The LXR User's Manual

for version 1.2

Revision history

Author	Date	Rev	Comment
P. Gerlier ajlittoz@users.sf.net	2011-03-05	0.8	Initial version (for release 0.9.8) and considered as beta release: chapter 1 based on INSTALL 0.9.8; following chapters, new material Published for 0.9.9 release
P. Gerlier	2011-04-01		Adding 0.10 features
P. Gerlier	2011-04-30		Updating 0.10 after bug fixes
P. Gerlier	2011-12-23 2012-01-12	0.9	Beginning revision for 0.10; changing styles End of revision
P. Gerlier	2012-01-17		Index added; changed licence to GNU FDL
P. Gerlier	2012-02-05	0.99	Upgrade to 0.11 features; revision 0.99 for beta status
P. Gerlier	2012-03-14	1.0	Index and review completed (document revision bumped at 1.0)
P. Gerlier	2012-09-22	1.1	New features for 1.0 (major version number switch considering the number of changes and new features)
P. Gerlier	2013-01-18	1.2	Adding 1.1 features
P. Gerlier	2013-01-24	1.3	Describing web server set-up (interactive configuration and customisation), new CSS style for disabled mode buttons
P. Gerlier	2013-03-17	1,4	Upgrading to 1.2 include syntax

Licence statement

Document name

The file name for this document is structured as `T-SR-L-DR.f` where:

- `T` is a short title (like `LXRUserManual`),
- `SR` is the software release number associated with this document (like `0.10)`,
- `L` is the ISO 639 alpha 2 language code with optional country variant (like `en_UK`),
- `DR` is the document revision number (like `1.0`),
- `f` is the file format or file name extension (like `odt` for Open Document Format or `pdf`).

Table of Contents

0 Foreword

0.1.a. What is LXR?

The acronym LXR used to stand for *Linux Cross Referencer*. LXR arose from a need to navigate "comfortably" inside the Linux kernel source during its development because of lack of adequate written documentation, constantly evolving architecture, uncoordinated user contributions, …

The first version was really focused on the Linux kernel with an emphasis on C language parsing. But such a tool has nothing specific to Linux or to operating system development. It can in fact be used against any source tree. It was thus extended to accept many programming languages. It was also extended to deal with source trees managed by some *source code management*[1] systems or SCMs, namely ***CVS***, ***Git***, ***Mercurial***, ***Subversion*** or ***BitKeeper***, without having to extract the files to real directories.

LXR usage can be traced back as early as 1994[2]. The present software is the direct descendant of this venerable ancestor. An offset specifically dedicated to the Linux kernel (***LXRng***) has been created to deal with this huge source-tree (see http://lxr.linux.no). The technology is quite different; source code is totally new. The product remains somewhat experimental even now and does not offer all functionality of the main line.

It would be nice if the initial developers could contribute with a short note on LXR history.

LXR presents on-screen a source file with editing improvements:

- Display through an HTML browser

  All the power of HTML and CSS is available. Moreover, no need to learn usage of a new application, you just launch your usual web browser. You can display several source files or directories simultaneously in different tabs or windows.

- Colour highlighting on syntactic objects

  Comments, keywords, variables and functions are displayed differently to be easily identified.

- User symbols are clickable

  Hyperlinks are build under these symbols to open a cross-reference page (LXR first goal), from where you jump to any occurrence with another click.

- Explicit search for identifiers

  You enter the name of a user symbol and the cross-reference page for it opens. It is really the same feature as the previous one, which could be considered an implicit or contextual case.

- Side-by-side presentation of two versions of the same file

[1] SCMs are also frequently referred to as *version control systems* or VCS's.
[2] This date needs to be checked with the initial developers if they can show up.

Differences are visually indicated with background colours. Hyperlinks are kept.

- Free-text search under certain restrictions

  If certain conditions are met, you can search (with pattern-matching if you like) any string of characters in your source-tree. These limitations are imposed by the free-text search tools, not by LXR itself.

0.1.b. What is the technology under the hood?

LXR is typical of *NIX products. It does not reinvent the wheel. It relies on “out-of-the-shelf” tools to do its job. It is some kind of supervisor launching services and aggregating the results.

The components are:

1. a database to store the user symbols and their cross-references
2. a language parser
3. a web server to display and navigate the source-tree
4. a free-text search engine for arbitrary inquiries

LXR is the “glue” between these components. It is written in *Perl*, thus you also need a *Perl* interpreter.

0.1.c. How to use it?

You meet LXR on three different occasions. Each one has a specific context and needs different skills:

- You install LXR once. The next chapter tells you how.
- You initialise the database every time you modify the source-tree. This is done with a script provided with LXR. You may need to update the configuration file when you add a new source-tree or a new version to an existing one.
- You launch your web browser whenever you want to read through the source-tree.

These tasks are listed in order of increasing frequency and decreasing effort.

0.1.d. Getting help

If you can't get LXR to work then have a look on the web site http://lxr.sourceforge.net[3], in particular at the “Troubleshooting” page http://lxr.sf.net/en/troubleshooting.shtml. If you do not find your answer, check the forum archives http://sf.net/p/lxr/discussion/?source=navbar or the mailing

[3] All `sourceforge.net` URLs can be shortened to `sf.net`, *e.g.* `lxr.sourceforge.net` to `lxr.sf.net`

list. If you are still facing difficulties, ask the developers at lxr-general@lists.sourceforge.net.

Note:

Due to spam flooding, it has been necessary to remove public access to the mailing lists. But you can register for free on *SourceForge* and you will be granted membership within a few hours or days.

Also, help improving this manual. Report any inaccuracy, error or presentation misfit on the forum or mailing list.

I Installing LXR

This chapter gives you instructions to succeed in installing LXR to display a single source tree. This is the simplest case and LXR can do much more.
These instructions were formerly contained in the INSTALL document that comes with every LXR distribution. Now the INSTALL document is only a streamlined version of this section.

1.1. What is needed?

As mentioned in the foreword, LXR is based on widely available components. The majority of them is already installed in common Linux distributions. This is a check-list of your configuration.

In all cases, the preferred method of installation of these tools is a package download of your favourite distribution, so that you do not bother with compiling and configuring the tool, which may require expertise beyond the need of using LXR.

LXR depends on the following tools:

1. A ***Perl*** interpreter

   LXR is written in ***Perl***.

   It is installed by default on most Linux platforms, but you must check version is at least 5.10[4]. To check, type in a terminal[5]:

   ```
   $ perl -v
   This is perl 5,  version 14, subversion 2 (v5.14.2) built for …
   ```

   If you get a message like `Unknown` command or if you read a number smaller than 10 after `version`, install a new version of the interpreter.

2. At least version 5 of the ***exuberant ctags*** program

[4] LXR release 0.10 needed *Perl* version 5.10, while earlier releases only needed version 5.8. Release 0.11 was modified to be again compatible with 5.8 considering the widespread use of the *Perl* interpreter version 5.8. Unhappily, parsing correctness requires the use of features introduced in version 5.10; it is thus impossible to maintain compatibility with older *Perl* interpreters. Note that *Perl* 5.10 was released in December 2007 and, as of this writing, current version is 5.16.
Technically, the parser has been modified to allow alternatives in the region delimiters (see 6.3.e Source file fragment categories). This was mandatory for the *Ruby* language. But do not assume other languages will always run without the patch. The change solves non trivial bugs in instruction handling and gives more power to describe language constructs. Consequently, it will certainly be generalised.
As a side-effect of giving up 5.8 compatibility, programming LXR will be made easier.

[5] Italic bold text is *user input*, roman light is computer output.

ctags is the language parser and dictionary builder used by LXR.

It is also generally installed by default. To be sure, type in a terminal:

```
$ ctags --version
Exuberant Ctags 5.8, Copyright (C) 1996-2009 Darren Hiebert
…
```

As a last resort, it is available from *SourceForge* at http://ctags.sourceforge.net (or http://sf.net/projects/ctags).

3. A relational database

   For performance reasons, LXR stores its dictionary (*i.e.* symbols and their cross-references) in a database because they are optimised for high volume retrieval.

   MySQL 4.x/5.x (http://www.mysql.com), ***Oracle*** (http://www.oracle.com), ***PostgreSQL*** (http://www.postgresql.org) and ***SQLite*** (http://www.sqlite.org) are supported.

   You will also need the right *Perl* ***DBI*** drivers for your particular database, usually available from CPAN (if not in your distribution).

   How do you choose between them?

   You may have no choice if your site has a database policy. You only have to configure LXR for the correct database engine.

   SQLite is very easy to install, requires no configuration and does not use much resources. But it has limitations, among which the most important is all data and indexes are stored in a single ordinary file. This means ***SQLite*** is unlikely to be able to cross reference the Linux kernel, even a single version. It is an adequate choice for a personal small-sized project on an individual computer, however indexing time may grow rapidly with project size.

   MySQL is widely available and supports heavy traffic. Its capabilities largely exceed LXR needs. It is faster than *SQLite* and scales very well on big projects like the Linux kernel.

   PostgreSQL is a good choice. It also scales very well on big projects. Its capabilities are a bit behind *MySQL* (but it does not matter as far as LXR is concerned) and it is nearly as fast as *MySQL*. It is also recommended for big projects, even if its configuration is not as easy as *MySQL*.

   Oracle is under a proprietary license. Consequently, it has not possible to the present maintainer to test this database engine. Bug reports are welcome.

4. A web server

   LXR outputs its results as HTML text to be read with any browser.

   Apache httpd (http://httpd.apache.org) with ***mod_perl*** (http://perl.apache.org) is recommended. LXR works with *Apache 1* and *Apache 2* provided ***mod_version*** is installed..

   Starting with release 0.11, ***lighttpd***, a resource friendly and very fast server, is also experimentally supported. Please send feedback to improve this support.

5. For free-text searching, either ***Glimpse*** (http://webglimpse.net) or ***Swish-e*** (http://swish-e.org) version 2.1 or later

   Swish-e is fully GPL'ed, while *Glimpse* is only free for non-commercial use. Thus you have little chance to find the latter among your distribution packages, but installing it is quite automated.

   How do you choose between them?

   They do not fulfil the same purpose. ***Glimpse*** will give you every reference and a hyperlink to the line containing the reference. ***Swish-e*** will give you only the files containing the text sorted by order of *pertinence* (according to its internal criteria, roughly the number of occurrences). Both need pre-indexing, but the amount of data kept in the indexes varies.

   As a rule of thumb, independently of personal taste, if you want very accurate references, use ***Glimpse***. If you are satisfied with only the file name for occurrences, or if you have a huge source tree and you observe excessive search time with ***Glimpse***, use ***Swish-e***.

   Glimpse is fit for in-depth source code analysis while ***Swish-e*** is more oriented towards data-mining on the web.

   Unhappily, ***Glimpse*** can handle only plain files. ***Swish-e*** is a little more generic in that it can handle plain files and *CVS* repositories. But none of them can cope with *Git* or *BitKeeper* repositories.

   Glimpse builds macro-blocks to speed up searches, while ***Swish-e*** duplicates your source files. ***Swish-e*** footprint is equal to the size of your source-tree while ***Glimpse*** footprint is a 5-10% of that size.

6. The ***Perl*** database driver interface ***DBI*** and the relevant ***DBD*** driver for the database you're using

   If not already present, they can be installed via CPAN. See http://dbi.perl.org/index.html for more info.

7. The ***Perl File::MMagic*** module, available from CPAN

   This module is used to infer the type of file from some “signatures” in its content. This allows the cross-reference generator to discard non-text files. You really need it with ***Swish-e*** but LXR will not compile without it.

8. If storing your source in a VCS you need to install the appropriate management system as well.

- ***CVS***: ***rcs***
- ***git***: usually packaged
- ***Subversion***: usually packaged
- ***Mercurial***: usually packaged *(experimental feature starting in 1.1)*

- ***BitKeeper***: ***bk*** (availability unknown)and ***Digest::SHA*** module (available from CPAN)

Remember: always try first to get the tool from a package of your distribution!

1.2. Create LXR installation directory

Decide where you want LXR installed. Consider if you just run a personal tool or offer system-wide service.

In the former case, you can play harmlessly in your user home directory.

In the latter case, you need administrator privileges to create files in the utility directories. A good choice could be */usr/local/share/lxr/*. If you have no privileges, you are left with installation in your home directory but you incur no loss in functionality.

First, download the tarball from the LXR project site at http://sourceforge.net/projects/lxr. Then expand the tarball in the LXR distribution into the selected directory of your choice like this:

- For a first try, in your own home directory:

  ```
  $ cd ~
  $ tar -zxf /path/to/downloaded/tarball/lxr-x.y.tgz
  $ mv lxr-x.y lxr
  ```

  The last command gets rid of release numbers in the LXR directory name.

