypage* to sort by page number instead. If *methods* is set, the fragments will be highlighted (see the `highlight()` method). If *maxoccs* is set, it defines the maximum result list length. *ctxwords* allows adjusting the individual snippet context size.

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

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.

4.3.4.1.4 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.

Please note that a `Doc` should never be instanciated by its constructor but instead by calling `db.doc()` or some other API method returning a doc object. Otherwise, the object will lack some necessary references.

The Doc object has many attributes to be read or set by its user. It mostly matches 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.cpp` C++ file for a full description of the predefined attributes. Here follows a short list.

- `url` the document URL but see also `getbinurl()`
- `ipath` the document `ipath` for embedded documents.
- `fbytes, dbytes` the document file and text sizes.
- `fmtime, dmtime` the document file and document times.
- `xdocid` the document Xapian document ID. This is useful if you want to access the document through a direct Xapian operation.
- `mtype` the document MIME type.

- `text` holds the document processed text, if the index itself is configured to store it (true by default) and if the `fetchtext` query `execute()` option was true. See also the `rclextract` module for accessing document contents.
- Other fields stored by default: `author`, `filename`, `keywords`, `recipient`

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.

get(key), [] operator Retrieve the named document attribute. You can also use `getattr(doc, key)` or `doc.key`.

doc.key = value Set the the named document attribute. You can also use `setattr(doc, key, value)`.

getbinurl() Retrieve the URL in byte array format (no transcoding), for use as parameter to a system call. This is useful for the filesystem indexer `file://` URLs which are stored unencoded, as binary data.

setbinurl(url) Set the URL in byte array format (no transcoding).

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

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

4.3.4.1.5 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...

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

4.3.4.2 The rclextract module

Prior to Recoll 1.25, index queries could not provide document content because it was never stored. Recoll 1.25 and later usually store the document text, which can be optionally retrieved when running a query (see `query.execute()` above - the result is always plain text).

Independantly, the `rclextract` module can give access to the original document and to the document text content, possibly as an HTML version. Accessing the original document is particularly useful if it is embedded (e.g. an email attachment).

You need to import the `recoll` module before the `rclextract` module.

4.3.4.2.1 The Extractor class

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 converted to either text/plain or text/html according to `doc.mimetype`. The typical use would be as follows:

```
from recoll import recoll, rclextract

qdoc = query.fetchone()
extractor = rclextract.Extractor(qdoc)
doc = extractor.textextract(qdoc.ipath)
# use doc.text, e.g. for previewing
```

Passing `qdoc.ipath` to `textextract()` is redundant, but reflects the fact that the `Extractor` object actually has the capability to access the other entries in a compound document.

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:

```
from recoll import recoll, rclextract

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

In all cases the output is a copy, even if the requested document is a regular system file, which may be wasteful in some cases. If you want to avoid this, you can test for a simple file document as follows:

```
not doc.ipath and (not "rclbes" in doc.keys() or doc["rclbes"] == "FS")
```

4.3.4.3 Search API usage example

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/python3

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: %d" % 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("%s : %s" % (k, getattr(doc, k)))
    print("%s\n" % db.makeDocAbstract(doc, query))
```

4.3.4.4 The fsudi module

The `fsudi` module contains a single method, which duplicates the code used by the main filesystem indexer to derive an UDI from a filesystem path. In turn, this allows external code to call the `getDoc()` method to retrieve the `Doc` object. This can be useful, for example, for updating the metadata without fully reindexing the document.

fsudi.fs_udi(path, ipath='') Obtain the UDI value for the given *`path`* and *`ipath`*. The returned value can be used with the `db.getDoc()` method.

4.3.5 Python indexing interface

4.3.5.1 Recoll external indexers

The Recoll indexer is capable of processing many different document formats. However, some forms of data storage do not lend themselves easily to standard processing because of their great variability. A canonical example would be data in an SQL database. While it might be possible to create a configurable handler to process data from a database, the many variations in storage organisation and SQL dialects make this difficult.

Recoll can instead support external indexers where all the responsibility to handle the data format is delegated to an external script. The script language has to be Python 3 at the moment, because this is the only language for which an API binding exists.

Up to Recoll 1.35, such an indexer had to work on a separate Recoll index, which would be added as an external index for querying from the main one, and for which a separate indexing schedule had to be managed. The reason was that the main document indexer purge pass (removal of deleted documents) would also remove all the documents belonging to the external indexer, as they were not seen during the filesystem walk (and conversely, the external indexer purge pass would delete all the regular document entries).

As of Recoll 1.36, an improvement and new API call allows external indexers to be fully integrated, and work on the main index, with updates triggered from the normal **recollindex** program.

An external indexer has to do the same work as the Recoll file system indexer: look for modified documents, extract their text, call the API for indexing them, and the one for purging the data for deleted documents.

A description of the API method follows, but you can also jump ahead for a look at some sample pseudo-code and a pair of actual implementations, one of which does something useful.

4.3.5.2 The Python indexing API

There are two parts in the indexing interface:

- Methods inside the `recoll` module allow the foreign indexer to update the index.
- An interface based on scripts execution is defined for executing the indexer (from **recollindex**) and to allow either the GUI or the `rclextract` module to access original document data for previewing or editing.

Two sample scripts are included with the Recoll source and described in more detail a bit further.

4.3.5.2.1 Python indexing interface methods

The update methods are part of the `recoll` module. The connect() method is used with a `writable=true` parameter to obtain a writable `Db` object. The following `Db` object methods are then available.

Note that the changes are only guaranteed to be flushed to the index when `db.close()` is called. This normally occurs when the program exits, but it is much safer to use an explicit call after making the changes.

addOrUpdate(udi, doc, parent_udi=None, metaonly=False) Add or update index data for a given document.

The `udi` string must define a unique id for the document. It is an opaque interface element and not interpreted inside the lower level Recoll code.

doc is a `Doc` object, containing the data to be indexed.

If *parent_udi* is set, this is a unique identifier for the top-level container, the document for which `needUpdate()` would be called (e.g. for the filesystem indexer, this would be the one which is an actual file).

If *metaonly* is set, the main document text (in the `doc.text` attribute) will be ignored and only the other metadata fields present in the *doc* object will be processed. This can be useful for updating the metadata apart from the main text.

Document attributes: `doc.text` should have the main text. It is ignored if *metaonly* is set. For actual indexing (*metaonly* not set), the `url` and possibly `ipath` fields should also be set to allow access to the actual document after a query. Other fields may also need to be set by an external indexer see the description further down: `rclbes`, `sig`, `mimetype`. Of course, any standard or custom Recoll field can also be added.

delete(udi) Purge the index from all data for *udi*, and all documents (if any) which have *udi* as `parent_udi`.

needUpdate(udi, sig) Test if the index needs to be updated for the document identified by *udi*. If this call is to be used, the `doc.sig` field should contain a signature value when calling `addOrUpdate()`. The `needUpdate()` call then compares its parameter value with the stored *sig* for *udi*. *sig* is an opaque value, compared as a string.

The filesystem indexer uses a concatenation of the decimal string values for file size and update time, but a hash of the contents could also be used.

As a side effect, if the return value is false (the index is up to date), the call will set the existence flag for the document (and any subdocument defined by its `parent_udi`), so that a later `purge()` call will preserve them.

