Xapian developer guide1.4

Release 1.4.3

Xapian Documentation Team
Contributors

August 19, 2017

Contents

Xapian is an open source search engine library, which allows developers to add advanced indexing and search facilities to their own applications. This manual aims to explain how to work on and contribute to Xapian itself; if you want to use Xapian in your own project, you should look at our Xapian user manual.

Note: This is very early days for this guide, so please let us know any issues you spot or how we can improve it in any way. There's a lot of additional information about developing for Xapian in the HACKING file in xapian-core which we hope to move here in future, and in the meantime that's a good place to look for information on coding style, writing and running tests, and so on.

CHAPTER 1

Getting in touch

The Xapian community typically works "in the open", via mailing lists and an IRC channel:

- Our mailing lists are open for anyone to join, although (because we don't want to relay spam to everyone) if you aren't subscribed to the list someone will have to manually approve your message. Please be patient and don't resend a message just because it doesn't appear right away.
- We're on #xapian on `irc.freenode.net`. IRC is a simple text chat system where many members of the community hang out; although because we're distributed around the world, you may not get an instant response. That doesn't mean we're ignoring you, so either hang on for a reply, or you can use the mailing list instead.

CHAPTER 2

Contents

Getting started

Note: Currently this guide is written assuming you are either developing on something like Unix (probably either Linux or OS X). You can use software suchas Virtual Box to run a 'virtual' Linux machine on another operating system; in this case we recommend using the most recent Ubuntu LTS release.

It should be possible to build and develop Xapian on Windows, but we currently don't have any documentation on doing so, or any active developers with suitable experience.

Getting the source code

First off, let's make sure you have a copy of the Xapian source and can build it and run the tests. This is generally a little different to if you're just installing Xapian to use it, because you'll be working with the entire source tree rather than individual pieces. The Xapian build system has some support for this, but let's get you a copy of everything first:

```
$ git clone git://git.xapian.org/xapian
$ cd xapian
```

This will 'clone' a complete copy of the Xapian source code, including not only the core library but also the variable language bindings (for use from Python, Lua, Ruby and so on) and the self-contained web search system 'Omega'. It also contains all the tests for those various components.

Installing the dependencies

Debian / Ubuntu

For a recent version of Debian or Ubuntu, this command should ensure you have all the necessary tools and libraries:

```
$ apt-get install build-essential m4 perl python zlib1g-dev uuid-dev \
  wget bison tcl libpcre3-dev libmagic-dev valgrind ccache eatmydata \
  doxygen graphviz help2man python-docutils pngcrush python-sphinx \
  python3-sphinx mono-devel openjdk-6-jdk lua5.2 liblua5.2-dev \
  php5-dev php5-cli python-dev python3-dev ruby-dev tcl-dev
```

OS X

You need to install Apple's XCode tools, which contain their compiler, debugger and various other tools. You can do that from within the AppStore.

We recommend using homebrew to install and manage additional libraries and tools on OS X. Once you've installed XCode and homebrew, you can get all the dependencies you need for Xapian using:

```
$ brew install libmagic pcre php56 python python3 lua ruby mono \
  doxygen help2man graphviz pngcrush
# and some python-specific documentation tools
$ pip install sphinx docutils
$ pip3 install sphinx
```

(We install documentation tools for both python2 and python3, in the same way we build the bindings for both of them.)

Building Xapian

Bootstrapping the code

Xapian needs to set up a few things with a fresh clone of the code, as well as downloading and building some tools for which we require very precise versions. You should run this command in the `xapian` directory that was created earlier when you cloned the source code:

```
$ ./bootstrap
```

To download tools, bootstrap will use `wget`, `curl` or `lwp-request` if installed. If not, it will give an error telling you the URL to download from by hand and where to copy the file to.

Note: As well as installing some tools, bootstrap will also run `autoreconf` on each of the checked-out subdirectories, and generate a top-level `configure` script. This configure script allows you to configure xapian-core and any other modules you've checked out with a single simple command, such that the other modules link against the uninstalled xapian-core (which is very handy for development work and a bit fiddly to set up by hand). It automatically passes `--enable-maintainer-mode` to the subprojects so that the autotools will be rerun if `configure.ac`, `Makefile.am`, etc are modified.