- For system-wide service installation (to be attempted once you have run LXR successfully from your home directory):

  ```
  $ cd /usr/local/share
  $ tar -zxf /path/to/downloaded/tarball/lxr-x.y.tgz
  $ mv lxr-x.y lxr
  ```

  (as *root* or with appropriate permissions through `su` or `sudo`).

Then enter this directory, hereafter called *LXR root directory*:

```
$ cd lxr
```

At this point, the *LXR root directory* contains

```
$ ls -F
custom.d/  doc/      ident*  LXRimages/  scripts/  source*     tests/
diff*      genxref*  lib/    robots.txt  search*   templates/
```

a set of **directories**:

- *doc/*
  documentation pertaining to this release, notably *INSTALL*, a digest of this chapter, and *CHANGES*, highlighting the main changes since the last release;
- *lib/*

all support routines;

- *LXRimages/*
  graphic material specific to LXR;
- *scripts/*
  a collection of scripts to automate installation;
- *templates/*
  a collection of template bits for the various configuration files and HTML page lay out;
- *tests/*
  a skeletal compatibility directory since LXR tests are now separate;

  Note:
  Directory *custom.d/* will be created later during the configuration step; this is where you store your customised files (configuration, templates, …).

scripts: *diff*, *genxref*, *ident*, *search*, *showconfig* and *source* implementing the various operating modes

and a web crawling **security file** *robots.txt*.

You can now run an optional test with an installed script. It will check *Perl* version and the existence of various tools. Of course, this test errors out because you have presently no valid LXR configuration. But you can verify that your environment allows you to proceed without trouble.

```
$ ./genxref --checkonly
ERROR: could not open configuration file lxr.conf
[  OK  ]     Perl     version ... 5.14.2
Parameter 'ectagsbin' not defined - trying to find ctags
ctags found at /usr/bin/ctags
[  OK  ]     ctags    version ... 5.8
Parameter 'glimpsebin' not defined - trying to find glimpse
glimpse found at /usr/local/bin/glimpse
Checked:    glimpse   version ... 4.18.5
Parameter 'glimpseindex' not defined - trying to find glimpseindex
glimpseindex found at /usr/local/bin/glimpseindexg
Checked: glimpseindex version ... 4.18.5
Parameter 'swishbin' not defined - trying to find swish-e
swish-e not found, `command -v swish-e` returned a null string
genxref stopped without indexing by --checkonly option
```

The main items to care for have been highlighted in the above computer output.

- ***Perl*** version: if `FAILED` is displayed, install a newer interpreter (at least 5.10);
- ***ctags*** version: if `FAILED` is displayed, install a newer one (at least 5);
- `'ectagsbin'`, `'glimpsebin'`, `'glimpseindex'`, `'swishbin'`: take note of the locations for step 1.5 Edit the LXR configuration file below.

  It is normal that one of `'glimpsebin'` *and* `'swishbin'` *gives a* `not found` *status because LXR needs only one free-text search engine, but having both installed on a computer does no harm.*

1.3. *Configure your installation*

Configuration is done with the *configure-lxr.pl* interactive script which builds configuration file *lxr.conf* from your answers. This file may later be tuned manually. References to chapter 5 Master configuration file lxr.conf are given to help associate a question to the relevant parameters.

Acceptable answers for “closed” questions are shown inside square brackets. Suggested default is shown in upper case (answers are case-insensitive). You only need to type enough characters to make the answer unambiguous. If your answer is not acceptable, the question is asked again. Just typing the “return” key is equivalent to the default answer.

For “open” questions, there is no limiting list in square brackets, however a reminder for the expected type of answer may be shown between parentheses. When an answer is mandatory, just typing the “return” key is not allowed and the question is asked again.

1.3.a. Simple configuration

This section covers the installation of a **single source-tree** *located in a* **“plain” directory** *(not in a VCS repository).*

CAVEAT!

Configuration for the *Linux kernel*, even if this is the only target source-tree, is considered advanced configuration and has a devoted section.

Our example source tree is named */home/source/tree*. This directory contains several versions of the project as:

- */home/source/tree/v1*
- */home/source/tree/v2*
- */home/source/tree/v3*

where the files composing each version are stored.

Launch script *configure-lxr.pl*. Until you are familiar with *configure-lxr.pl* behaviour, option `-vv` (or `--verbose`) is highly recommended. It prints informative text to help you choose among the numerous possible options.

```
$ ./scripts/configure-lxr.pl --verbose
*** LXR configurator (version: 1.13) ***

LXR root directory is /where/you/installed/LXR
Configuration will be stored in custom.d/
```

Directory *custom.d/* is created if it did not exist.

```
Configure for single/multiple trees? [S/m] >
Do you intend to add other trees later? [yes/NO] >
```

Just hit “return” to select the default `S` (single) and `NO` answer.

```
*** LXR web server configuration ***

LXR can be configured as the default server (the only service in your
computer), a section of this default server or an independent server
(with its own host name).
Refer to the User's Manual for a description of the differences.
Web server type? [1.DEFAULT
/2.section in default
/3.indepedent
/4.section in indepedent
] > 2
```

This question and the following are intended to define the web server configuration. The proposed choices covers two main categories:

- LXR in the default server,
- LXR in an independent site with its own host name (different from `//localhost`),

with two variants in each category:

- the server is completely dedicated to LXR (choices 1 and 3),
- other information is presented besides LXR (choices 2 and 4).

Choose 1 or 2 for initial simple installation.

As far as LXR is concerned, choices 3 and 4 are not more difficult, but they involve more sophisticated configuration of DNS and web servers. Anyway, switching from 1-2 to 3-4 can be done manually (editing lxr.conf *and xxx*-lxrserver.conf*) without running again the configurator for the indexed trees.*

In this guide, we choose case 2, LXR as a section of a wider site.

```
The computer hosting the server is described by an URL. The form is
scheme://host_name:port
  where:
   - scheme is either http or https (http: can be omitted),
   - host_name can be given as an IP address such as 123.45.67.89
               or a domain name like localhost or lxr.url.example,
   - port may be omitted if standard for the scheme.
The following question asks for a primary URL. Later, you'll have
the opportunity to give aliases to this primary URL.
--- Host name or IP? [//localhost] >
```

Accept the default choice. Later, you can define aliases like `//127.0.0.1` or `//mycomputer.mydomain`.

CAUTION!

The host name or IP MUST look like an URL, *i.e.* the host name or IP must be preceded by `//`. The question is asked again if your entry does not follow the advertised form.

```
URL section name for LXR in your server? [/lxr] >
```

This is the name of the part of the server dedicated to LXR. It corresponds to a directory in

`DocumentRoot` and to the initial URL path.

Notes:

LXR is composed of *scripts* and some sites may impose restrictions on the possible locations of such scripts. For instance, scripts may only be launched from *cgi-bin/* in the server directory. In this case, answer with the allowed location like `/cgi-bin`.

As mentioned above, this section is the name of a subdirectory in the `DocumentRoot` hierarchy and can be as long as needed, like `/level1/dir2/sect3/terminal4`.

Always start the section name with a slash (`/`).

The default choice is adequate in the majority of cases.

```
*** LXR database configuration ***

The choice of the database engine can make a difference in indexing
performance, but resource consumption is also an important factor.
  * For a small personal project, try SQLite which do not
    need a server and is free from configuration burden.
  * For medium to large projects, choice is between MySQL,
    PostgreSQL and Oracle.
    Oracle is not a free software, its interface has not been
    tested for a long time.
  * PostgreSQL databases are smaller than MySQL's and performance
    is roughly equivalent.
  * MySQL is at its best with large-sized projects
    (such as kernel cross-referencing) where it is fastest at the cost
    of bigger databases.
  * Take also in consideration the number of connected users.
Database engine? [MYSQL/oracle/postgres/sqlite] >
```

Choose the database available on your computer. If you have not yet installed a database server, you might consider ***SQLite*** which is a sensible choice for a test since it has very few dependencies (if any) and does not demand lots of resources. But do not use it on a big project.

Depending on auto-detection of the free-text search engine *(see 5.4.a Auxiliary tools)*:

```
--- Directory for glimpse databases? > /home/myself/glimpse_databases
--- Directory for swish-e databases? > /home/myself/swish_databases
```

Make sure that directory */home/myself/glimpse_databases* or */home/myself/swish_databases* has *world-read permission* since it will be accessed from the web server.

In case you installed the free-text search engine in a private location (*i.e.* the binary executable cannot be detected by command *command -v*), you must manually tell the path to it:

```
ERROR: neither glimpse nor swish-e found in $PATH!
Is your source tree stored in a VCS repository? [yes/NO] >
```

Answer `NO` since the source tree is stored in plain files.

```
Does one of them exist in a non standard directory? [YES/no] >
```

Tip:

If you answer `NO` *here, free-text search is disabled. This is a way to try LXR without the trouble of installing a free-text search engine.*

```
--- Which is it? [GLIMPSE/swish-e] >
```

For ***Glimpse***:

```
--- Location? (e.g. /usr/share/glimpse-dir/glimpse) >
/home/myself/glimpse/glimpse
--- Location of indexer? (e.g. /usr/share/glimpse-dir/glimpseindex) >
/home/myself/glimpse/glimpseindex
```

For ***Swish-e***:

```
--- Location? (e.g. /usr/share/swish-dir/swish-e) >
/home/myself/swish/swish-e
```

You are now back to the question about directory for the free-text search engine.

Note:

When both ***Glimpse*** and ***Swish-e*** are automatically found, the script opts for ***Glimpse***. If you prefer ***Swish-e***, edit the resulting *custom.d/lxr.conf* file once configuration is complete.

Script continues with:

```
templates directory templates/ now protected read-only
```

IMPORTANT:

This message means you are prevented from making direct changes inside *templates/*. You must first copy any file to *custom.d/* and reset its access permissions to *read/write* before editing it.

```
Is your Apache version 2.4 or higher? [YES/no] > n
```

Apache access controls changed between 2.2 and 2.4, causing incompatibilities in some directives. Answer according to your *Apache* version. If you install another web server, like *lighttpd*, any answer is acceptable.

```
file .htaccess written into LXR root directory
file apache2-require.pl written into configuration directory
file apache-lxrserver.conf written into configuration directory
file lighttpd-lxrserver.conf written into configuration directory
```

Important files are customised, based on your previous answers, and copied to their final destinations. *apache-lxrserver.conf* will need to be copied into the *Apache* configuration directory (often */etc/httpd/conf.d/*).

Now, the real configuration may begin:

```
*** LXR master configuration file setup ***
    Global section part
*** Configuring auxiliary tool paths
*** Configuring host name or IP (as http://...)
*** Host name previously defined as //localhost
--- Alias name ? (hit return to stop) >
```

See 5.4.b Computer DNS names.

For this first trial, configure for a local server without alias names.

When you get used to LXR configuration, this is where you can add other host names for your server, like `//127.0.0.1`, `https://secured.access.example`, *etc*.

CAUTION!

The alias name must look like an URL, but this is not checked here.

Chapter 7 Web server configuration is fully devoted to web server configurations from the simplest (as here) to the most sophisticated.

```
*** Configuring HTML parameters
*** 'Buttons-and-menus' interface is recommended for the kernel
    to avoid screen cluttering.
--- Use 'buttons-and-menus' instead of 'link' interface? [YES/no] > NO
*** Configuring file subsection
*** Configuring "common factors"
*** Marking tree section

*** LXR master configuration file setup ***
    Tree section part
    SQL script for database initialisation

*** Configuring LXR server parameters
*** The virtual root is the fixed URL part after the hostname.
*** You previously defined the virtual root as /lxr
```

Host name and virtual root define how you reference the LXR server from within your browser *(see 5.5.a Server configuration)*. In this example, the URL is `http://localhost/lxr`.

```
--- Caption in page header? (e.g. Project XYZZY displayed by LXR) >
First example
Do you need a specific encoding for this tree ? [yes/NO] >
```

What you define as a caption is printed in a header area of every page displayed by LXR *(see 5.5.a Server configuration)*. Pages are emitted with UTF-8 encoding by default.

```
*** Describing tree location
How is your tree stored? [FILES/cvs/git/svn/hg/bk] >
```

The default answer is for “plain” files and directories. The other choices are for VCS's.

```
*** A source directory contains one sub-directory for every version
--- Source directory? (e.g. /home/myself/project-tree) >
/home/source/tree
```

Name here the directory containing the source-tree *(see 5.5.b.1 Plain files)*.

```
Name to display for the path root? (e.g. Project or $v for version) [$v]
>
```

Default answer $v is replaced by the version name in the file path displayed below the caption. Otherwise, the text is a fixed substitute for the file path root (*i.e.* for */home/source/tree/*version/). *See 5.5.c Other parameters*.

```
*** Enumerating versions
Label for version selection menu?  [Version] >
*** Versions can be explicitly enumerated, be read from a file or
computed by a function. The latter case is recommended for VCS-stored
trees.
Version enumeration method? [LIST/file/function] >
--- Version name? (hit return to stop) > v1
--- Version name? (hit return to stop) > v2
--- Version name? (hit return to stop) > v3
--- Version name? (hit return to stop) >
```

The explicit list of versions is built from the successive answers *(see 5.5.d Version selection)*. To exit from the loop, give an empty answer.