The use of `needUpdate()` and `purge()` is optional, and the indexer may use another method for checking the need to reindex or to delete stale entries.

preparePurge(backend_name) Mark all documents which do *not* belong to *`backend_name`* as existing. *`backend_name`* is the value chosen for the `rclbes` field for the indexer documents (e.g. "MBOX", "JOPLIN"... for the samples). This is a mandatory call before starting an update if the index is shared with other backends and you are going to call purge() after the update, else all documents for other backends will be deleted from the index by the purge.

purge() Delete all documents that were not touched during the just finished indexing pass (since `preparePurge()`). These are the documents for which the `needUpdate()` call was not performed, indicating that they no longer exist in the storage system.

createStemDbs(lang|sequence of langs) Create stemming dictionaries for query stemming expansion. Note that this is not needed at all if the indexing is done from the **recollindex** program, as it will perform this action after calling all the external indexers. Should be called when done updating the index. Available only after Recoll 1.34.3. As an alternative, you can close the index and execute:

```
recollindex -c <confdir> -s <lang(s)>
```

The Python module currently has no interface to the Aspell speller functions, so the same approach can be used for creating the spelling dictionary (with option `-S`) (again, not needed if **recollindex** is driving the indexing).

4.3.5.2.2 Query data access for external indexers

Recoll has internal methods to access document data for its internal (filesystem) indexer. An external indexer needs to provide data access methods if it needs integration with the GUI (e.g. preview function), or support for the `rclextract` module.

An external indexer needs to provide two commands, for fetching data (typically for previewing) and for computing the document signature (for up-to-date checks when opening or previewing). The sample MBOX and JOPLIN implementations use the same script with different parameters to perform both operations, but this is just a choice. A third command must be provided for performing the indexing proper.

The "fetch" and "makesig" scripts are called with three additional arguments: `udi`, `url`, `ipath`. These were set by the indexer and stored with the document by the `addOrUpdate()` call described above. Not all arguments are needed in all cases, the script will use what it needs to perform the requested operation. The caller expects the result data on `stdout`.

recollindex will set the `RECOLL_CONFDIR` environment variable when executing the scripts, so that the configuration can be created as

```
rclconf = rclconfig.RclConfig()
```

if needed, and the configuration directory obtained as

```
confdir = rclconf.getConfDir()
```

4.3.5.3 External indexers configuration

The index data and the access method are linked by the `rclbes` (recoll backend storage) `Doc` field. You should set this to a short string value identifying your indexer (e.g. the filesystem indexer uses either `FS` or an empty value, the Web history indexer uses `BGL`, the Joplin notes indexer uses `JOPLIN`).

The link is actually performed inside a `backends` configuration file (stored in the configuration directory). This defines commands to execute to access data from the specified indexer. Example, for the mbox indexing sample found in the Recoll source (which sets `rclbes="MBOX"`):

```
[MBOX]
fetch = /path/to/recoll/src/python/samples/rclmbox.py fetch
makesig = /path/to/recoll/src/python/samples/rclmbox.py makesig
index = /path/to/recoll/src/python/samples/rclmbox.py index
```

When updating the index, the **recollindex** will execute the value of the the `index` parameter, if present (it may not be present if this concerns an external index).

If an external indexer needs to store additional configuration parameters, e.g. path to a specific instance of the indexed application, etc., I suggest storing them inside `recoll.conf`, with a backend-specific prefix (e.g. *`joplin_db`*, *`mbox_directory`*) and using methods from the `rclconfig` module to access them.

4.3.5.4 External indexer samples

First a quick look at an indexer main part, using pseudo-Python3 code:

```
# Connect to the recoll index. This will use the RECOLL_CONFDIR variable, set
# by the parent recollindex process, to use the right index.
rcldb = recoll.connect(writable=1)

# Important: tell the Recoll db that we are going to update documents for the
# MYBACK backend. All other documents will be marked as present so as
# not to be affect by the subsequent purge.
rcldb.preparePurge("MYBACK")

# Walk your dataset (of course your code will not look like this)
for mydoc in mydoclist:
    # Compute the doc unique identifier and the signature corresponding to its update state
    # (e.g. mtime and size for a file).
    udi = mydoc.udi()
    sig = mydoc.sig()
    # Check with recoll if the document needs updating. This has the side-effect or marking
    # it present.
    if not rcldb.needUpdate(udi, sig):
        continue
    # The document data does not exist in the index or needs updating. Create and add a  ↩
       Recoll
    # Doc object
    doc = recoll.Doc()
    doc.mimetype = "some/type"
    # Say that the document belongs to this indexer
    doc.rclbes = "MYBACK"
    # The url will be passed back to you along with the udi if the fetch
    # method is called later (for previewing), or may be used for opening the document with
    # its native app from Recoll. The udi has a maximum size because it is used as a Xapian
    # term. The url has no such limitation.
    doc.url = "someurl"
    doc.sig = sig
    # Of course add other fields like "text" (duh), "author" etc. See the samples.
    doc.text = mydoc.text()
    # [...]
    # Then add or update the data in the index.
    self.db.addOrUpdate(udi, doc)

# Finally call purge to delete the data for documents which were not seen at all.
db.purge()
```

The Recoll source tree has two samples of external indexers.

- rclmbox.py indexes a directory containing `mbox` folder files. Of course it is not really useful because Recoll can do this by itself, but it exercises most features in the update interface, and it has both top-level and embedded documents so it demonstrates the uses of the `ipath` values.
- rcljoplin.py indexes a Joplin application main `notes` SQL table. Joplin sets an an update date attribute for each record in the table, so each note record can be processed as a standalone document (no `ipath` necessary). The sample has full preview and

open support (the latter using a Joplin callback URL which allows displaying the result note inside the native app), so it could actually be useful to perform a unified search of the Joplin data and the regular Recoll data. As of Recoll 1.37.0, the Joplin indexer is part of the default installation (see the features section of the Web site for more information).

See the comments inside the scripts for more information.

4.3.5.5 Using an external indexer index in conjunction with a regular one

When adding an external indexer to a regular one for unified querying, some elements of the foreign index configuration should be copied or merged into the main index configuration. At the very least, the `backends` file needs to be copied or merged, and also possibly data from the `mimeconf` and `mimeview` files. See the `rcljoplin.py` sample for an example.

Chapter 5

Installation and configuration

5.1 Installing a binary copy

Recoll binary copies are always distributed as regular packages for your system. They can be obtained either through the system's normal software distribution framework (e.g. Debian/Ubuntu apt, FreeBSD ports, etc.), or from some type of "backports" repository providing versions newer than the standard ones, or found on the Recoll Web site in some cases. The most up-to-date information about Recoll packages can usually be found on the Recoll Web site downloads page

The Windows version of Recoll comes in a self-contained setup file, there is nothing else to install.

On Unix-like systems, the package management tools will automatically install hard dependencies for packages obtained from a proper package repository. You will have to deal with them by hand for downloaded packages (for example, when **dpkg** complains about missing dependencies).

In all cases, you will 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.

5.2 Supporting packages

Note
The Windows installation of Recoll is self-contained. Windows users can skip this section.

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.

The past has proven that I was unable to maintain an up to date application list in this manual. Please check https://www.recoll.org/pages/features.html for a complete list along with links to the home pages or best source/patches pages, and misc tips. What follows is only a very short extract of the stable essentials.

- PDF files need **pdftotext** which is part of Poppler (usually comes with the `poppler-utils` package). Avoid the original one from Xpdf.
- MS Word documents need **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.

- RTF files need **unrtf**, which, in its older versions, has much trouble with non-western character sets. Many Linux distributions carry outdated **unrtf** versions. Check https://www.recoll.org/pages/features.html for details.
- Pictures: Recoll uses the Exiftool Perl package to extract tag information. Most image file formats are supported.
- Up to Recoll 1.24, many XML-based formats need the **xsltproc** command, which usually comes with libxslt. These are: abiword, fb2 ebooks, kword, openoffice, opendocument svg. Recoll 1.25 and later process them internally (using libxslt).