Warning: If you are tracking development in git, there will sometimes be changes to the build system sources which require regeneration of the generated makefiles and associated machinery. We aim to make the build system automatically regenerate the necessary files, but in the event that a build fails after an update, it may be worth re-running the bootstrap script to regenerate the build system from scratch, before looking for the cause of the error elsewhere.

Configuring the code

Configuring the code is mostly about Xapian's build system automatically detecting where all its dependencies are on your computer, so it knows how to use them. However there are various options that allow you to either override the autodetection (for instance if you wanted to build python bindings against a particular version of python) or change some defaults. For now, however, we'll just run it accepting all its defaults:

```
$ ./configure
```

Note that on OS X you probably want to turn off the Perl and TCL8 bindings when developing, as there are some complexities when developing against the system versions, and the homebrew versions are slightly awkward:

```
$ ./configure --without-perl --without-tcl
```

Building Xapian

Building Xapian is just a matter of typing:

```
$ make
```

First it will build xapian-core, the core library. Then it will build Omega and the language bindings, using the version of xapian-core you've just built, but not yet installed. (This is the bit that causes some problems on OS X if you use system versions of any of the languages.)

Running the tests

Xapian has a comprehensive test suite, and it's a good idea to get into the habit of running it. From the top of the clone, just run:

```
$ make check
```

Again, the tests for xapian-core are run first, then Omega and then the language bindings. If any test fails, the build system will stop there.

Summary

Now you've got everything working, you probably want to look at writing code, or if you're trying to fix a bug then you might want to learn about debugging Xapian.

Todo

The other sections of the manual haven't been written yet, so this part isn't terribly helpful. Sorry!

Contributing to Xapian

Xapian is an open source project, depending on a community of volunteers. There are lots of ways of contributing to Xapian:

- Join our mailing lists or IRC channel (#xapian on Freenode) and help people out.
- Talk or write about Xapian. By explaining how you've used Xapian you not only help spread the word, but also might learn more about Xapian itself.
- Let us know (either via our bug tracker or by the mailing lists or IRC) if you run into any problems with our documentation, or with bugs you come across while using Xapian.
- Help improve our documentation, either by suggesting changes or by writing up something on our list of missing documentation.
- Contribute new features by working on one of our project ideas
- Tackle an existing bug or feature request, or something you yourself want to see in Xapian.

- Help others get their changes into shape for inclusion in Xapian.

One of the things this guide will do is take you through the process of getting comfortable with the Xapian codebase and submitting your first "patch". There's also lots of more detailed information if you want to get more deeply involved in writing code for Xapian.

Licensing your contributions

If you want a patch to be considered for inclusion in Xapian, you must own the copyright on your changes. Employers often claim copyright on code written by their employees (even if the code is written in their spare time), so please check with your employer if this applies. Be aware that even if you are a student your university may try and claim some rights on code which you write.

Patches which are submitted to Xapian will only be included if the copyright holder(s) dual-license them under each of the following licences:

- GPL version 2 and all later versions (see the file COPYING in `xapian-core` for details).
- MIT/X license:

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The current distribution of Xapian contains many files which are only licensed under the GPL, but we are working towards being able to distribute Xapian under a more permissive license, and are not willing to accept patches which we will have to rewrite before this can happen.

Advice for new contributors

New to Xapian (or even open source)? Don't worry! Here we try to guide you through your first contribution, but if anything is unclear or you want to ask a question, please *get in touch*. This may look a bit daunting the first time, but we're here to help, and a lot of the details will become natural over time.

Checking out and building Xapian

A good way to start learning about Xapian is to check out the code. and get it to build. It's better to use the latest code from the repository rather than a release, as that's what we want the projects to be based on.

We recommend you use Linux or another UNIX-like system for development work, as we're better set up for development on such platforms. In particular we use them ourselves, so can more easily help with any set up issues you

may encounter. If you want to run Linux (perhaps virtualised) for development and have no existing preference, we suggest Debian or Ubuntu as our documentation covers these well.