Note:

Since at least one version is mandatory, hitting the return key the first time does not exit the loop. Version name is requested until one is given.

```
*** By default, first version in list is displayed. You may also indicate
a prefered version.
--- Default displayed version is first in 'range'? [YES/no] > n
--- Default version name? > v3
```

A non-default answer of `no` has been given to be able to choose the version displayed at start-up (`v3` instead of `v1`, the first of the list).

Tip:

If you want to ease future version additions and always have the latest version displayed first, list your versions in reverse order: latest first, then older and older. Later, add the new version at the head of the list in file *lxr.conf* and nothing else needs to be changed.

```
*** Setting directory lists
*** Some directories may contain non-public project data (binaries,
compilers caches, SCM control data, ...). They can be hidden from LXR.
--- Directory to ignore, e.g. CVSROOT or .git? (hit return to stop) >
```

Ignore this item a first-time trial.

See 5.5.e Exclude directories (Subdirectory) for an explanation and enter the directory names, one per line.

```
*** If your source code uses "include" statements (#include,
require, ...) LXR needs hints to resolve the destination file.
--- Include directory, e.g. /include? (hit return to stop) >
```

Ignore this item a first-time trial.

The *include directory* list is useful to create the hyperlinks to the included files (mentioned on `#include` statements in C or equivalent in other languages) if they are not located in the same directory as the including file. All paths are relative to the source root directory, but you must write them with an initial separator. See 5.5.g Include directories (Subdirectory) for an explanation and enter the directory names, one per line.

```
*** Configuring data storage
--- Database name? > tree
--- DB user name? [lxr] >
--- DB password? [lxrpw] >
--- DB table prefix? [lxr_] >
```

It is suggested to give the database the same name as the source-tree. The user name and password will be needed later for database initialisation. Take note of them. *See 5.5.h Auxiliary data storage.*

```
configuration saved in custom.d/lxr.conf
DB initialisation sript is custom.d/initdb.sh
```

You are done. Three files have been created: configuration file *lxr.conf*, shell script *initdb.sh* and internal technical file *lxr.ctxt* (do not tamper with its content for reliable *configure-lxr.pl* operation).

Skip now to 1.4 Create a database.

1.3.b. Multiple-trees configuration

This section covers the installation of **several source-trees** *located in* **"plain" directories** *(not in a VCS repository). It also applies to the case of a single source tree if you intend to add others later.*

You can configure as many trees as needed. There is no other limit than your patience and this task can be split into several sessions (see 1.3.c Adding a new tree to a previous configuration).

Launch script *configure-lxr.pl*. Until you are familiar with *configure-lxr.pl* behaviour, option `-vv` (or `--verbose`) is highly recommended but all output is not shown here.

```
$ ./scripts/configure-lxr.pl --verbose
...
Configure for single/multiple trees? [S/m] > m
```

Answer `m` (multiple).

If you inadvertently pressed "return", you can recover with the next question:

```
Configure for single/multiple trees? [S/m] >
Do you intend to add other trees later? [yes/NO] > y
NOTE: installation switched to multiple mode
      but describe just a single tree.
```

Proceed with the web server configuration.

```
*** LXR web server configuration ***
```

```
LXR can be configured as the default server (the only service in your
computer), a section of this default server or an independent server
(with its own host name).
Refer to the User's Manual for a description of the differences.
Web server type? [1.DEFAULT
/2.section in default
/3.indepedent
/4.section in indepedent
] > 2
```

We choose again case 2, LXR as a section of a wider site.

```
The computer hosting the server is described by an URL. The form is
scheme://host_name:port
  where:
   - scheme is either http or https (http: can be omitted),
   - host_name can be given as an IP address such as 123.45.67.89
               or a domain name like localhost or lxr.url.example,
   - port may be omitted if standard for the scheme.
The following question asks for a primary URL. Later, you'll have
the opportunity to give aliases to this primary URL.
--- Host name or IP? [//localhost] >
URL section name for LXR in your server? [/lxr] >
```

See the previous section for notes and advices.

```
The built-in method to manage several trees with a single instance of LXR
is to include a designation of the tree in the URL at the end of the
section name.
This sequence after host name is called "virtual root".
Supposing one of your trees is to be referred as "my-tree", an URL to
list the content of the default version directory would presently be:
     //localhost/lxr/my-tree/source
with virtual root equal to /lxr/my-tree

Use built-in multiple trees management with tree designation at end of
virtual root? [YES/no] >
```

Managing several trees with a single instance of LXR requires some “URL black magic” which needs expertise in web server configuration. To work around this skill, a generic policy is built in LXR. A tree identification is coded at the end of the URL path to LXR service (`//localhost/lxr` in the present case) and before the script name. This is adequate for most people.

If you answer `NO`, you must give the full virtual root for every tree. And, **MOST IMPORTANT**, after configuration, you must write your routing policy in the xxx-*lxrserver.conf*.

If you answer `YES` (or accept the default answer), the built-in policy is used and the definition of each tree virtual root is simplified to giving the tree identification.

Accept the default answer and you proceed with database configuration.

```
Database engine? [MYSQL/oracle/postgres/sqlite] >
```

Choose the database available on your computer. If you have not yet installed a database server, you might consider ***SQLite*** which is a sensible choice for a test since it has very few dependencies (if any) and does not demand lots of resources. But do not use it on a big project.

```
The safest option is to create one database per tree.
You can however create a single database for all your trees with a
specific set of tables for each tree (though this is not recommended).
How do you setup the databases? [PER TREE/global] >
```

Unless you are restricted by site-local rules (such as having only one database per user), use the default database policy. With *Oracle*, there is only one global database; you have no choice. The databases for each tree are distinguished by a unique table prefix.

```
All databases can be accessed with the same username and can also be
described under the same names.
Will you share database characteristics? [YES/no] >
```

The default answer is recommended but there is strictly no performance advantage in either case.

```
Will you use the same username and password for all DBs? [YES/no] >
--- DB user name? [lxr] >
--- DB password ? [lxrpw] >
Will you give the same prefix to all tables? [YES/no] >
--- Common table prefix? [lxr_] >
```

You might change the default password `lxrpw` for something else. If you prefer to access the databases without password, you need to edit the resulting *lxr.conf* file since there is no way to enter an “empty” answer. For that, locate the following line in the *“Common factor” subsection* before the *Tree configuration sections*:

```
, 'dbpass'       => 'lxrpw'
```

to change it to:

```
, 'dbpass'       => ''
```

```
--- Directory for glimpse databases? /home/myself/glimpse_databases
--- Directory for swish-e databases? /home/myself/swish_databases
```

From there on, refer to the previous section to finish global parameters set-up (up to `Alias name` question).

The first tree is configured between `virtual root` line and database characteristics.

```
*** The virtual root is the fixed URL part after the hostname.
*** The tree needs to be uniquely identified as e.g. /lxr/the_tree
--- Tree designation for URL? (e.g. the_tree) > tree
```

We choose here to designate the tree with the same name as its directory (any designation will do as long as it is unique among the trees, but it is good practice to use a designation related to the true tree name to facilitate bug resolution).

This identification is added at the end of the current URL path to make the effective virtual root.

```
--- Caption in page header? (e.g. Project XYZZY displayed by LXR) >
First tree in multiple example
Do you want a speed switch button for this tree ? [YES/no] >
```

The *speed switch* button is an hyperlink (a so-called button) added in the page header area allowing to “jump” to another tree without the need to write the URL. If you answer YES, you are asked to give a (short) legend to this “button”:

```
--- Short title for button? (e.g. XYZZY) > Project
```

Note:

Do not exceed approximately 10 characters for this short title since space is limited in the header area, notably if you have many trees. *See 5.5.c Other parameters* 'shortcaption'.

When finished with a source tree, you can configure another one:

```
*** Configure another tree? [YES/no] > n
```

You are done. Three files have been created: configuration file *lxr.conf*, shell script *initdb.sh* and internal technical file *lxr.ctxt* (do not tamper with its content for reliable *configure-lxr.pl* operation).

Skip now to 1.4 Create a database unless you need a break before continuing with other trees in the following section.

I.3.c. Adding a new tree to a previous configuration

The new description will be added at the end of configuration file *lxr.conf* and database initialisation script *initdb.sh*.

CAUTION!

initdb.sh contains commands and directives that DELETE users and databases. Consequently, if you have already created the databases by a previous execution of this script, you MUST NOT run it again. To prevent data loss, delete file *initdb.sh* before launching *configure-lxr.pl*. If the databases have not been yet created, keep *initdb.sh*.

Run *configure-lxr.pl* with option --add:

```
$ ./scripts/configure-lxr.pl --add --verbose
...
== ADD MODE ==
Initial context custom.d/lxr.ctxt is reloaded
```

The script first retrieves important parameters used in the initial configuration to avoid inconsistencies. This is why you must not tamper with **.ctxt* files.

```
Your DB engine was: MySQL
Advanced users can configure different DB engines for different trees.
This is not recommended for average users.
Use previous DB engine? [YES/no] >
```

Unless you have a real good reason to change the database engine (such as comparing performances of database servers on the same source tree), keep the same engine. This item is mainly for developers.

Then you continue with the usual questions and answers.

```
*** LXR master configuration file setup ***
    Tree section part
    SQL script for database initialisation

*** Configuring LXR server parameters
*** The virtual root is the fixed URL part after the hostname.
*** The tree needs to be uniquely identified as e.g. /lxr/the_tree
--- Tree designation for URL? (e.g. the_tree) >
```

Note that the initial part of the virtual root is recovered from the context file.

When you are done with additions, the new tree descriptions have been added to the end of configuration file *lxr.conf* and new commands for the databases to the end of script *initdb.sh*.

Make a break before new additions or see 1.4 Create a database.

Limitations:

- The script does not attempt to check that all virtual roots are unique.
- It does not check that if a database user name is used for different trees its associated password is the same for the same database.
- It does not check that the combination database name/table prefix is unique.

  An individual tree-database must have an unique name.

  In a database shared by several trees, table prefixes must be different.
- Database users are blindly created in the database initialisation script *initdb.sh* when they are new in the current session. No attempt is made to collect already existing users in the initialisation script and detect collisions.

  This requires manual editing to remove duplications in *initdb.sh***.**

1.3.d. Configuration for Linux kernel

The Linux kernel tree can be described like any other source tree. However, to fully benefit of the automatic “architecture” and “include” navigation feature, you need to thoroughly understand the directory structure.

This knowledge has been transcribed in a specific configuration driver. It is valid only for a “plain files” tree, not for VCS repositories.

To configure only for the kernel, run (omitting `--verbose` option if you are now familiar with the script output):

```
$ ./scripts/configure-lxr.pl lxrkernel.conf
```

Script output will be in *custom.d/lxrkernel.conf* (and *custom.d/lxrkernel.ctxt*) and *initdb.sh*.

To add the kernel to an existing configuration, run:

```
$ ./scripts/configure-lxr.pl --add --conf-out=lxr.conf lxrkernel.conf
```

Script output is added to *custom.d/lxr.conf* and *initdb.sh* (previous context information being retrieved from *custom.d/lxr.ctxt*).

The differences are in version enumeration and directory relationships.

Versions are identified by another script along with architectures (the Linux word for computer family upon which the kernel has been ported) and platforms (subfamilies).

Rules for include directory relationships are computed so that clicking on a `#include` statement warps you to the correct file in the selected architecture directory.

Altogether, this builds a set of buttons in the page header area to select the desired version and architecture for browsing.

Note:

The *buttons-and-menus* interface (see 2.2 General aspect of LXR pages) is recommended for kernel browsing. This interface style can be selected either by a choice during configuration or by editing the configuration file.