5.3 Building from source

5.3.1 Prerequisites

The following prerequisites are described in broad terms and Debian package names. The dependencies should be available as packages on most common Unix-like systems, and it should be quite uncommon that you would have to build one of them. Finding the right package name for non-Debian systems is left to the sagacity of the reader.

Up to version 1.37, the Recoll build process used the GNU autotools. Versions 1.38 and later use meson/ninja instead.

If you do not need the GUI, you can avoid all GUI dependencies by disabling its build: see the configure section further down: `-Dqtgui=false`.

Check the Recoll download page for up to date Recoll version information and links to source release files in **tar** format.

The shopping list follows:

- If you start from git repository source code, you will need the **git**, obviously (package: `git`).
- On Unix-like systems systems, you will need the **meson** and **ninja** commands. (package: `meson`, this will bring `ninja` as a dependency). Not needed on MacOS systems at the moment.
- The pkg-config command is needed for configuring the build (package: `pkg-config`).
- The **make** command is needed for building the GUI, unneeded if you disable this. (package: `make`).
- A C++ compiler with at least C++17 compatibility (`g++` or `clang`). Recoll Versions 1.33.4 and older only required c++11.
- The **bison** command is not generally needed, but might be if you modify the query language yacc source or if some file modification times are not right (package: `bison`).
- For building the documentation: the **xsltproc** command, and the Docbook XML and style sheet files. You can avoid this dependency by disabling documentation building with the `-Duserdoc=false` setup option.
- Development files for Xapian core (`libxapian-dev`).
- Development files for libxml2 and libxslt (packages: `libxslt1-dev`, which will pull `libxml2-dev`).
- Development files for zlib (`zlib1g-dev`).
- Development files for libmagic (`libmagic-dev`).
- Development files for libaspell (package: `libaspell-dev`). Can be avoided with the `-Daspell=false` setup option.
- If you want to process CHM files, you will need libchm (`libchm-dev`), else you can set the `-Dpython-chm=false` option to the setup command.
- If you want the daemon indexer process to monitor the session for quitting, you need the X11 development library (package: `libx11-dev`). Else use the -Dx11mon=false setup option.
- If you want to build the GUI: **qmake** and development files for Qt 5. Else give the `-Dqtgui=false` setup option. Packages: `qtbase5-dev`, `qttools5-dev-tools`, `libqt5webkit5-dev`. Replace `libqt5webkit5-dev` with `libqt5webengin` if you use -Dwebengine=true.
- Development files for Python3 (packages: `python3-all-dev`, `python3-setuptools`). You can use the `-Dpython-module` setup option for disabling the build of the Python extension.
- You may also need libiconv. On Linux systems, the iconv interface is part of libc and you should not need to do anything special.

5.3.2 Building

Recoll has been built on Linux, FreeBSD, MacOS, and Solaris, most versions after 2005 should be ok, maybe some older ones too (Solaris 8 used to be ok). Current Recoll versions (1.34 and later) need a c++17 compiler and Qt5, so they will not build on old systems, but if really needed, you can probably find an older version which will work for you. If you build on another system, and need to modify things, I would very much welcome patches.

5.3.2.1 meson setup options

Of course the usual **meson** `setup` options, like `-Dprefix=/usr` apply.

`-Daspell=false` will disable the code for phonetic matching of search terms.

`-Dfam=true` or `-Dinotify=true` will enable the code for real time indexing. Inotify support is enabled by default on Linux systems. MacOS systems and Windows platforms now have real time indexing enabled by default and need no setup options.

`-Dqzeitgeist=true` will enable sending Zeitgeist events about the visited search results, and needs the qzeitgeist package.

`-Dqtgui=false` will disable the Qt graphical interface, which allows building the indexer and the command line search program in absence of a Qt environment.

`-Dwebkit=false` will implement the result list with a Qt QTextBrowser instead of a WebKit widget if you do not or can't depend on the latter.

`-Dwebengine=true` will enable the use of Qt Webengine (only meaningful if the Qt GUI is enabled), in place or Qt Webkit.

`-Dwebpreview=false`: do not implement the GUI preview windows with webkit or webengine instead of qtextbrowser. Using webxx will usually produce a better display, but will sometimes fail to display anything because of javascript issues.

`-Dguidebug=true` will build the recoll GUI program with debug symbols. This makes it very big (~50MB), which is why it is stripped by default.

`-Didxthreads=false` will 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.

`-Dpython-module=false` will avoid building the Python extension.

`-Dpython-chm=false` will avoid building the Python libchm interface used to index CHM files.

`-Dpython-aspell=false` will avoid building the Python libaspell interface. This is used to supplement queries with spelling guesses.

`-Dindexer=false` will prevent building the indexer. Possibly useful if you just need the lib (e.g. for the Python extension).

`-Dsimdutf=false` will prevent the use of the simdutf code normally used to speed up character code conversions.

`-Dcamelcase=true` 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).

`-Dlibmagic=false`: disable the use of libmagic (use a `file`-like command instead).

`-Dfile-command=somecommand` Specify the version of the 'file' command to use (e.g.: -Dfile-command=/usr/local/bin/file). Can be useful to enable the gnu version on systems where the native one is bad.

`-Dx11mon=false` Disable X11 connection monitoring inside **recollindex**. Together with -Dqtgui=false, this allows building recoll without Qt and X11.

`-Duserdoc=false` will avoid building the user manual. This avoids having to install the Docbook XML/XSL files and the TeX toolchain used for translating the manual to PDF.

`-Drecollq=true` Enable building the **recollq** command line query tool (recoll -t without need for Qt). This is done by default if -Dqtgui=false is used but this option enables forcing it.

`-Dsystemd=false` Disable the automatic installation of systemd unit files. Normally unit files are installed if the install path can be detected.

`-Dsystem-unit-dir=DIR` Provide an install path for the systemd system unit template file.

`-Duser-unit-dir=DIR` Provide an install path for the systemd user unit file.

5.3.2.2 Normal procedure, for source extracted from a tar distribution)

For versions 1.38 and later (else check the manual inside the older source):

```
cd recoll-xxx
meson setup [options] build
ninja -C build
```

5.3.3 Installing

Use **sudo ninja install** in your build tree. This will copy the commands to *prefix*/bin and the sample configuration files, scripts and other shared data to *prefix*/share/recoll.

5.3.4 Python API package

The Python interface can be found in the source tree, under the python/recoll directory.

The normal Recoll build procedure (see above) installs the API package for Python3.

For meson-based versions: the python/recoll/ directory still contains a setup.py. This is obsoleted by meson.build but might be useful in some cases.

5.4 Settings, 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).

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 with a plain text editor 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 approaches on the same configuration.

For each index, there are at least two sets of configuration files. System-wide configuration files are kept in a directory named like /usr/share/recoll/examples, and define default values, shared by all indexes (the values in these files are often commented out, and just present to indicate the default coded in the program). For each index, a parallel set of files defines the customized parameters.

On Unix-like systems, the default location of the customized configuration is the .recoll directory in your home. On Windows it is C:/Users/[you]/AppData/Local/Recoll. Most people will only use this directory.

The default location for the configuration directory can be changed, or others can be added for separate indexes with the RECOLL_CONFDIR environment variable or the -c option parameter to **recoll** and **recollindex**.

In addition, 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 overridden 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.