It can take a while to get the code if your network connection is slow, and it may take a while to build and run the testsuite if your computer is slow - while you are waiting, you might want to make a start on the next section.

Learn about Xapian's API

It's a good idea to get familiar with Xapian by going through the user guide. The online version has examples in Python, but you can also grab the source and build for other languages; most example code is also available in C++ and PHP.

For more details on individual classes, you may want to look at the automatically generated API documentation. If you're building from git, this will be built for you in `xapian-core/docs`; the API may have some changes between the stable release documented on the website and the latest version in git.

Get familiar with the code

If you're going to be writing code, it's a good idea to read some of Xapian's existing sources, particularly in the main library (`xapian-core`). When you come to write your own, you'll want to follow the style of how the current code works, both in terms of layout (where spaces go and so on) and how we use various C++ language features.

Note: There is *further information on this* that hasn't yet been added to this guide.

Picking something to start with

It can be difficult sometimes to find a place to start, so here are some suggestions:

- Start small.

  By picking a small contribution first, other people can help you with details of how Xapian's documentation, code and so on work. It's much easier to get feedback on a small change to start off with.

  If documentation is your thing, then you might like to take a look at our list of missing documentation.

  On the features side, our'"bite-sized" projects'_ are intended to be suitable for someone new to Xapian to pick up. We've tried to have a range of things to work on across different parts of Xapian.

Note: No change is too small to consider! Some people's first contributions to an open source project are fixing a single stray letter in documentation, or adding a line break where one is needed.

- Pick something you care about, or which you already have some knowledge of.

  For instance, if you've been using Omega, you might want to pick up a small bug or feature for that. Or if you've studied (or are studying now!) different Information Retrieval weighting schemes, you might want to implement one of the ones we don't currently support.

- Don't be afraid to *ask for help*.

  Please pop onto the mailing list or IRC if you need any help in getting started or picking something to work on.

Do some work

Add or correct some documentation, fix a bug or implement a new feature! You'll probably find our *suggested workflow* helpful. We also have *detailed information* on contributing your changes back to Xapian for inclusion in future releases.

A helpful workflow

If you're working on a bug or new feature, you'll follow something like the following steps. If you haven't worked on Xapian before, it's a good idea to adopt this workflow.

Claim the ticket in trac

Trac is where we keep track of proposed features and known bugs. You'll have to sign up for an account, including verifying your email address, in order to make changes. (This is to prevent unwanted spam on the wiki and in tickets.)

To 'claim' the ticket so others know you're working on it, you can either reassign the ticket to yourself and then claim it, or you can just add a comment saying you're working on it. If you haven't already done so, now's a good time to drop a message either in IRC or to the mailing list saying what you'l be working on.

Note: If there isn't a ticket for what you want to work on, you should create one. (If you're creating one from something on our project list on the wiki then it's good practice to update the wiki to link to the new ticket, so other people can find it easily.)

Create a branch in your local git repository

You want somewhere where you can keep your changes until they're ready, and that won't get confused with anyone else's work. In git these are called branches.

Before you create your branch, you will generally want to make sure your local copy of the code is up to date with the central repository. Generally you can this as follows:

```
$ git checkout master
$ git pull origin/master
```

You can check create your new branch:

```
$ git checkout -b feature-x
```

The branch name doesn't really matter, but you'll probably find it easiest to name it something related to the work that you're doing.

Write a plan

Even the smallest contribution is worth thinking about before starting to type, and with larger changes it's all but essential. If you put your plan in the ticket on trac, it will help when someone else comes to review your patch. Also, if you ask for help, having a plan that someone else can refer to can let them see how you're thinking, so they can provide some useful advice or recommendations more easily.

For a larger piece of work, you ideally want to be able to break down the work into smaller "sub-projects" which can be completed, reviewed and released in turn. (If you're familiar with agile development, think of it as a series of development sprints.)

When planning work on a bug or new feature, you should bear in mind that there are a number of *standards we work to* when accepting changes into Xapian, and the more of them you cover yourself the easier it will be to get your changes into a future release. In particular, it's worth thinking about documentation and tests in advance.