Collecting version information is done with script *kernel-vars-grab.sh*. With the kernel source tree in */home/source/kernel*, run as:

```
$ ./scripts/kernel-vars-grab.sh --erase /home/source/kernel
```

It creates in *custom.d/* a set of files whose names end with `_list.txt`. These files contain lists of versions, architectures, platforms, … Without `--erase` option, results are *added* to the lists which allows to gather cumulative lists across kernel versions while keeping only the last one, thus sparing disk space.

You can build independent lists with option --`suffix=_other.txt` for example. But you will need to manually update the resulting *lxrkernel.conf* or *lxr.conf* file.

1.3.e. Configuration for VCS's

For general information, see 10 Using LXR with SCMs.

Tree storage is selected by:

```
How is your tree stored? [FILES/cvs/git/svn/hg/bk] > c
```

1.3.e.1. CVS

Answer `c` for ***CVS*** *(see 5.5.b.2 CVS repository and 10.2 CVS).*

```
*** A CVS repository is a directory containing ,v files
--- CVS repository? (e.g. /home/myself/project-CVS) >
```

Answer with the name of the directory containing the *,v* files, like */home/myself/project-cvs*.

```
Name to display for the path root? (e.g. Project or $v for version) [$v]
>
*** Enumerating versions
Label for version selection menu? [Version] >
*** Versions can be explicitly enumerated, be read from a file or
computed by a function. The latter case is recommended for VCS-stored
trees.
Version enumeration method? [LIST/file/function] > fu
```

Use of a function gives an up-to-date list of versions on a “live” repository.

```
*** This template contains generic nearly-all-purpose functions.
*** Since designing such a function is not a trivial exercise,
*** you'd better choose an available one. You can later refine it
*** to fit your needs.
--- Generic or custom function? [files/cvs/git/svn/hg/custom] > cvs
*** With a function, you MUST indicate a default version.
--- Default version name? [head] >
```

Select the `cvs` function and accept the default `head` name.

1.3.e.2. GIT

Answer `G` for ***GIT*** *(see 5.5.b.3 GIT repository and 10.3 Git).*

```
*** A Git repository is a directory containing objects, refs, index, ...
subdirectories. It is usually named .git in some user directory and is
thus not visible.
--- Git repository? (e.g. /home/myself/project-git/.git) > /…/.git
```

Answer with the name of the *.git* directory, like */home/myself/project/.git*.

```
--- display revision-ids? [YES/no] >
--- display revision author name? [YES/no] >
Name to display for the path root? (e.g. Project or $v for version) [$v]
>
*** Enumerating versions
Label for version selection menu?  [Version] >
*** Versions can be explicitly enumerated, be read from a file or
computed by a function. The latter case is recommended for VCS-stored
trees.
Version enumeration method? [LIST/file/function] > fu
```

The default for the display variants are appropriate for nearly everybody. Use of a function gives an up-to-date list of versions on a “live” repository.

```
*** This template contains generic nearly-all-purpose functions.
*** Since designing such a function is not a trivial exercise,
*** you'd better choose an available one. You can later refine it
*** to fit your needs.
--- Generic or custom function? [files/cvs/git/svn/hg/custom] > git
*** With a function, you MUST indicate a default version.
--- Default version name? [HEAD] >
```

Select the `git` function and accept the default `HEAD` name.

1.3.e.3. Subversion

Answer `S` for ***Subversion*** *(see 5.5.b.4 Subversion repository and 10.4 Subversion).*

```
*** A Subversion repository is a directory containing a database for the
source-tree. The present backend implementation in LXR limits access to
local repositories.
--- Subversion repository? (e.g. /home/myself/project-svn) >
```

Answer with the name of the *Subversion* directory, like */home/myself/project-svn*.

```
--- display revision-ids? [YES/no] >
--- display revision author name? [YES/no] >
Name to display for the path root? (e.g. Project or $v for version) [$v]
>
*** Enumerating versions
Label for version selection menu? [Version] >
*** Versions can be explicitly enumerated, be read from a file or
computed by a function. The latter case is recommended for VCS-stored
trees.
Version enumeration method? [LIST/file/function] > fu
```

Use of a function gives an up-to-date list of versions on a “live” repository.

```
*** This template contains generic nearly-all-purpose functions.
*** Since designing such a function is not a trivial exercise,
*** you'd better choose an available one. You can later refine it
*** to fit your needs.
--- Generic or custom function? [files/cvs/git/svn/hg/custom] > svn
*** With a function, you MUST indicate a default version.
--- Default version name? [head] >
```

Select the `svn` function and accept the default `head` name.

1.3.e.4. Mercurial

CAUTION!

This support is still experimental and exhibits poor performance for directory retrieval.

Answer `H` for ***Mercurial*** *(see 5.5.b.5 Mercurial repository and 10.5 Mercurial).*

```
*** A Mercurial repository is a directory containing a database for the
source-tree. The present backend implementation in LXR limits access to
local repositories.
--- Subversion repository? (e.g. /home/myself/project-hg) >
```

Answer with the name of the directory containing *.hg/*, like */home/myself/project-hg*.

```
--- display revision-ids? [YES/no] >
--- display revision author name? [YES/no] >
Name to display for the path root? (e.g. Project or $v for version) [$v]
>
*** Enumerating versions
Label for version selection menu? [Version] >
*** Versions can be explicitly enumerated, be read from a file or
computed by a function. The latter case is recommended for VCS-stored
trees.
Version enumeration method? [LIST/file/function] > fu
```

Use of a function gives an up-to-date list of versions on a "live" repository.

```
*** This template contains generic nearly-all-purpose functions.
*** Since designing such a function is not a trivial exercise,
*** you'd better choose an available one. You can later refine it
*** to fit your needs.
--- Generic or custom function? [files/cvs/git/svn/hg/custom] > hg
*** With a function, you MUST indicate a default version.
--- Default version name? [tip] >
```

Select the `hg` function and accept the default `tip` name.

1.3.e.5. BitKeeper

Answer `B` for ***BitKeeper*** but the interface has not been tested for a long time. If you opt for it, please share your experience. *See 5.5.b.6 BitKeeper repository and 10.6 BitKeeper.*

1.3.f. Advanced use of configurator

```
$ configure-lxr.pl    [options]    [template]
```

Options are:

`-h` or `--help`

Print a summary and quit

`--version`

Print version information and quit

`-vv` or `--verbose`

Print comments and progress information

Note:

`-vv` is strictly equivalent to `--verbose`, while `-v` (a single `v`) prints fewer messages. Technically, the former prints `@LOG` and `@MSG` statement argument from the configuration template while the latter only prints `@LOG` statement argument.

`--root-dir=`*LXR_root_directory*

To be used if current working directory is not equal to *LXR root directory*

`--tmpl-dir=`*templates_directory*

For alternate templates directory (default is *templates/* in LXR root directory)

`--conf-dir=`*output_configuration_directory*

Alternate location for output file (default is *custom.d/* in LXR root directory)

`--conf-out=`*filename*

Name, without directory, of output configuration file (default equal to template name with extension replaced by *.conf*); stored in `--conf-dir` directory

`--script-out=`*filename*

Name, without directory, of output script (default is *initdb.sh*); stored in `--conf-dir` directory

template is the name, without directory, of a model in `--tmpl-dir` directory. It contains a skeletal *lxr.conf* configuration file with directives describing interaction with the user. Interactive data is used to customise the template.

1.4. Create a database

The previous step created an *initdb.sh* script (unless you changed its name with `--script-out` option). Its contains all commands to create the databases and users.

```
$ ./custom.d/initdb.sh
```

Note:

When done, either delete this script so that future additions will not cause deletion of existing trees or put it aside with a different name. In case of need, it can easily be regenerated from the content of *lxr.conf*.

1.4.a. MySQL

User creation commands[6] are issued under the default master account `root`. If it bears another name on your installation, edit the script to change all occurrences of `-u root` to the local master account or, if you are not allowed to use it, any local user with right to create databases and users.

The password for the master account is asked at most twice[7] for every tree with:

```
Enter password :
```

User creation is optional (depends on global or override choice). It may fail with `ERROR 1396` if user already exists. This precludes changing password this way. To do that, drop user manually before launching script.

SECURITY WARNING!

Users are created with **ALL** privileges (except CREATE), meaning they can access and modify any other database in your installation. If this is an issue, manually drop the privileges.

[6] Database initialisation templates can be configured to issue ALL commands under the master *MySQL* account. But this implies you must manually enter its password twice for every user and twice for every database!

[7] Calling `mysql` command separately is a design choice to avoid a failing step would cancel the following ones.

In particular, you can remove all write permissions after *genxref* if you do not intend to run it again.

Database creation is also optional (depends also on global or per-tree policy).

Tables are always created for a tree.

You can use ***phpMyAdmin*** *to check the result if you have it installed.*

1.4.b. *PostgreSQL*

User creation commands[8] are issued under the default master account `postgres`. If it bears another name on your installation, edit the script to change all occurrences of `-U postgres` to the local master account or, if you are not allowed to use it, any local user with right to create databases and users.

Since there is no way to hand over the password to command *createuser*, you must enter it manually after the prompt:

```
*** PostgreSQL - Creating global/tree user xyz
Enter password for new role:
Enter it again:
```

Read the line above the question to know which user (role in *PostgreSQL* terminology) is concerned.

PostgreSQL issues many warnings and errors. Ignore those related to non-existent users, databases, tables, functions … The script plays it safe trying to delete an object before creating it.

You can use ***phpPgAdmin*** *to check the result if you have it installed.*

1.4.c. *SQLite*

There is no notion of user. The file containing the database is created when you initialise the database. You may eventually need to create a directory for it before running the script. Make sure it *world read AND write enabled.*

You can use ***Sqliteman*** *to check the result if you have it installed.*

1.4.d. *Oracle*

Script execution could not be checked on a real case because the software is proprietary. Script generation was adapted from the original sources and evolution for the other database engines. If you opt for this engine, please report your experience.

[8] Database initialisation templates can be configured to issue ALL commands under the master *PostgreSQL* account. Usually, *PostgreSQL* is configured in *trusted* mode for local access and you do not need to provide the master password.

1.5. *Edit the LXR configuration file*

Your customised *lxr.conf* file (or whatever name you gave with `--conf-out=`) is stored in the *custom.d/* directory. There is usually no need to edit this file apart from tuning it. This can be done any time once you have gained some practice with LXR.

The main parameters to tune are enumerated below. A full discussion on the parameters is found in chapter 5 Master configuration file lxr.conf.

The following titles are the same as those for the sections in the file.

1.5.a. Global configuration section, HTML subsection

Important!

All file paths in this subsection are anchored at the LXR root directory. You cannot use OS absolute paths.

You can choose the interface style (between *link* and *buttons-and-menus*). Select the right template with **one** of the following:

```
, 'htmlhead'  => 'templates/html-head.html'
, 'htmlhead'  => 'templates/html-head-btn.html'
```

If you change free-text search engine after configuring (*e.g.* adding one afterwards[9]), make `'htmlsearch'` point to the right template with **one** of the following:

```
, 'htmlsearch'  => 'templates/html-search-glimpse.html'
, 'htmlsearch'  => 'templates/html-search-swish.html'
```

1.5.b. Global configuration section, file management subsection

This is where you specify files to ignore and custom icons for directory listing *(see 5.4.d File management)*.

1.5.c. Tree configuration section, version selection subsection

In case you chose `custom` function for computed version list, you MUST define your own function for `'range'` because the generated skeleton function returns an empty list. Consequently, without modification, the version selection feature is disabled and LXR no longer understands the version concept.

1.5.d. Tree configuration section, subdirectory subsection

You list here which directories or files to ignore when indexing and browsing the tree. The tree rule takes precedence over the one defined in the global file management subsection, Consequently, do

9 If no search engine is detected during configuration, this feature is disabled and `'htmlsearch'` defaults to `html-search-glimpse.html`.

not forget to copy first this global rule (so that its general exclusions are kept) before extending or tuning it.

```
, 'ignorefiles'  => regexp
, 'ignoredirs'   => list_of_directories
```

Parameters `'filterdirs'` and `'filterfiles'` may also be specified to exclude designated files (instead of *families* of files) but their use should be avoided as much as possible for performance reason.

1.6. Copy configuration

Copy the configuration file where LXR expects to find it:

```
$ cp custom.d/lxr.conf .
```

Do not forget the final dot.

1.7. Generate index

It is now time to generate the index. This is done using the program *genxref*. *genxref* takes two arguments `--url=` and `--version=`.

- `url` is the URL where the LXR web front-end appears.

  It must be identical to the concatenation of `'host_names'` and `'virtroot'` (or, for compatibility, to `'baseurl'` or one of the `'baseurl_aliases'`).

  You *can drop* `http:` *from the URL; however the argument MUST look like an URL and at least begin with* `//`.
- `version` is the name of a directory in your `'sourceroot'` directory.