If the default configuration directory does not exist when either **recoll** or **recollindex** is 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 which case, 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 files and directories to index.
topdirs =  ~/docs /usr/share/doc

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

There are three kinds of lines:

- Comment lines start with a hash mark #.
- Parameter assignment: *name* = *value*.
- Section definitions: [*somedirname*].

Lines which are empty or only containing white space are ignored.

Long lines can be broken by ending each incomplete part with a backslash (\).

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.

Important
Global parameters *must not* be defined in a directory subsection, else they will not be found at all by the Recoll code, which looks for them at the top level (e.g. `skippedPaths`).

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. The same convention is used on Windows.

Some parameters are lists of strings. White space is used for separation. List elements with embedded spaces can be quoted using double-quotes. Double quotes inside these elements can be escaped with a backslash.

No value inside a configuration file can contain a newline character. Long lines can be continued by escaping the physical newline with backslash, even inside quoted strings.

```
astringlist =  "some string \
with spaces"
thesame = "some string with spaces"
```

Parameters which are not part of string lists can't be quoted, and leading and trailing space characters are stripped before the value is used.

Important
Quotes processing is ONLY applied to parameter values which are lists. Double quoting a single value like, e.g. `dbdir` will result in an incorrect value, with quotes included. This is quite confusing, and may have been a design mistake but it is much too late to fix.

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 (meaning unaccenting exception translations) should be encoded in UTF-8. If your system locale is not UTF-8 (which is now very rare), 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 Environment variables

RECOLL_CONFDIR Defines the main configuration directory.

RECOLL_TMPDIR, TMPDIR Locations for temporary files, in this order of priority. The default if none of these is set is to use `/tmp`. Big temporary files may be created during indexing, mostly for decompressing, and also for processing, e.g. email attachments.

RECOLL_CONFTOP, RECOLL_CONFMID Allow adding configuration directories with priorities below and above the user directory (see above the Configuration overview section for details).

RECOLL_EXTRA_DBS, RECOLL_ACTIVE_EXTRA_DBS Help for setting up external indexes. See this paragraph for explanations.

RECOLL_DATADIR Defines replacement for the default location of Recoll data files, normally found in, e.g., `/usr/share/recoll`).

RECOLL_FILTERSDIR Defines replacement for the default location of Recoll filters, normally found in, e.g., `/usr/share/recoll/filters`).

ASPELL_PROG aspell program to use for creating the spelling dictionary. The result has to be compatible with the `libaspell` which Recoll is using.

5.4.2 Recoll main configuration file, recoll.conf

5.4.2.1 Parameters affecting what documents we index

topdirs Space-separated list of files or directories to recursively index. You can use symbolic links in the list, they will be followed, independently of the value of the followLinks variable. The default value is ~ : recursively index $HOME.

monitordirs Space-separated list of files or directories to monitor for updates. When running the real-time indexer, this allows monitoring only a subset of the whole indexed area. The elements must be included in the tree defined by the 'topdirs' members.

skippedNames File and directory names which should be ignored. White space separated list of wildcard patterns (simple ones, not paths, must contain no '/' characters), which will be tested against file and directory names.

Have a look at the default configuration for the initial value, some entries may not suit your situation. The easiest way to see it is through the GUI Index configuration "local parameters" panel.

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" "~/.evolution" to "topdirs".

Not even the file names are indexed for patterns in this list, see the "noContentSuffixes" variable for an alternative approach which indexes the file names. Can be redefined for any subtree.

skippedNames- List of name patterns to remove from the default skippedNames list. Allows modifying the list in the local configuration without copying it.

skippedNames+ List of name patterns to add to the default skippedNames list. Allows modifying the list in the local configuration without copying it.

onlyNames Regular file name filter patterns. This is normally empty. If set, only the file names not in skippedNames and matching one of the patterns will be considered for indexing. Can be redefined per subtree. Does not apply to directories.

noContentSuffixes List of name endings (not necessarily dot-separated suffixes) for which we don't try MIME type identification, and don't uncompress or index content. Only the names will be indexed. This complements the now obsoleted recoll_noindex list from the mimemap file, which will go away in a future release (the move from mimemap to recoll.conf allows editing the list through the GUI). This is different from skippedNames because these are name ending matches only (not wildcard patterns), and the file name itself gets indexed normally. This can be redefined for subdirectories.

noContentSuffixes- List of name endings to remove from the default noContentSuffixes list.

noContentSuffixes+ List of name endings to add to the default noContentSuffixes list.

skippedPaths Absolute paths we should not go into. Space-separated list of wildcard expressions for absolute filesystem paths (for files or directories). The variable must be defined at the top level of the configuration file, not in a subsection.

Any value in the list must be textually consistent with the values in topdirs, no attempts are made to resolve symbolic links. In practise, if, as is frequently the case, /home is a link to /usr/home, your default topdirs will have a single entry '~' which will be translated to '/home/yourlogin'. In this case, any skippedPaths entry should start with '/home/yourlogin' *not* with '/usr/home/yourlogin'.

The index and configuration directories will automatically be added to the list.

The expressions are matched using 'fnmatch(3)' with the FNM_PATHNAME flag set by default. This means that '/' characters must be matched explicitly. You can set 'skippedPathsFnmPathname' to 0 to disable the use of FNM_PATHNAME (meaning that '/*/dir3' will match '/dir1/dir2/dir3').

The default value contains the usual mount point for removable media to remind you that it is in most cases a bad idea to have Recoll work on these. Explicitly adding '/media/xxx' to the 'topdirs' variable will override this.

skippedPathsFnmPathname Set to 0 to override use of FNM_PATHNAME for matching skipped paths.

nowalkfn File name which will cause its parent directory to be skipped. Any directory containing a file with this name will be skipped as if it was part of the skippedPaths list. Ex: .recoll-noindex

daemSkippedPaths skippedPaths equivalent specific to real time indexing. This enables having parts of the tree which are initially indexed but not monitored. If daemSkippedPaths is not set, the daemon uses skippedPaths.

zipUseSkippedNames Use skippedNames inside Zip archives. Fetched directly by the rclzip.py handler. Skip the patterns defined by skippedNames inside Zip archives. Can be redefined for subdirectories. See https://www.recoll.org/faqsandhowtos/Filte

zipSkippedNames Space-separated list of wildcard expressions for names that should be ignored inside zip archives. This is used directly by the zip handler. If zipUseSkippedNames is not set, zipSkippedNames defines the patterns to be skipped inside archives. If zipUseSkippedNames is set, the two lists are concatenated and used. Can be redefined for subdirectories. See https://www.recoll.org/faqsandhowtos/FilteringOutZipArchiveMembers.html

followLinks Follow symbolic links during indexing. 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. Links in the 'topdirs' list itself are always followed.

indexedmimetypes Restrictive list of indexed MIME types. Normally not set (in which case all supported types are indexed). If it is set, only the types from the list will have their contents indexed. The names will be indexed anyway if indexallfilenames is set (default). MIME type names should be taken from the mimemap file (the values may be different from xdg-mime or file -i output in some cases). Can be redefined for subtrees.

excludedmimetypes List of excluded MIME types. Lets you exclude some types from indexing. MIME type names should be taken from the mimemap file (the values may be different from xdg-mime or file -i output in some cases) Can be redefined for subtrees.

nomd5types MIME types for which we don't compute a md5 hash. md5 checksums are used only for deduplicating results, and can be very expensive to compute on multimedia or other big files. This list lets you turn off md5 computation for selected types. It is global (no redefinition for subtrees). At the moment, it only has an effect for external handlers (exec and execm). The file types can be specified by listing either MIME types (e.g. audio/mpeg) or handler names (e.g. rclaudio.py).