About the only time you don't need to write a plan is when you're making a small change to some documentation, correcting a spelling mistake or making something clearer.

Make your changes!

Now you can start making changes. There's a host of *other information that can help you* if you're writing code. As usual, there are *other people in the community who can help* if you need.

If you discover partway through that your plan isn't working, it's a good idea to stop and write a new plan. You'll have learned something about the problem that you didn't know when you wrote the initial plan, so taking the time to think things through from the beginning can often unblock you and let you start moving again. Of course, if it doesn't clear things up, that's a good time to ask the community for help.

Make commits out of your changes

This is where you probably want to know a little more about git. A very quick introduction is that you first "stage" changes, then you "commit" those changes. You don't have to stage all your changes at once, which means you can keep small notes or parts of future work lying around while you're creating your commits, without them creeping into your commits and confusing matters.

To stage changes for your next commit:

```
$ git add -p
```

The `-p` tells git that you want it to find all the changes, then one by one ask you if you want each staged. Just type `y` to stage a change (it calls them "hunks"), or `n` to skip it this time round. If the file is completely new, you can run `git add <path>` to stage the whole file. (There are lots of other options available in `git add -p`; if you type `?` then it will explain what they all do.)

Then to make a commit:

```
$ git commit -v
```

git will open your editor for you to write a commit message. The `-v` means that your changes will be shown at the bottom of the editor (although they won't be included in the commit message), which helps you do a final check that you're committing only what you want, and everything that is needed.

A good commit in git relies on getting two things right: changes that do a single thing, and a commit message that describes the thing clearly. We have some quick tips on each.

Good commits

Structuring your changes into commits can take a bit of getting used to, but makes it a lot easier for other people to review, both before we merge into Xapian and in the future when someone – which might be you! – needs to understand why a change was made in the past, to help them do whatever work they need to do. There's a good article by Anna Shipman that may help you think about structuring your changes into a set of commits that are easy for others to read.

- Only make *one change* per commit, and make the *whole change* in that commit – you don't want to end up with essential bits of code in a different commit.

Many people struggle with this at first, and it can be difficult to get into the habit of thinking in terms of the distinct changes to the system rather than in terms of how you did the work. A plan here can help structure your commits once you've finished working.

One of the reasons we suggest using `git add -p` is that it enables you to review every single change that goes into a commit, which can help you put only the right things into it.

- Avoid committing code that has been commented out. If we need it again, it's in the git history.

Good commit messages

Writing a great commit message is important both for people reviewing your code now to help get it ready for a future Xapian release, and for when someone needs to understand how and why a particular change was made, months or years in the future – when that someone might be you!

- Start with a short (50 characters) summary line.

  git (and github) are designed to work better this way. The summary should be in the imperative ("Fix bug on OS X" rather than "Fixed bug on OS X"). This matches git's automatic messages around merges, reverts and so on.

- Follow that with more detail as needed, wrapping long lines at 72 characters (one exception is that long URLs are best not wrapped).
- Describe the effect, not the code. The important thing is for people to be able to read the commit message and understand what you were trying to achieve when you made those changes. That way, if someone needs to work on that part of the code in future, they can understand the purpose of it, and not accidentally remove some useful functionality. (Tests help here, but the commit message is very important.)

There are a few articles around on writing good commit messages; Thoughtbot's "5 Useful Tips For A Better Commit Message" has some good advice.

> **Warning:** Lots of online git tutorials will tell you to write commit messages on the command line, using `git commit -m <message>`. If you do that, you'll never write really good commit messages.

For more details on using git, there are free books and resources online, such as Pro Git.

Contribute your changes

We have *detailed information* to help you here.

Contributing changes

Some things that we look for

Beyond looking for changes that improve Xapian, code that works and so forth, there are a number of things that we aim for when accepting changes. This then is a list of good practices when contributing changes.

Code that compiles cleanly and looks like existing code

We like Xapian to compile without any warnings. In "maintainer mode", which will be how you're building Xapian if you're working from a git clone, all warnings will actually become errors. You should fix the problems rather than change the compilation settings to ignore these warnings.