  It is identical to one tag in the `'range'` list for parameter variable `'v'` (see 5.5.d Version selection). It is worth noting that one *lxr.conf* file can be used for several different source-trees. Which configuration block to use is selected according to the `url` argument.

You can also use the `--allversions` argument to automatically index all the versions defined in the version variable.

But before running the indexing for real, you have the opportunity to check your installation with argument `--checkonly`:

```
$ ./genxref --url=//localhost/lxr --checkonly
[   OK  ]     Perl     version ... 5.14.2
[   OK  ]     ctags    version ... 5.8
Checked:    glimpse    version ... 4.18.5
Checked: glimpseindex version ... 4.18.5
```

```
Parameter 'swishbin' not defined - trying to find swish-e
swish-e not found, `command -v swish-e` returned a null string
genxref stopped without indexing by --checkonly option
```

If an error is reported, review the previous steps.

You can now roll the indexing.

```
$ ./genxref --url=http://localhost/your_lxr_virtual_root --version=v2
```

Note that *genxref* can be a very slow process the first time it is run, for example on a 4Gb source tree a full run can take several days[10]. However, on future runs it will only index changed files, thus speeding the process.

I.8. Set up the web server

Storing the *LXR root directory* somewhere else than the traditional web server directory (usually */var/www/*) may lead to security warnings or access denials on security enhanced OS's like *SELinux*. The solution to this problem is to label appropriately the *LXR root directory*. Under *SELinux*, to copy the correct label use:

```
$ chcon --reference /var/www/cgi-bin/ -R /LXR/root/directory/
```

I.8.a. Apache server

Configuration traditionally goes into httpd.conf/, *but you are advised to take benefit from the* conf.d/ *include feature and use a separate configuration file for your server. In this way, global Apache configuration and yours are independent.*

Step 1.3 Configure your installation above created an *apache-lxrserver.conf* configuration file. This file handles automatically all known cases (*mod_perl* or bare CGI scripts, *Apache* 1 or 2).

If you configured LXR as an independent server (*i.e.* not the default `//localhost`), *apache-lxrserver.conf* contains a `<VirtualHost>` section where only the primary host name is listed. The primary host name is the one given to answer `--- Host name?` question. You need to manually add the aliases into the `ServerAlias` directive:

```
ServerName     //primary.server.example
ServerAlias    lxr.server.example  lxr.lan  192.168.1.1
```

Note:

The configurator kept `//` in the primary server name, but these characters are not accepted in the aliases.

If you declared a specific port, a `Listen` directive is created. However, you may encounter incompatibility if this port is already listed in another configuration file. In this case, remove the

[10] Note from the editor: as usual, the original author of this assertion omitted to quote the technical environment to permit to predict the time needed on one's configuration (clock frequency, main memory, disk channel speed, …).

offending port. ***Apache*** expects at most one `Listen` directive per port.

When done adjusting configuration, copy the file into the *Apache* directory:

```
$ cp custom.d/apache-lxrserver.conf /etc/httpd/conf.d
```

Note:

You need administrator privileges to access this directory.

If you configured for ***Apache*** 2.2 or earlier and later upgrade to ***Apache*** 2.4 without reinstalling LXR from scratch, uncomment the 2.4-specific lines in *apache-lxrserver.conf* and in *.htaccess*.

I.8.a.1. Apache 1.x specific

You must manually put the *Perl* modules that LXR uses into a directory on your system that will be searched by `mod_perl` when the LXR scripts are executed (typically *site_perl*):

```
$ cp -r lib/*  /usr/lib/perl5/site_perl/
```

Note:

You may need administrator privileges to access this directory.

I.8.a.2. Apache 2.x specific

No other step is necessary.

Note:

Apache 2 can run either as unthreaded or threaded server. This should now be transparent to LXR. However, if you meet difficulties, report to the LXR maintainer with the following information.

You can tell which *Multi-Processing Module* (MPM) is active with the following command:

```
$ apachectl[11] -V
Server version: Apache/2.2.21 (Unix)
Server built:   Sep 13 2011 12:26:57
Server's Module Magic Number: 20051115:30
Server loaded:  APR 1.4.5, APR-Util 1.3.12
Compiled using: APR 1.4.5, APR-Util 1.3.12
Architecture:   64-bit
Server MPM:     Prefork
  threaded:     no
    forked:     yes (variable process count)
Server compiled with....
```

Look at the line beginning with `Server MPM`.

If it reads `Prefork`, you are unthreaded.

If it reads `Worker`, you are threaded.

[11] Depending on your distribution, the command name may be `apache2` or `apache`.

1.8.b. *Lighttpd* server

*Though **lighttpd** configuration has been integrated into the common configuration frame, this is still experimental. Please report any difficulty or work around so that this part can be improved.*

Step 1.3 Configure your installation above created a *lighttpd-lxrserver.conf* configuration file. But it needs to be edited. Paths are based on *Fedora* distribution architecture where ***lighttpd*** configuration is stored in */etc/lighttpd/* (notably modules configuration in *conf.d/*).

Open the file with your favourite editor (*vi*, *emacs*, *Kwrite*, *gedit*, …) and change the relevant values. It is divided in sections for easy navigation.

In “Variable definition”, `var.server_root` is the location of servers directory, `var.conf_dir` is the configuration directory:

```
var.server_root = "/var/www/"
var.conf_dir    = "/etc/lighttpd"
```

In “Load the modules”, the following directive will load a minimal set of ***lighttpd*** modules required by LXR.

```
include "LXR_root_dir/templates/lighttpd/lighttpd-lxr-modules.conf"
```

Should you need a different set, copy *templates/lighttpd/lighttpd-lxr-modules.conf* to *custom.d/* and edit the file. Then change the directive to:

```
include "LXR_root_dir/custom.d/lighttpd-lxr-modules.conf"
```

In “Basic configuration”, you can change the port ***lighttpd*** is listening to if LXR service is answering on a non-standard port. This is normally forwarded by the configurator.

```
server.port = 80
```

Scroll down to the bottom of the file at “LXR section”. `server.document-root` and `cgi.assign` have been set for LXR operation.

```
server.document-root = "LXR_root_dir/"
cgi.assign += ( "/source" => ""
              , "/ident"  => ""
              , "/diff"   => ""
              , "/search" => ""
              , "/showconfig" => ""
              )
```

You may add here the host names you defined in *lxr.conf*'s `'host_names'` as (one line per host name):

```
$HTTP["host"] = "127.0.0.1"     {server.document-root = "LXR_root_dir/"}
$HTTP["host"] = "mycomputer.outside.domain.example.com" {
                                 server.document-root = "LXR_root_dir/"}
$HTTP["host"] = "192.168.12.34:56" {
```

```
                                server.document-root = "LXR_root_dir/"}
```

Save the modified file and make it checked by ***lighttpd***:

```
$ lighttpd -t -f custom.d/lighttpd-lxrserver.conf
```

If you pass the test, you are ready to launch ***lighttpd***. For a private test, run command:

```
$ lighttpd -D -f custom.d/lighttpd-lxrserver.conf
```

Note:

This may require editing configuration file *custom.d/lighttpd-lxrserver.conf* to change value of `var.state_dir` to a user-write enabled directory.

To stop the server, type `ctl-C`.

For system-wide service, use script *lighttpd-init* with command (on a single line – document processor breaks line at dash):

```
$ LIGHTTPD_CONF_PATH="LXR_root_directory/custom.d/lighttpd-
lxrserver.conf" ./scripts/lighttpd-init start
```

Lighttpd will daemonise and run in the background. To stop, type:

```
$ ./scripts/lighttpd-init stop
```

Note:

You may also copy this script in */etc/init.d/* as *lighttpd* and use it with the standard `start`, `stop` and `status` argument:

```
$ cp scripts/lighttpd-init /etc/init.d/lighttpd
$ LIGHTTPD_CONF_PATH="LXR_root_directory/custom.d/lighttpd-
lxrserver.conf" /etc/init.d/lighttpd start
$ /etc/init.d/lighttpd stop
$ /etc/init.d/lighttpd status
```

Integrating LXR service as part of an existing web site is left as an exercise.

Read next section about icons.

1.8.c. Other web servers

Most web servers should be capable of supporting the CGI script versions of LXR - consult the server documentation for information on how to configure this.

If you are using a web server other than *Apache*, you need to provide the following icons (their name in HTML-absolute form):

- */icons/back.gif*
- */icons/folder.gif*
- */icons/c.gif*

- */icons/text.gif*
- */icons/compressed.gif*
- */icons/image2.gif*
- */icons/generic.gif*

LXR works perfectly without these icons; they only have an aesthetic role. Starting with release 0.10 a versatile feature is incorporated to allow the use of any icon of your choice (see 5.4.d File management).

1.9. Run a test

LXR should now work. Fire up your web browser and go to `http://localhost/lxr/source` (or replace `localhost` by any of your `'host_names'`) and you should see the listing for the top of your source tree.

If you don't see the LXR page, first check that you configured correctly your web server. Try to go to `//localhost/lxr/` to display the content of the LXR root directory.

> *To enable directory listing by Apache server, you need to change a line in the* .htaccess *file. It is the first directive in the "Access restrictions" section. Replace the minus sign by a plus:*
>
> ```
> Options +Indexes
> ```
>
> *Directory listing is enabled by default in lighttpd.*

If nothing still appears, check your logs, especially the error and access logs to see what's happening. Problems are often caused by the web server not being able to access the LXR files – make sure all the paths and file permissions are correct.

1.10. Site-specific customisation

LXR has a wealth of parameters giving it flexibility and allowing control over many aspects of its behaviour.

The places to look for customisation are:

- *lxr.conf*: controls most basic settings, including how file names are mapped to languages, tab settings, *etc*. See chapter 5 Master configuration file lxr.conf.
- *templates/html**, *templates/lxr.css*: templates and CSS for the HTML display.

  This is where you can change how the website looks like, *e.g.* by adding site logos, links, or changing colours, fonts, *etc*. See chapters 8 Customising page architecture and 9 Customising LXR appearance.
- *templates/ectags.conf*: advanced configuration for ***ectags*** when parsing various language files -

this is one place to look if you want to add support for another language.

- *lib/LXR/Lang/generic.conf*: configures the generic language support module that handles most of the languages LXR recognises. Configuration here enables support for new languages. See chapter 6 Generic parser configuration file.
- *lib/Local.pm*: various routines that may need to be customised to provide useful file summaries in the directory and file listing views.

Remember the golden rule:

Always copy first any template file to *custom.d/* directory before editing it. Do not forget to change the location of any customised file in *lxr.conf*.

1.11. Troubleshooting

Common errors are listed on the web site at http://lxr.sf.net/en/troubleshooting.shtml. Only the most frequent for a first installation are mentioned here.

`** Fatal: Couldn't open configuration file lxr.conf`

The configuration file *lxr.conf* does not exist, is not in the LXR root directory (it might have been left in the *templates/* or *custom.d/* directory) or cannot be accessed (check the file permissions: it must be *world readable*).

`** Fatal: Can't find config for URL_of_your_LXR_tree`

The value of the `--url` argument (or URL given to your browser) matches none of the `'hostnames'/'virtroot'` combinations. Check the spelling of the URL or *lxr.conf*.

`** Fatal: Can't create index xxx`

This message is usually preceded by others about DB, lock or socket. Check the data base description in *lxr.conf* and your access permissions to the data base.

`** Fatal: Can't find database`

This message comes from the *Perl* DB backend. The likely cause is that the web server URL passed to the browsing scripts does not match any URL in *lxr.conf*. This means the configuration will not be found, and thus the database will not be found.

You see HTTP headers (such as `Content-Type)` appearing in the page or output not being interpreted as HTML

This can be caused by warning messages output before the script sends the right headers to tell the browser that the output is HTML. This can normally be solved by changing the value of `$wwwdebug` to 0 in *Common.pm*. But please report the warning message as a bug at http://sourceforge.net/projects/lxr first! The developers left debugging features in the release version, which should never happen.

1.12. Note on security

The footer of every page generated by LXR exhibits an HTML-compliant logo which is a link to the W3C verification service. If you click on it, you can verify that LXR or the modifications you made to its standard appearance are really compliant.

But, intentional or result of a security breech, within 24 hours, robots will try to index your site. Good job if it is intended to be public. Rather bad if you want to stay in the shadow. To alleviate this, LXR has now a very restrictive *robots.txt* file which asks well-behaved robots to stay away. Adapt it to suit your need or remove it (simply change its name) to allow full access.