compressedfilemaxkbs Size limit for compressed files. We need to decompress these in a temporary directory for identification, which can be wasteful in some cases. Limit the waste. Negative means no limit. 0 results in no processing of any compressed file. Default 100 MB.

textfilemaxmbs Size limit for text files. Mostly for skipping monster logs. Default 20 MB. Use a value of -1 to disable.

textfilepagekbs Page size for text files. If this is set, text/plain files will be divided into documents of approximately this size. This will reduce memory usage at index time and help with loading data in the preview window at query time. Particularly useful with very big files, such as application or system logs. Also see textfilemaxmbs and compressedfilemaxkbs.

textunknownasplain Process unknown text/xxx files as text/plain Allows indexing misc. text files identified as text/whatever by 'file' or 'xdg-mime' without having to explicitely set config entries for them. This works fine for indexing (also will cause processing of a lot of useless files), but the documents indexed this way will be opened by the desktop viewer, even if text/plain has a specific editor.

indexallfilenames Index the file names of unprocessed files. Index the names of files the contents of which we don't index because of an excluded or unsupported MIME type.

usesystemfilecommand Use a system mechanism as last resort to guess a MIME type. Depending on platform and version, a compile-time configuration will decide if this actually executes a command or uses libmagic. This last-resort identification (if the suffix-based one failed) is generally useful, but will cause the indexing of many bogus extension-less 'text' files. Also see 'systemfilecommand'.

systemfilecommand Command to use for guessing the MIME type if the internal methods fail. This is ignored on Windows or with Recoll 1.38+ if compiled with libmagic enabled (the default). Otherwise, this should be a "file -i" workalike. The file path will be added as a last parameter to the command line. "xdg-mime" works better than the traditional "file" command, and is now the configured default (with a hard-coded fallback to "file")

processwebqueue Decide if we process the Web queue. The queue is a directory where the Recoll Web browser plugins create the copies of visited pages.

membermaxkbs Size limit for archive members. This is passed to the MIME handlers in the environment as RECOLL_FILTER_MAX

5.4.2.2 Parameters affecting how we generate terms and organize the index

indexStripChars Decide if we store character case and diacritics in the index. If we do, 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. When using multiple indexes for a search, this parameter must be defined identically for all. Changing the value implies an index reset.

indexStoreDocText Decide if we store the documents' text content in the index. Storing the text allows extracting snippets from it at query time, instead of building them from index position data.

Newer Xapian index formats have rendered our use of positions list unacceptably slow in some cases. The last Xapian index format with good performance for the old method is Chert, which is default for 1.2, still supported but not default in 1.4 and will be dropped in 1.6.

The stored document text is translated from its original format to UTF-8 plain text, but not stripped of upper-case, diacritics, or punctuation signs. Storing it increases the index size by 10-20% typically, but also allows for nicer snippets, so it may be worth enabling it even if not strictly needed for performance if you can afford the space.

The variable only has an effect when creating an index, meaning that the xapiandb directory must not exist yet. Its exact effect depends on the Xapian version.

For Xapian 1.4, if the variable is set to 0, we used to use the Chert format and not store the text. If the variable was 1, Glass was used, and the text stored. We don't do this any more: storing the text has proved to be the much better option, and dropping this possibility simplifies the code.

So now, the index format for a new index is always the default, but the variable still controls if the text is stored or not, and the abstract generation method. With Xapian 1.4 and later, and the variable set to 0, abstract generation may be very slow, but this setting may still be useful to save space if you do not use abstract generation at all, by using the appropriate setting in the GUI, and/or avoiding the Python API or recollq options which would trigger it.

nonumbers Decides if terms will be generated for numbers. For example "123", "1.5e6", 192.168.1.4, would not be indexed if nonumbers is set ("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, where setting nonumbers will reduce the index size. This can only be set for a whole index, not for a subtree.

notermpositions Do not store term positions. Term positions allow for phrase and proximity searches, but make the index much bigger. In some special circumstances, you may want to dispense with them.

dehyphenate Determines if we index 'coworker' also when the input is 'co-worker'. This is new in version 1.22, and on by default. Setting the variable to off allows restoring the previous behaviour.

indexedpunctuation String of UTF-8 punctuation characters to be indexed as words. The resulting terms will then be searchable and, for example, by setting the parameter to "%€" (without the double quotes), you would be able to search separately for "100%" or "100€" Note that "100%" or "100 %" would be indexed in the same way, the characters are their own word separators.

backslashasletter Process backslash as a normal letter. This may make sense for people wanting to index TeX commands as such but is not of much general use.

underscoreasletter Process underscore as normal letter. This makes sense in so many cases that one wonders if it should not be the default.

maxtermlength Maximum term length in Unicode characters. Words longer than this will be discarded. The default is 40 and used to be hard-coded, but it can now be adjusted. You may need an index reset if you change the value.

nocjk Decides if 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.

hangultagger External tokenizer for Korean Hangul. This allows using an language specific processor for extracting terms from Korean text, instead of the generic n-gram term generator. See https://www.recoll.org/pages/recoll-korean.html for instructions.

chinesetagger External tokenizer for Chinese. This allows using the language specific Jieba tokenizer for extracting meaningful terms from Chinese text, instead of the generic n-gram term generator. See https://www.recoll.org/pages/recoll-chinese.html for instructions.

indexstemminglanguages Languages for which to create stemming expansion data. Stemmer names can be found by executing 'recollindex -l', or this can also be set from a list in the GUI. The values are full language names, e.g. english, french...

defaultcharset Default character set. This is used for files which do not contain a character set definition (e.g.: text/plain). Values found inside files, e.g. a 'charset' tag in HTML documents, will override it. If this is not set, the default character set is the one defined by the NLS environment ($LC_ALL, $LC_CTYPE, $LANG), or ultimately iso-8859-1 (cp-1252 in fact). If for some reason you want a general default which does not match your LANG and is not 8859-1, use this variable. This can be redefined for any sub-directory.

unac_except_trans 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. The value is global and affects both indexing and querying. We also convert a few confusing Unicode characters (quotes, hyphen) to their ASCII equivalent to avoid "invisible" search failures.

Examples: Swedish: unac_except_trans = ää Ää öö Öö üü Üü ßss œoe Œoe æae Æae ffff fifi flfl åå Åå " ’ " – . German: unac_except_trans = ää Ää öö Öö üü Üü ßss œoe Œoe æae Æae ffff fifi flfl " ’ " – . French: you probably want to decompose oe and ae and nobody would type a German ß unac_except_trans = ßss œoe Œoe æae Æae ffff fifi flfl " ’ " – . The default for all until someone protests follows. These decompositions are not performed by unac, but it is unlikely that someone would type the composed forms in a search. unac_except_trans = ßss œoe Œoe æae Æae ffff fifi flfl " ’ " –

maildefcharset Overrides the default character set for email messages which don't specify one. This is mainly useful for readpst (libpst) dumps, which are utf-8 but do not say so.

localfields Set fields on all files (usually of a specific fs area). Syntax is the usual: name = value ; attr1 = val1 ; [...] value is empty so this needs an initial semi-colon. This is useful, e.g., for setting the rclaptg field for application selection inside mimeview.

testmodifusemtime Use mtime instead of ctime to test if a file has been modified. The time is used in addition to the size, which is always used. Setting this can reduce re-indexing on systems where extended attributes are used (by some other application), but not indexed, because changing extended attributes only affects ctime. Notes: - This may prevent detection of change in some marginal file rename cases (the target would need to have the same size and mtime). - You should probably also set noxattrfields to 1 in this case, except if you still prefer to perform xattr indexing, for example if the local file update pattern makes it of value (as in general, there is a risk for pure extended attributes updates without file modification to go undetected). Perform a full index reset after changing this.