We don't currently have a formal coding standards document, so you should try to follow the style of the existing code. In particular, it's a good idea to pay close attention to code alignment and where we have spaces.

Updated documentation

If you add a new feature, please ensure that you've documented it. Don't worry too much about the language you use, or if English isn't your first language. Others can help get the documentation into shape, but having a first draft from the person who wrote the feature is usually the best way to get started.

- API classes, methods, functions, and types should be documented by documentation comments alongside the declaration in `include/xapian/*.h`.

  These are collated by doxygen – see doxygen's documentation for details of the supported syntax. We've decided to prefer to use `@` rather than `\` to introduce doxygen commands (the choice is essentially arbitrary, but `\` introduces C/C++ escape sequences so `@` is likely to make for easier to read mark up for C/C++ coders).

- The documentation comments don't give users a good overview, so we also need documentation which gives a good overview of how to achieve particular tasks.

  If there's relevant documentation already in the user guide, then you should update that. For completely new features, you should create either a "how to" or an "advanced feature" document in the user manual, so that people can get started without having to start with the API documentation.

- Internal classes, etc should also be documented by documentation comments where they are declared.

Automated tests

If you're fixing a bug, you should first write a regression test. The test will fail on the existing code, then when you fix the bug it will pass. In the future, the test will make sure no one accidentally re-introduces the same bug.

If you're adding a new feature, you'll want to write tests that it behaves correctly. Thinking about the tests you need to write can often help you plan how to implement the feature; it can also help when thinking about what API any new classes or methods should expose.

Updated attributions

If necessary, modify the copyright statement at the top of any files you've altered. If there is no copyright statement, you may add one (there are a couple of Makefile.am's and similar that don't have copyright statements; anything that small doesn't really need one anyway, so it's a judgement call). If you've added files which you've written from scratch, they should include the GPL boilerplate with your name only.

If you're not in there already, add yourself to the `xapian-core/AUTHORS` file.

Consider backporting bug fixes

If there's an active release branch, please check if the bug is present in that branch, and if the fix is appropriate to backport - if the fix breaks ABI compatibility or is very invasive, you may need to fix it in a different way for the release branch, or decide not to backport the fix.

License grant

We ask everyone contributing changes to Xapian to *dual-license* under the GPL (which Xapian currently uses) and the MIT/X license (which we would like to move to in future). The simplest way to do this is to drop an email to the xapian-devel mailing list stating that you own the copyright on your changes and are happy to dual-license accordingly.

Submit your patch

There are two ways of working, depending on whether you want to use Github or not. In both cases, review and acceptance of the changes will generally go more easily if you've included tests, updated documentation and so on *as discussed earlier*.

Attach a patch directly to the trac ticket

We find patches in unified diff format easiest to work with. `git diff` produces the right output for a single commit (or `git format-patch` for a series of commits).

Someone from the community will then be able to review the patch and decide if it needs further work before integrating. If so, they'll leave comments on the trac ticket (trac will generally email you if you're marked as the owner, or you can explicitly add yourself to the "cc" list for a ticket).

Open a Pull Request on github

Github pull requests provide a web-based interface for review and discussion of changes before they are accepted into Xapian. Github's documentation explains how you can go about opening them.

If your patch is a sub-project in a larger piece of work, then it's important not to assume the patch is fine as it stands and to immediately start the next sub-project. Instead you should concentrate on completing the sub-project before moving on. Since you'll almost always have to wait at least a little time to get feedback on any changes, you may want to put the code and tests up while still working on documentation.

You should add further changes to pull requests by creating additional commits locally, typically by using `git commit --fixup`, and then pushing the branch up to Github. Only once everything's been approved should you squash your commits together to keep the history clean.

Note: Once you've opened a pull request, you shouldn't have to close it until it's merged (in which case we'll generally close it for you). Even if you need to redo some work, you can either add fixup commits or (with agreement from whoever is reviewing the PR) unwind your work and create completely new commits, force pushing to replace the previous commits in the pull request.

It makes it much harder to review if you close a pull request in the middle of a review only to open another with similar code.

License

This license applies to all documentation and example code in this book.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.