2 Using LXR

This chapter covers only every day use of LXR so that you can get the best of LXR. Tuning the installation is dealt with in another chapter.

2.1. Launching LXR

You navigate through the source tree with your favourite web browser. Just fill-in the address bar with one of the `'host_names'` followed by your `'virtroot'` and add ***/source***, you will be transported to the root of the source-tree. From that point, you have access to all LXR functions and all files in the tree.

From the example configuration in the previous chapter:

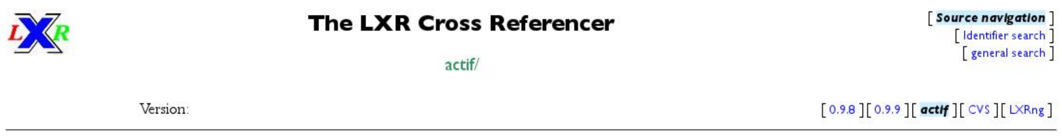
```
http://localhost/lxr/source
```

You can also access directly the identifier or free-text search pages with ***/ident*** or ***/search*** respectively.

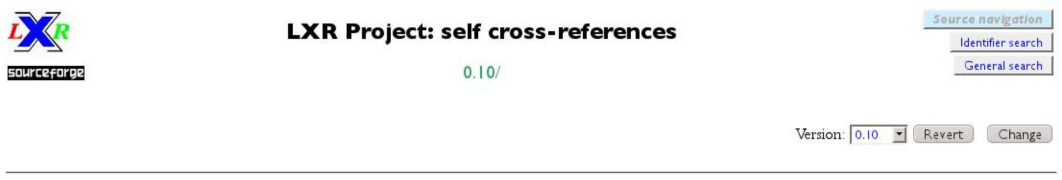
2.2. General aspect of LXR pages

The pages are all displayed with a common layout. They have a header, a body and a footer.

The **header** looks like this:



or like this, with the *buttons-and-menus* interface:



The logo in the upper left is a customisation of the template.

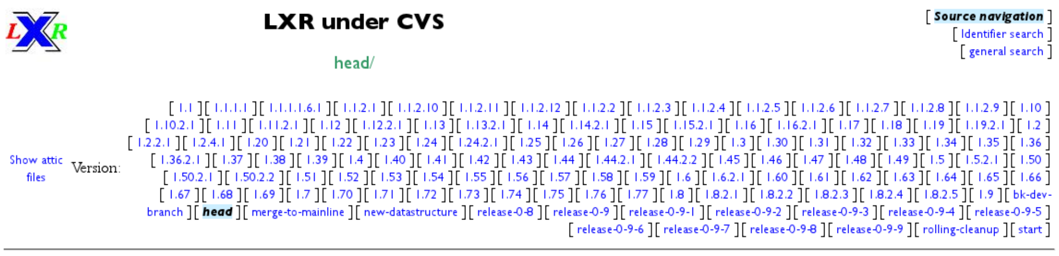
The upper centre displays the content of `'caption'` as a title and the path to the current file or directory from `'sourceroot'` with the prefix given in `'sourcerootname'`. Every component of this path may be individually clicked to transport you directly to this node of the tree.

The upper right contains a set of “buttons” to switch between the different modes. The selected mode is highlighted.

The lower part displays the states of the `'variables'`. In this example, only variable `'v'` with name *Version* is defined. Its possible values are in the lower right, with the current value enhanced. Note that `'sourceroot'` has been configured to reflect the current value of this variable.

Independently of personal taste, when is it preferable to use the *buttons-and-menus* interface or the *link* interface?

The *link* interface opens a new line for every variable. All the values are then displayed, which may necessitate many lines (think of all the versions for the Linux kernel). This in turn leaves little room for the real information you are interested in: the source-tree. The following figure show a typical bad choice of the *link* interface.



LXR under CVS

head/

[Source navigation]
[Identifier search]
[general search]

Show attic files

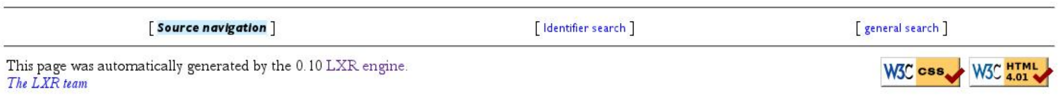
Version: [1.1][1.1.1.1][1.1.1.1.6.1][1.1.2.1][1.1.2.10][1.1.2.11][1.1.2.12][1.1.2.2][1.1.2.3][1.1.2.4][1.1.2.5][1.1.2.6][1.1.2.7][1.1.2.8][1.1.2.9][1.10][1.10.2.1][1.11][1.11.2.1][1.12][1.12.2.1][1.13][1.13.2.1][1.14][1.14.2.1][1.15][1.15.2.1][1.16][1.16.2.1][1.17][1.18][1.19][1.19.2.1][1.2][1.2.2.1][1.2.4.1][1.20][1.21][1.22][1.23][1.24][1.24.2.1][1.25][1.26][1.27][1.28][1.29][1.3][1.30][1.31][1.32][1.33][1.34][1.35][1.36][1.36.2.1][1.37][1.38][1.39][1.4][1.40][1.41][1.42][1.43][1.44][1.44.2.1][1.44.2.2][1.45][1.46][1.47][1.48][1.49][1.5][1.5.2.1][1.50][1.50.2.1][1.50.2.2][1.51][1.52][1.53][1.54][1.55][1.56][1.57][1.58][1.59][1.6][1.6.2.1][1.60][1.61][1.62][1.63][1.64][1.65][1.66][1.67][1.68][1.69][1.7][1.70][1.71][1.72][1.73][1.74][1.75][1.76][1.77][1.8][1.8.2.1][1.8.2.2][1.8.2.3][1.8.2.4][1.8.2.5][1.9][bk-dev-branch][**head**][merge-to-mainline][new-datastructure][release-0-8][release-0-9][release-0-9-1][release-0-9-2][release-0-9-3][release-0-9-4][release-0-9-5][release-0-9-6][release-0-9-7][release-0-9-8][release-0-9-9][rolling-cleanup][start]

When you meet this case, prefer the *buttons-and-menus* interface. The variables will occupy a single line (unless you really defined many) and their value will rest in a drop-down menu which displays only the selected value until you click it.

With the *buttons-and-menus* interface, you can change several variables before asking LXR to show the new version with the *Change* button. The *Revert* button restores the value of the variable to their initial state (when entering this view, not their default value). With the provided *lxr.css*, initial values are displayed bold black.

The only annoyance with the *buttons-and-menus* interface is: you need one more click than with the *link* interface. You preset the values of the variables with a click, then you activate these changes with a click on the *Change* button. In the *link* interface the individual changes of the variables take effect immediately, but that means you receive an HTML page after each click, which might be time-consuming on lower performance computers.

The **footer** looks like this:



[Source navigation] [Identifier search] [general search]

This page was automatically generated by the 0.10 LXR engine.
The LXR team

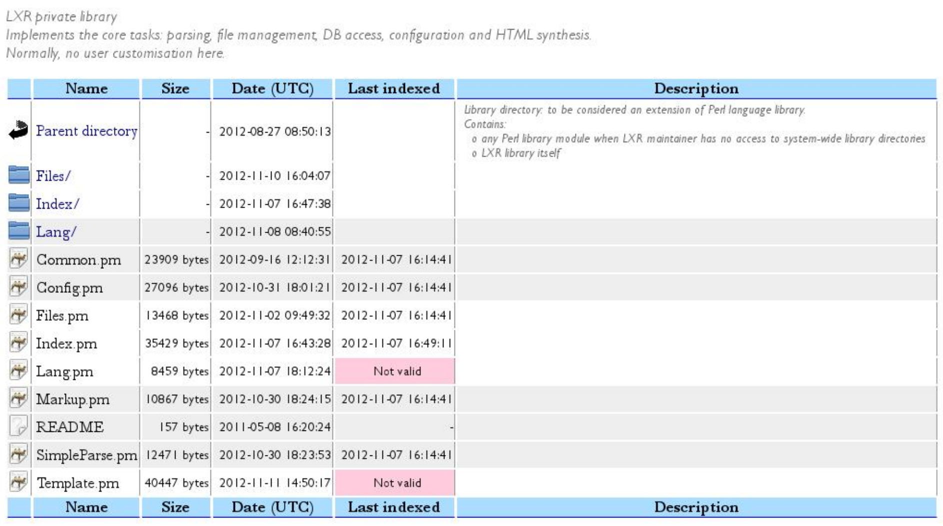
The switching “buttons” are repeated at the bottom of the window. There is a link to the LXR project on *SourceForge* and links to the W3C HTML validation service under the logos at right.

The **body** presents data related to the task requested by the command in the URL (*source*, *ident*, *diff* or *search*). It is described in the following sections.

2.3. *Browsing the source tree: source*

2.3.a. Listing directories

You initially enter this mode. You can always get back to it by clicking on the *Source navigation* button in the header or footer.


LXR private library
Implements the core tasks: parsing, file management, DB access, configuration and HTML synthesis.
Normally, no user customisation here.

	Name	Size	Date (UTC)	Last indexed	Description
	Parent directory	-	2012-08-27 08:50:13		*Library directory: to be considered an extension of Perl language library.* *Contains:* *o any Perl library module when LXR maintainer has no access to system-wide library directories* *o LXR library itself*
	Files/	-	2012-11-10 16:04:07		
	Index/	-	2012-11-07 16:47:38		
	Lang/	-	2012-11-08 08:40:55		
	Common.pm	23909 bytes	2012-09-16 12:12:31	2012-11-07 16:14:41	
	Config.pm	27096 bytes	2012-10-31 18:01:21	2012-11-07 16:14:41	
	Files.pm	13468 bytes	2012-11-02 09:49:32	2012-11-07 16:14:41	
	Index.pm	35429 bytes	2012-11-07 16:43:28	2012-11-07 16:49:11	
	Lang.pm	8459 bytes	2012-11-07 18:12:24	Not valid	
	Markup.pm	10867 bytes	2012-10-30 18:24:15	2012-11-07 16:14:41	
	README	157 bytes	2011-05-08 16:20:24	-	
	SimpleParse.pm	12471 bytes	2012-10-30 18:23:53	2012-11-07 16:14:41	
	Template.pm	40447 bytes	2012-11-11 14:50:17	Not valid	
	Name	Size	Date (UTC)	Last indexed	Description

The text above the directory content table is supposed to be a description for this directory. It is extracted from *README* files (*README.txt*, *README* or *README.html* in that order, first match wins).

> You can take advantage of this feature for your own source trees. By providing such files, an excerpt of their content is displayed and a link created to access the full text.

The icon and the name columns contain links to the directory or file.

Size and date columns show informative data for the file.

For files, column *Last indexed* gives the state of the cross-references relative to this file:

- the date and time when *genxref* was run against this file,
- *Not valid*, a warning flag for files added or modified since last *genxref*,
- a dash for not indexed files (because there is no parser for this kind of file).

Note:

> This information is of primary interest for developers to remind them that references into the modified files may point to the wrong line. To remove the flag, run *genxref*. For stable or archive trees, this column may confuse the user. It is possible to hide it by requesting a different template in the configuration file (see section 8.4 Markers for directory listing).

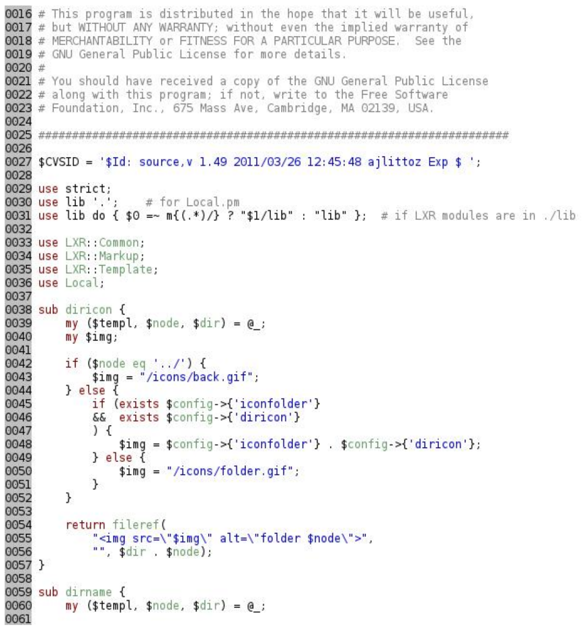
Text in the *Description* column is extracted from *README* files for a directory or from the initial comment for a source file.

2.3.b. Listing files

To list a file, you click on its name or icon in the directory content. To open the file in another tab or window of your browser, right-click the icon or name.

You also enter this mode if you click on the hyperlink for an *include* file.