noxattrfields Disable extended attributes conversion to metadata fields. This probably needs to be set if testmodifusemtime is set.

metadatacmds Define commands to gather external metadata, e.g. tmsu tags. There can be several entries, separated by semi-colons, each defining which field name the data goes into and the command to use. Don't forget the initial semi-colon. All the field names must be different. You can use aliases in the "field" file if necessary. As a not too pretty hack conceded to convenience, any field name beginning with "rclmulti" will be taken as an indication that the command returns multiple field values inside a text blob formatted as a recoll configuration file ("fieldname = fieldvalue" lines). The rclmultixx name will be ignored, and field names and values will be parsed from the data. Example: metadatacmds = ; tags = tmsu tags %f; rclmulti1 = cmdOutputsConf %f

5.4.2.3 Parameters affecting where and how we store things

cachedir Top directory for Recoll data. Recoll data directories are normally located relative to the configuration directory (e.g. ~/.recoll/xapiandb, ~/.recoll/mboxcache). If 'cachedir' is set, the directories are stored under the specified value instead (e.g. if cachedir is ~/.cache/recoll, the default dbdir would be ~/.cache/recoll/xapiandb). This affects dbdir, webcachedir, mboxcachedir, aspellDicDir, which can still be individually specified to override cachedir. Note that if you have multiple configurations, each must have a different cachedir, there is no automatic computation of a subpath under cachedir.

maxfsoccuppc Maximum file system occupation over which 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. This parameter is only checked when the indexer starts, it will not change the behaviour or a running process.

dbdir Xapian database directory location. This will be created on first indexing. If the value is not an absolute path, it will be interpreted as relative to cachedir if set, or the configuration directory (-c argument or $RECOLL_CONFDIR). If nothing is specified, the default is then ~/.recoll/xapiandb/

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

mboxcachedir Directory location for storing mbox message offsets cache files. This is normally 'mboxcache' under cachedir if set, or else under the configuration directory, but it may be useful to share a directory between different configurations.

mboxcacheminmbs 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.

mboxmaxmsgmbs Maximum mbox member message size in megabytes. Size over which we assume that the mbox format is bad or we misinterpreted it, at which point we just stop processing the file.

webcachedir Directory where we store the archived web pages after they are processed. This is only used by the Web history indexing code. Note that this is different from webdownloadsdir which tells the indexer where the web pages are stored by the browser, before they are indexed and stored into webcachedir. Default: cachedir/webcache if cachedir is set, else $RECOLL_CONFDIR/webcache

webcachemaxmbs Maximum size in MB of the Web archive. This is only used by the web history indexing code. Default: 40 MB. Reducing the size will not physically truncate the file.

webqueuedir The path to the Web indexing queue. This used to be hard-coded in the old plugin as ~/.recollweb/ToIndex so there would be no need or possibility to change it, but the WebExtensions plugin now downloads the files to the user Downloads directory, and a script moves them to webqueuedir. The script reads this value from the config so it has become possible to change it.

webdownloadsdir The path to the browser add-on download directory. This tells the indexer where the Web browser add-on stores the web page data. The data is then moved by a script to webqueuedir, then processed, and finally stored in webcachedir for future previews.

webcachekeepinterval Page recycle interval By default, only one instance of an URL is kept in the cache. This can be changed by setting this to a value determining at what frequency we keep multiple instances ('day', 'week', 'month', 'year'). Note that increasing the interval will not erase existing entries.

aspellDicDir Aspell dictionary storage directory location. The aspell dictionary (aspdict.(lang).rws) is normally stored in the directory specified by cachedir if set, or under the configuration directory.

filtersdir Directory location for executable input handlers. If RECOLL_FILTERSDIR is set in the environment, we use it instead. Defaults to $prefix/share/recoll/filters. Can be redefined for subdirectories.

iconsdir Directory location for icons. The only reason to change this would be if you want to change the icons displayed in the result list. Defaults to $prefix/share/recoll/images

5.4.2.4 Parameters affecting indexing performance and resource usage

idxflushmb Threshold (megabytes of new data) where we flush from memory to disk index. Setting this allows some control over memory usage by the indexer process. A value of 0 means no explicit flushing, which lets Xapian perform its own thing, meaning flushing every $XAPIAN_FLUSH_THRESHOLD documents created, modified or deleted: as memory usage depends on average document size, not only document count, the Xapian approach is is not very useful, and you should let Recoll manage the flushes. The program compiled value is 0. The configured default value (from this file) is now 50 MB, and should be ok in many cases. You can set it as low as 10 to conserve memory, but if you are looking for maximum speed, you may want to experiment with values between 20 and 200. In my experience, values beyond this are always counterproductive. If you find otherwise, please drop me a note.

filtermaxseconds Maximum external filter execution time in seconds. Default 1200 (20mn). Set to 0 for no limit. This is mainly to avoid infinite loops in postscript files (loop.ps)

filtermaxmbytes Maximum virtual memory space for filter processes (setrlimit(RLIMIT_AS)), in megabytes. Note that this includes any mapped libs (there is no reliable Linux way to limit the data space only), so we need to be a bit generous here. Anything over 2000 will be ignored on 32 bits machines. The high default value is needed because of java-based handlers (pdftk) which need a lot of VM (most of it text), esp. pdftk when executed from Python rclpdf.py. You can use a much lower value if you don't need Java.

thrQSizes Task queue depths for each stage and threading configuration control. There are three internal queues in the indexing pipeline stages (file data extraction, terms generation, index update). This parameter defines the queue depths for each stage (three integer values). In practise, deep queues have not been shown to increase performance. The first value is also used to control threading autoconfiguration or disabling multithreading. If the first queue depth is set to 0 Recoll will set the queue depths and thread counts based on the detected number of CPUs. The arbitrarily chosen values are as follows (depth,nthread). 1 CPU -> no threading. Less than 4 CPUs: (2, 2) (2, 2) (2, 1). Less than 6: (2, 4), (2, 2), (2, 1). Else (2, 5), (2, 3), (2, 1). If the first queue depth is set to -1, multithreading will be disabled entirely. The second and third values are ignored in both these cases.

thrTCounts Number of threads used for each indexing stage. If the first entry in thrQSizes is not 0 or -1, these three values define the number of threads used for each stage (file data extraction, term generation, index update). 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).

thrTmpDbCnt Number of temporary indexes used during incremental or full indexing. If not set to zero, this defines how many temporary indexes we use during indexing. These temporary indexes are merged into the main one at the end of the operation. Using multiple indexes and a final merge can significantly improve indexing performance when the single-threaded Xapian index updates become a bottleneck. How useful this is depends on the type of input and CPU. See the manual for more details.

5.4.2.5 Miscellaneous parameters

loglevel Log file verbosity 1-6. A value of 2 will print only errors and warnings. 3 will print information like document updates, 4 is quite verbose and 6 very verbose.

logfilename Log file destination. Use 'stderr' (default) to write to the console.

idxloglevel Override loglevel for the indexer.

idxlogfilename Override logfilename for the indexer.

helperlogfilename Destination file for external helpers standard error output. The external program error output is left alone by default, e.g. going to the terminal when the recoll[index] program is executed from the command line. Use /dev/null or a file inside a non-existent directory to completely suppress the output.

daemloglevel Override loglevel for the indexer in real time mode. The default is to use the idx... values if set, else the log... values.

daemlogfilename Override logfilename for the indexer in real time mode. The default is to use the idx... values if set, else the log... values.

pyloglevel Override loglevel for the python module.

pylogfilename Override logfilename for the python module.

idxnoautopurge Do not purge data for deleted or inaccessible files This can be overridden by recollindex command line options and may be useful if some parts of the document set may predictably be inaccessible at times, so that you would only run the purge after making sure that everything is there.

orgidxconfdir Original location of the configuration directory. This is used exclusively for movable datasets. Locating the configuration directory inside the directory tree makes it possible to provide automatic query time path translations once the data set has moved (for example, because it has been mounted on another location).

curidxconfdir Current location of the configuration directory. Complement orgidxconfdir for movable datasets. This should be used if the configuration directory has been copied from the dataset to another location, either because the dataset is readonly and an r/w copy is desired, or for performance reasons. This records the original moved location before copy, to allow path translation computations. For example if a dataset originally indexed as '/home/me/mydata/config' has been mounted to '/media/me/mydata', and the GUI is running from a copied configuration, orgidxconfdir would be '/home/me/mydata/config', and curidxconfdir (as set in the copied configuration) would be '/media/me/mydata/config'.

idxrundir Indexing process current directory. The input handlers sometimes leave temporary files in the current directory, so it makes sense to have recollindex chdir to some temporary directory. If the value is empty, the current directory is not changed. If the value is (literal) tmp, we use the temporary directory as set by the environment (RECOLL_TMPDIR else TMPDIR else /tmp). If the value is an absolute path to a directory, we go there.

checkneedretryindexscript Script used to heuristically check if we need to retry indexing files which previously failed. The default script checks the modified dates on /usr/bin and /usr/local/bin. A relative path will be looked up in the filters dirs, then in the path. Use an absolute path to do otherwise.

recollhelperpath Additional places to search for helper executables. This is used, e.g., on Windows by the Python code, and on Mac OS by the bundled recoll.app (because I could find no reliable way to tell launchd to set the PATH). The example below is for Windows. Use ':' as entry separator for Mac and Ux-like systems, ';' is for Windows only.

idxabsmlen Length of abstracts we store while indexing. Recoll stores an abstract for each indexed file. 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.

idxmetastoredlen Truncation length of stored metadata fields. This does not affect indexing (the whole field is processed anyway), just the amount of data stored in the index for the purpose of displaying fields inside result lists or previews. The default value is 150 bytes which may be too low if you have custom fields.

idxtexttruncatelen Truncation length for all document texts. Only index the beginning of documents. This is not recommended except if you are sure that the interesting keywords are at the top and have severe disk space issues.

idxsynonyms Name of the index-time synonyms file. This is only used to issue multi-word single terms for multi-word synonyms so that phrase and proximity searches work for them (ex: applejack "apple jack"). The feature will only have an effect for querying if the query-time and index-time synonym files are the same.

idxniceprio "nice" process priority for the indexing processes. Default: 19 (lowest) Appeared with 1.26.5. Prior versions were fixed at 19.

noaspell Disable aspell use. The aspell dictionary generation takes time, and some combinations of aspell version, language, and local terms, result in aspell crashing, so it sometimes makes sense to just disable the thing.

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 dicts" to see a list The default if this is not set is to use the NLS environment to guess the value. The values are the 2-letter language codes (e.g. 'en', 'fr'...)

aspellAddCreateParam Additional option and parameter to aspell dictionary creation command. Some aspell packages may need an additional option (e.g. on Debian Jessie: --local-data-dir=/usr/lib/aspell). See Debian bug 772415.

aspellKeepStderr Set this to have a look at aspell dictionary creation errors. There are always many, so this is mostly for debugging.

monauxinterval Auxiliary database update interval. The real time indexer only updates the auxiliary databases (stemdb, aspell) periodically, because it would be too costly to do it for every document change. The default period is one hour.

monixinterval Minimum interval (seconds) between processings of the indexing queue. The real time indexer does not process each event when it comes in, but lets the queue accumulate, to diminish overhead and to aggregate multiple events affecting the same file. Default 30 S.

mondelaypatterns Timing parameters for the real time indexing. Definitions for files which get a longer delay before reindexing is allowed. This is for fast-changing files, that should only be reindexed once in a while. A list of wildcardPattern:seconds pairs. The patterns are matched with fnmatch(pattern, path, 0) You can quote entries containing white space with double quotes (quote the whole entry, not the pattern). The default is empty. Example: mondelaypatterns = *.log:20 "*with spaces.*:30"

monioniceclass ionice class for the indexing process. Despite the misleading name, and on platforms where this is supported, this affects all indexing processes, not only the real time/monitoring ones. The default value is 3 (use lowest "Idle" priority).

monioniceclassdata ionice class level parameter if the class supports it. The default is empty, as the default "Idle" class has no levels.

5.4.2.6 Query-time parameters (no impact on the index)

autodiacsens auto-trigger diacritics sensitivity (raw index only). 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 auto-trigger case sensitivity (raw index only). IF the index is not stripped (see indexStripChars), 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.

maxTermExpand Maximum query expansion count for a single term (e.g.: when using wildcards). This only affects queries, not indexing. We used to not limit this at all (except for filenames where the limit was too low at 1000), but it is unreasonable with a big index. Default 10000.

maxXapianClauses Maximum number of clauses we add to a single Xapian query. This only affects queries, not indexing. In some cases, the result of term expansion can be multiplicative, and we want to avoid eating all the memory. Default 50000.

snippetMaxPosWalk Maximum number of positions we walk while populating a snippet for the result list. The default of 1,000,000 may be insufficient for very big documents, the consequence would be snippets with possibly meaning-altering missing words.

thumbnailercmd Command to use for generating thumbnails. If set, this should be a path to a command or script followed by its constant arguments. Four arguments will be appended before execution: the document URL, MIME type, target icon SIZE (e.g. 128), and output file PATH. The command should generate a thumbnail from these values. E.g. if the MIME is video, a script could use: ffmpegthumbnailer -iURL -oPATH -sSIZE.

stemexpandphrases Default to applying stem expansion to phrase terms. Recoll normally does not apply stem expansion to terms inside phrase searches. Setting this parameter will change the default behaviour to expanding terms inside phrases. If set, you can use a 'l' modifier to disable expansion for a specific instance.

autoSpellRarityThreshold Inverse of the ratio of term occurrence to total db terms over which we look for spell neighbours for automatic query expansion When a term is very uncommon, we may (depending on user choice) look for spelling variations which would be more common and possibly add them to the query.

autoSpellSelectionThreshold Ratio of spell neighbour frequency over user input term frequency beyond which we include the neighbour in the query. When a term has been selected for spelling expansion because of its rarity, we only include spelling neighbours which are more common by this ratio.

kioshowsubdocs Show embedded document results in KDE dolphin/kio and krunner Embedded documents may clutter the results and are not always easily usable from the kio or krunner environment. Setting this variable will restrict the results to standalone documents.