Note:
Every component of the *include* path file is clickable to transport you directly to this node of the path.



```
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

######################################################################

$CVSID = '$Id: source,v 1.49 2011/03/26 12:45:48 ajlittoz Exp $ ';

use strict;
use lib '.';    # for Local.pm
use lib do { $0 =~ m{(.*)/} ? "$1/lib" : "lib" };  # if LXR modules are in ./lib

use LXR::Common;
use LXR::Markup;
use LXR::Template;
use Local;

sub diricon {
    my ($templ, $node, $dir) = @_;
    my $img;

    if ($node eq '../') {
        $img = "/icons/back.gif";
    } else {
        if (exists $config->{'iconfolder'}
        &&  exists $config->{'diricon'}
        ) {
            $img = $config->{'iconfolder'} . $config->{'diricon'};
        } else {
            $img = "/icons/folder.gif";
        }
    }

    return fileref(
        "<img src=\"$img\" alt=\"folder $node\">",
        "", $dir . $node);
}

sub dirname {
    my ($templ, $node, $dir) = @_;

```

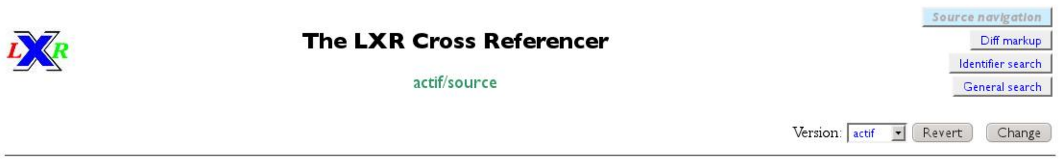
If the file was modified or added since last indexation, the *Warning, cross-references need to be fixed* reminder is printed at the top of the page. Brand new symbols will not be highlighted nor hyperlinked because they are not recorded in the LXR dictionary.

LXR colours the words according to its known classification. The colours are defined in *lxr.css* and can be adapted to your taste. Here user variables and functions are displayed green, reserved words dark red, strings blue, comments light grey … Note that some variables are black, meaning LXR does not know anything about them. Blame ***ctags*** for that. It happens to miss some declarations generally because its language of the given language is too much rudimentary.

Tip:
If you want to display the file without any highlighting, add `?_raw=1` at the end of the URL in the navigation bar of your browser. With this query parameter, you enter *raw mode*. In particular, it can be used to prevent LXR from considering HTML files as “development” files so that they are sent “as is” to the browser to be displayed with HTML layout.

The magic of LXR is: there is a hyperlink under all variables and functions that leads you to the references of that symbol (see section 2.5 Looking up an identifier: ident).

When displaying files, you see a new "button" in the header upper right: *diff markup*. See section 2.4 Comparing two source files: diff for its use.

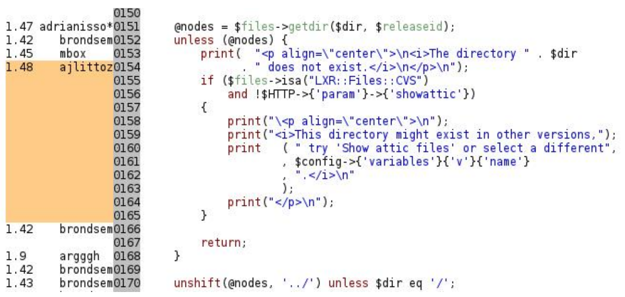


LXR can also display **graphic files** under rather limited conditions. The file must be considered graphic material (this is a matter of configuration in *lxr.conf*), can be accessed through HTML and its format is handled by your browser.

Note:

This feature does not work if your source tree resides in an SCM.

The screen is slightly different if LXR takes the file in a **VCS repository**:



```

1.47 adrianisso*    @nodes = $files->getdir($dir, $releaseid);
1.42    brondsem    unless (@nodes) {
1.45    mbox            print(  "<p align=\"center\">\n<i>The directory " . $dir
1.48    ajlittoz              . " does not exist.</i>\n</p>\n");
                        if ($files->isa("LXR::Files::CVS")
                            and !$HTTP->{'param'}->{'showattic'})
                        {
                            print("\<p align=\"center\">\n");
                            print("<i>This directory might exist in other versions,");
                            print   ( " try 'Show attic files' or select a different",
                                    , $config->{'variables'}{'v'}{'name'}
                                    , ".</i>\n"
                                    );
                            print("</p>\n");
                        }
1.42    brondsem
                        return;
1.9     argggh      }
1.42    brondsem
1.43    brondsem    unshift(@nodes, '../') unless $dir eq '/';
```

A new column is added to the left of the source text. It contains annotation information from the CVS or Git repository: the revision number the line was added or modified and the author. This field is truncated if it is too long to fit[12] (currently 16 characters) and truncation is shown with an ellipsis at right or left according to the VCS to keep the most distinctive fragment. A white field means "same data as in previously marked line".

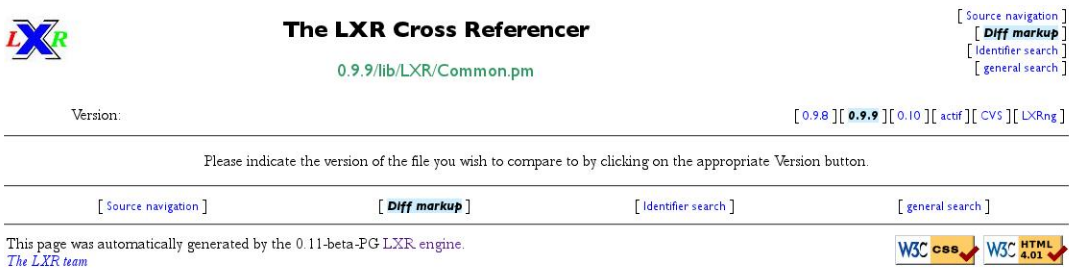
All lines belonging to the same *commit* group share the same background colour[13]. Strong orange background means the lines were added in the latest revision.

2.4. Comparing two source files: diff

If you want to know the differences about a source file between two versions, just click on *Diff markup* when you display that file. You will be asked first to choose the target version:

[12] The revision number displayed under Git is already truncated since it is the 8-hexadecimal digits prefix of the 40-characters SHA1 tag.

[13] This is only local. No effort is attempted to give the same background to different line groups of a single commit.



The LXR Cross Referencer

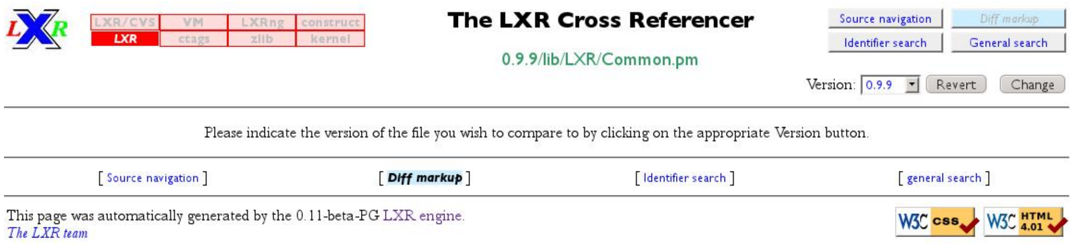
0.9.9/lib/LXR/Common.pm

Version:

Please indicate the version of the file you wish to compare to by clicking on the appropriate Version button.

This page was automatically generated by the 0.11-beta-PG LXR engine.
The LXR team

or:



The LXR Cross Referencer

0.9.9/lib/LXR/Common.pm

Version: 0.9.9

Please indicate the version of the file you wish to compare to by clicking on the appropriate Version button.

This page was automatically generated by the 0.11-beta-PG LXR engine.
The LXR team

The user is currently focused on version 0.9.9. Let him click on the 0.9.8 link or select 0.9.8 from the menu and click on the *Change* button.

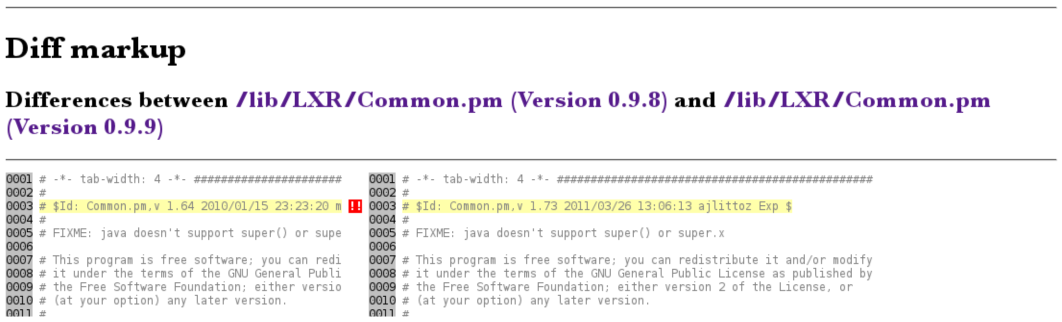
Notes:

diff uses the shell command *diff* but does not give access to all its functionality. It allows to compare only two versions of a file with a given name, *i.e.* the differences will be computed against two file having the same name in different directories.

With the *link* interface, the files may differ by the value of only one "characteristic". A characteristic correspond to one `'variable'` in the source-tree description of *lxr.conf*. At least, you can request a *Version* difference as in the pictures above. Usually, you find an *Architecture* choice in the kernel LXR. You may have more if you find a use for them.

With the *buttons-and-menus* interface, you can set as many "characteristics" as you want by selecting a choice in the menus. Then, click on the *Change* button.

After this selection, you are shown both files side by side.



Diff markup

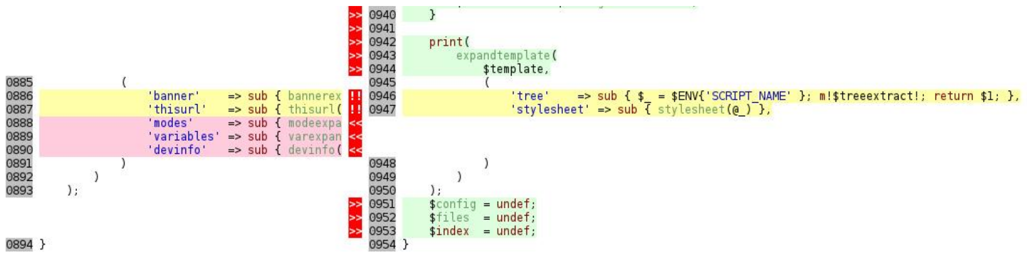
Differences between /lib/LXR/Common.pm (Version 0.9.8) and /lib/LXR/Common.pm (Version 0.9.9)

```
# -*- tab-width: 4 -*- ######################      # -*- tab-width: 4 -*- ###############################################
#                                                  #
# $Id: Common.pm,v 1.64 2010/01/15 23:23:20 m !!   # $Id: Common.pm,v 1.73 2011/03/26 13:06:13 ajlittoz Exp $
#                                                  #
# FIXME: java doesn't support super() or supe      # FIXME: java doesn't support super() or super.x

# This program is free software; you can redi      # This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Publi      # it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either versio      # the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.              # (at your option) any later version.
#                                                  #
```

LXR is kind enough to remind you the name and versions of the files. The most recently selected file is in the left column, the initial file in the right column. This was chosen on the assumption that you are working on a “live” source-tree and you want to compare the current version of a file against an older one. Thus, the wider column is offered to the “live” version. If you want it the other way round, reverse the order of the choices.

The differences are marked with red symbols <<, !! or >> between the two sources. << means new line at left, !! different lines and >> new lines at right. Here is an example:



```
                                                  >> 0940     }
                                                  >> 0941
                                                  >> 0942     print(
                                                  >> 0943         expandtemplate(
                                                  >> 0944             $template,
0885            (                                    0945             (
0886                'banner'    => sub { bannerex !! 0946                 'tree'      => sub { $_ = $ENV{'SCRIPT_NAME' }; m!$treeextract!; return $1; },
0887                'thisurl'   => sub { thisurl( !! 0947                 'stylesheet' => sub { stylesheet(@_) },
0888                'modes'     => sub { modeexpa <<
0889                'variables' => sub { varexpan <<
0890                'devinfo'   => sub { devinfo( <<
0891            )                                    0948             )
0892        )                                        0949         )
0893    );                                           0950     );
                                                  >> 0951     $config = undef;
                                                  >> 0952     $files  = undef;
                                                  >> 0953     $index  = undef;
0894 }                                               0954 }
```

The background colours help identifying the changes: pink for new lines at left, yellow for differing lines and light green for new lines at right. All colours are defined in *lxr.css* for you to play with.