5.4.2.7 Parameters for the PDF input script

pdfocr Attempt OCR of PDF files with no text content. This can be defined in subdirectories. The default is off because OCR is so very slow.

pdfoutline Extract outlines and bookmarks from PDF documents (needs pdftohtml). This is not enabled by default because it is rarely needed, and the extra command takes a little time.

pdfattach Enable PDF attachment extraction by executing pdftk (if available). This is normally disabled, because it does slow down PDF indexing a bit even if not one attachment is ever found.

pdfextrameta Extract text from selected XMP metadata tags. This is a space-separated list of qualified XMP tag names. Each element can also include a translation to a Recoll field name, separated by a 'l' character. If the second element is absent, the tag name is used as the Recoll field names. You will also need to add specifications to the "fields" file to direct processing of the extracted data.

pdfextrametafix Define name of XMP field editing script. This defines the name of a script to be loaded for editing XMP field values. The script should define a 'MetaFixer' class with a metafix() method which will be called with the qualified tag name and value of each selected field, for editing or erasing. A new instance is created for each document, so that the object can keep state for, e.g. eliminating duplicate values.

5.4.2.8 Parameters for OCR processing

ocrprogs OCR modules to try. The top OCR script will try to load the corresponding modules in order and use the first which reports being capable of performing OCR on the input file. Modules for tesseract (tesseract) and ABBYY FineReader (abbyy) are present in the standard distribution. For compatibility with the previous version, if this is not defined at all, the default value is "tesseract". Use an explicit empty value if needed. A value of "abbyy tesseract" will try everything.

ocrcachedir Location for caching OCR data. The default if this is empty or undefined is to store the cached OCR data under $RECOLL_CONFDIR/ocrcache.

tesseractlang Language to assume for tesseract OCR. Important for improving the OCR accuracy. This can also be set through the contents of a file in the currently processed directory. See the rclocrtesseract.py script. Example values: eng, fra... See the tesseract documentation.

tesseractcmd Path for the tesseract command. Do not quote. This is mostly useful on Windows, or for specifying a non-default tesseract command. E.g. on Windows. tesseractcmd = C:/ProgramFiles(x86)/Tesseract-OCR/tesseract.exe

abbyylang Language to assume for abbyy OCR. Important for improving the OCR accuracy. This can also be set through the contents of a file in the currently processed directory. See the rclocrabbyy.py script. Typical values: English, French... See the ABBYY documentation.

abbyyocrcmd Path for the abbyy command The ABBY directory is usually not in the path, so you should set this.

5.4.2.9 Parameters for running speech to text conversion

speechtotext Activate speech to text conversion The only possible value at the moment is "whisper" for using the OpenAI whisper program.

sttmodel Name of the whisper model

sttdevice Name of the device to be used by for whisper

5.4.2.10 Parameters for miscellaneous specific handlers

orgmodesubdocs Index org-mode level 1 sections as separate sub-documents This is the default. If set to false, org-mode files will be indexed as plain text

5.4.2.11 Parameters set for specific locations

mhmboxquirks Enable thunderbird/mozilla-seamonkey mbox format quirks Set this for the directory where the email mbox files are stored.

5.4.3 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 default 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. There is a more complete explanation of what prefixes are in used by a standard recoll installation. In a nutshell: extension prefixes should be all caps, begin with XY, and short. E.g. XYMFLD.

[values] Fields listed in this section will be stored as Xapian `values` inside the index. This makes them available for range queries, allowing to filter results according to the field value. This feature currently supports string and integer data. See the comments in the file for more detail

[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

[queryaliases] This section also defines aliases for the canonic field names, with the difference that the substitution will only be used at query time, avoiding any possibility that the value would pick-up random metadata from documents.

handler-specific sections Some input handlers may need specific configuration for handling fields. Only the email message handler 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 handler 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 =

[queryaliases]
filename = fn
containerfilename = cfn

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

5.4.3.1 Extended attributes in the fields file

Recoll processes 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.

Name translations are set as `xattrname = fieldname`. They are case-sensitive. E.g. the following would map a quite an extended attribute named "tags" into the "keywords" field: `tags = keywords`.

Entering an empty translation will disable any use of the attribute.

The values from the extended attributes will not replace the data found from equivalent fields inside the document, instead they are concatenated.

Special case: an extended attribute named `modificationdate` will set the `dmtime` field (document date) only if it is not set by an internal document field (e.g. email `Date:`).

5.4.4 The mimemap file

`mimemap` specifies the file name extension to MIME type mappings.

For file names without an extension, or with an unknown one, recent Recoll versions will use libmagic. Older versions would execute a system command (**file** `-i`, or **xdg-mime**) will be executed to determine the MIME type (this can be switched off, or the command changed inside the main configuration file).

All extension values in mimemap must be entered in lower case. File names extensions are lower-cased for comparison during indexing, meaning that an upper case mimemap entry will never be matched.

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

```
[~/.kde/share/apps/okular/docdata]
.xml = application/x-okular-notes
```

The recoll_noindex mimemap variable has been moved to recoll.conf and renamed to noContentSuffixes, while keeping the same function, as of Recoll version 1.21. For older Recoll versions, see the documentation for noContentSuffixes but use recoll_noindex in mimemap.

5.4.5 The mimeconf file

The main purpose of the mimeconf file is to specify how the different MIME types are handled for indexing. This is done in the [index] section, which should not be modified casually. See the comments in the file.

The file also contains other definitions which affect the query language and the GUI, and which, in retrospect, should have been stored elsewhere.

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

The [categories] section defines the groupings of MIME types into categories as used when adding an rclcat clause to a query language query. rclcat clauses are also used by the default guifilters buttons in the GUI (see next).

The filter controls appear at the top of the **recoll** GUI, either as checkboxes just above the result list, or as a dropbox in the tool area.

By default, they are labeled: media, message, other, presentation, spreadsheet and text, and each maps to a document category. This is determined in the [guifilters] section, where each control is defined by a variable naming a query language fragment.

A simple example will hopefully make things clearer.

```
[guifilters]

Big Books = dir:"~/My Books" size>10K
My Docs = dir:"~/My Documents"
Small Books = dir:"~/My Books" size<10K
System Docs = dir:/usr/share/doc
```

The above definition would create four filter checkboxes, labelled Big Books, My Docs, etc.

The text after the equal sign must be a valid query language fragment, and, when the button is checked, it will be combined with the rest of the query with an AND conjunction.

Any name text before a colon character will be erased in the display, but used for sorting. You can use this to display the checkboxes in any order you like. For example, the following would do exactly the same as above, but ordering the checkboxes in the reverse order.

```
[guifilters]

d:Big Books = dir:"~/My Books" size>10K
c:My Docs = dir:"~/My Documents"
b:Small Books = dir:"~/My Books" size<10K
a:System Docs = dir:/usr/share/doc
```

As you may have guessed, The default [guifilters] section looks like:

```
[guifilters]
text = rclcat:text
spreadsheet = rclcat:spreadsheet
presentation = rclcat:presentation
media = rclcat:media
message = rclcat:message
other = rclcat:other
```

5.4.6 The mimeview file

`mimeview` specifies which programs are started when you click on an Open link in a result list. E.g.: 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, e.g.: *`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 (e.g.: to extract a subdocument from a container).
- **%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. If it is set, a significant term will be chosen in the query, and %p will be substituted with the first page where the term appears. Can be used to start the editor at the right page for a match or snippet.
- **%l** Line number. Only significant for document types with relevant line breaks, mostly text/plain and analogs. If it is set, a significant term will be chosen in the query, and %p will be substituted with the first line where the term appears.
- **%s** Search term. The value will only be set for documents with indexed page or line numbers and if %p or %l is also used. 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** 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.

5.4.7 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 example 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
```

5.4.8 Examples of configuration adjustments

5.4.8.1 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.

5.4.8.2 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
  ```

  Or if the files are mostly text and you don't need to process them for indexing:

  ```
  application/x-blobapp = internal text/plain
  ```

- 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/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`* handler should be an executable program or script which exists inside `/usr/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 an input handler.