Tip:

The left text column may not show what you are interested in. It can be stretched or shrunk from the URL box of your browser. You only have to set argument `_diffleftwidth` to an appropriate value.

If the URL[14] of the above page is:

```
http://localhost/diff/lib/LXR/Common.pm?v=0.9.9&~v=0.9.9&!v=0.9.8
```

add at the end `&_diffleftwidth=100` to get a 100-characters left column:

```
http://localhost/diff/lib/LXR/Common.pm?v=0.9.9&~v=0.9.9&!
v=0.9.8&_diffleftwidth=100
```

Note:

Do not forget the underscore (_) at the beginning of the argument name.

If you want to permanently use a given width for the left column, consider configuring parameter `'diffleftwidth'` in *lxr.conf*.

You can request a new difference comparison, but it will be computed against the most recently selected version, not the initial version. If you want to run through a series of comparisons against the same base version, you must back up to display that base version before selecting another target version.

Tip:

Alternatively, you can play with the URL. The base version is defined by argument `~v` (or only `v` with the *link* interface). Do not alter it. Change the target version after `!v=` from:

```
http://localhost/diff/lib/LXR/Common.pm?v=0.9.9&~v=0.9.9&!v=0.9.8
```

to, for instance:

[14] It is written on a single line in your browser, but the document processor justifies it on two lines.

```
http://localhost/diff/lib/LXR/Common.pm?v=0.9.9&~v=0.9.9&!v=0.9.7
```

If the difference was requested against another "characteristic" than *Version* associated to variable `v`, you find the name of the variable after `~` and `!`.

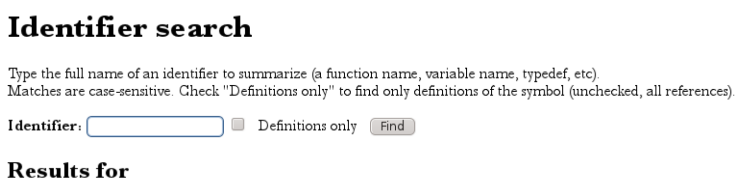
Note:

In the *buttons-and-menus* interface, the base "characteristics" are described by variables prefixed with a tilde character (~), while the target variables are prefixed with an exclamation mark (`!`). The variables without prefix are taken into account only if there is no corresponding `!`variable (`!` means "override").

In the *link* interface, the base "characteristics" are given by variables without prefix character.

2.5. *Looking up an identifier: ident*

Click on *Identifier search* to get:



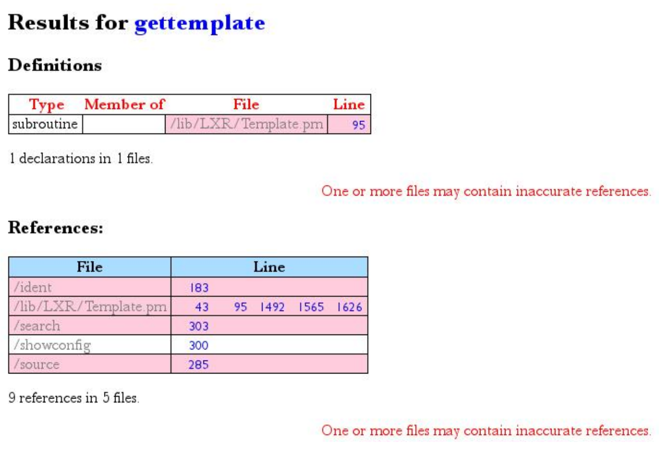
Identifier search

Type the full name of an identifier to summarize (a function name, variable name, typedef, etc).
Matches are case-sensitive. Check "Definitions only" to find only definitions of the symbol (unchecked, all references).

Fill-in the name of the identifier and click *Find*.

An alternate way is to click on an enhanced name in a source display. The form will be preset and the results will appear instantly. If the name is not "decorated", that name is not recorded in LXR dictionary and this occurrence will not be found by *ident*.

The results are displayed as follows:



Results for gettemplate

Definitions

Type	Member of	File	Line
subroutine		/lib/LXR/Template.pm	95

1 declarations in 1 files.

One or more files may contain inaccurate references.

References:

File	Line
/ident	183
/lib/LXR/Template.pm	43 95 1492 1565 1626
/search	303
/showconfig	300
/source	285

9 references in 5 files.

One or more files may contain inaccurate references.

The first table displays the definitions of the identifier, the second the references. Either table may be absent if there is no occurrence in its category.

Coloured background highlights results for modified files since last indexation. The associated references may not be accurate and a reminder is printed after the table.

There may be many definitions because LXR makes no difference between a local and a global symbol, nor between symbols in different languages. There is no way to limit search to one of these types. Some definitions may also be missing if ***ctags*** did not found them.

All the line numbers are hyperlinks to the lines in the files. Click to go there.

Note:

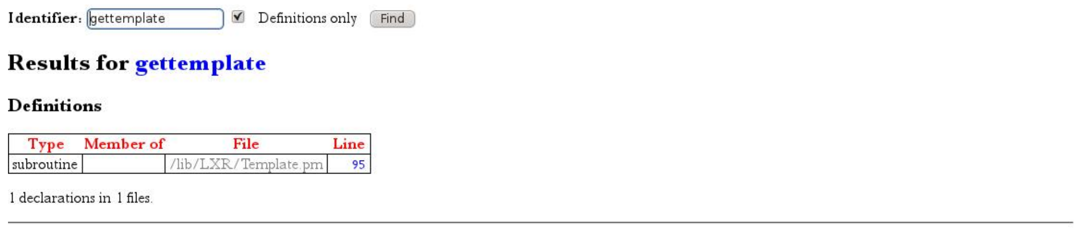
A line number prefixed with a symbol[15] (defined through CSS) denotes a case-insensitive match. This may help spot misspellings or identify related usages such as variable name and its all-capitals initialisation constant in C. It is is also necessary in case-insensitive languages like *Fortran* or *SQL*.

If you click on the file name, you go to the first line of that file.

Tip:

A better use is to right-click to open the destination in a new browser tab so that you have multiple views (usually files) on the identifier with a single query.

Sometimes, the identifier you are looking for is so heavily used that LXR may run into memory shortage when gathering the references (*e.g.* `printk` in the 3.1 kernel occurs more than 55 000 times). If you are only interested in the definitions, you can limit the search to these definitions by clicking the *Definitions only* check box.



Results for gettemplate

Definitions

Type	Member of	File	Line
subroutine		/lib/LXR/Template.pm	95

1 declarations in 1 files.

2.6. *Free-text search: search*

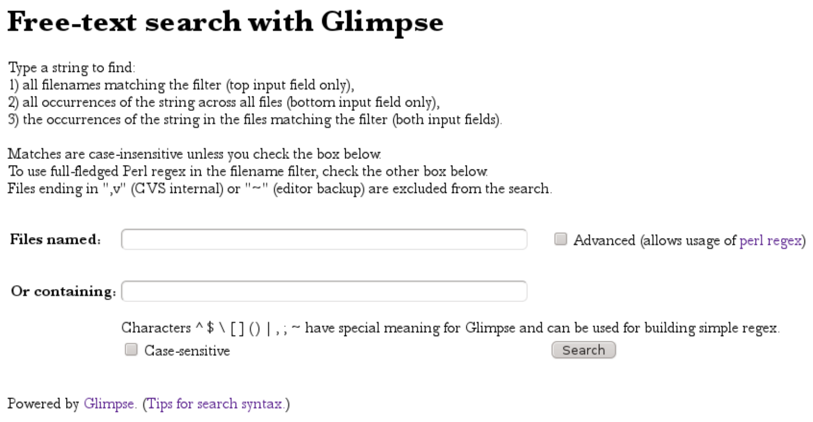
CAVEAT!

Free text search is not available if your tree is managed by a VCS.

In case you stumble on the limitations of the built-in simple parsing of identifiers or you want to query something inside a wider unmanaged unit such as a string or a comment, you must use *General search*.

[15] The default symbol has been chosen to be clearly visible. Unhappily, this Unicode symbol does not reside in the part common with 8-bit character sets. If your tree uses a specific encoding (*e,g,* ISO-8859-1), you must change this symbol to get correct display. Some alternatives are suggested in the provided CSS style sheet. All you need to do is to add a comment prefix `/*` on the Unicode symbol line and remove the comment prefix `/*` on an alternate line.

2.6.a. *Glimpse* engine


Free-text search with Glimpse

Type a string to find:
1) all filenames matching the filter (top input field only),
2) all occurrences of the string across all files (bottom input field only),
3) the occurrences of the string in the files matching the filter (both input fields).

Matches are case-insensitive unless you check the box below.
To use full-fledged Perl regex in the filename filter, check the other box below.
Files ending in ",v" (CVS internal) or "~" (editor backup) are excluded from the search.

Files named: ☐ Advanced (allows usage of perl regex)

Or containing:

Characters ^ $ \ [ ] ( ) | , ; ~ have special meaning for Glimpse and can be used for building simple regex.
☐ Case-sensitive

Powered by Glimpse. (Tips for search syntax.)

Fill-in *Containing* with the text you are searching. Use *Files named* to find a file whose name contains the given string. Both fields may be combined to look for a “string present only in the designated files”.

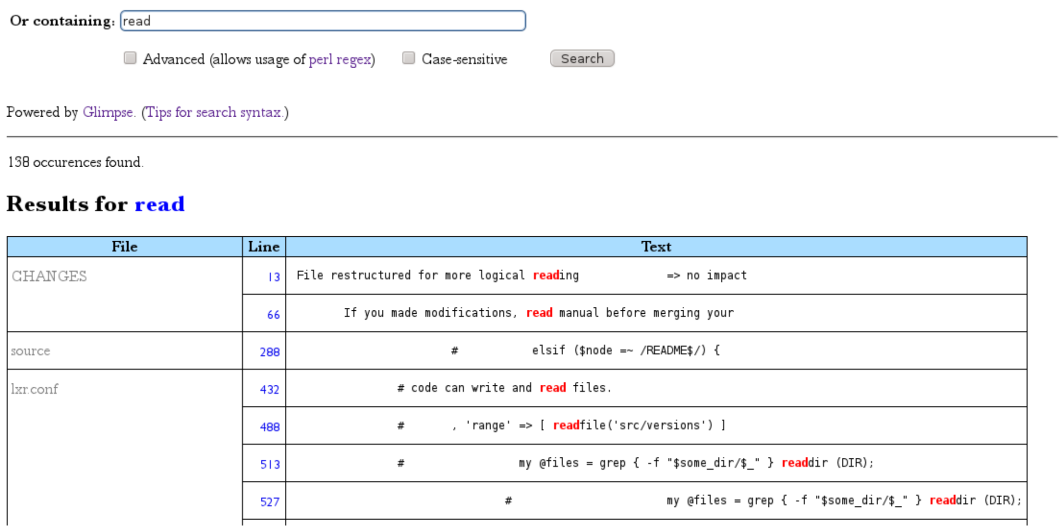
The *Files named* string is compared to the name of the file. If it occurs in any position, the file is retained for content scanning. When the *Advanced* box is checked, the string is considered a *Perl* regular expression and can be as selective or complex as desired (*e.g.* with ^ or $ to anchor the string at the beginning or end of the name).

The *Containing* text is always a ***Glimpse*** regular expression. The ***Glimpse*** regular expressions are simpler than *Perl*'s and have different writing rules and more reserved characters. Consequently, when in doubt, escape your target character with a backslash (\). Consult the ***Glimpse*** manual for details on pattern syntax.

As is usual with free-text query, search is case-insensitive. This helps to catch typing errors but may result in a excessive number of hits in computer related text (think, for instance, of C style coding where a variable is lowercase and the associated constant is uppercase). Check the *Case sensitive* box to force exact match, both for file name and text.

The results appear after a click on the *Search* button, but are limited to 1000 occurrences[16] to avoid memory problems.

[16] Releases prior to 0.11 had a flaw in their algorithm when both fields where used. The file information was used only after having gathered the results. The 1000 limit could be reached before the free-text search engine scanned the relevant files. From release 0.11 on, the algorithm has a correct behaviour where the file filter is applied first.



Or containing: read

☐ Advanced (allows usage of perl regex) ☐ Case-sensitive Search

Powered by Glimpse. (Tips for search syntax.)

138 occurences found.

Results for read

File	Line	Text
CHANGES	13	`File restructured for more logical reading             => no impact`
	66	`If you made modifications, read manual before merging your`
source	288	`#           elsif ($node =~ /README$/) {`
lxr.conf	432	`# code can write and read files.`
	488	`#       , 'range' => [ readfile('src/versions') ]`
	513	`#                   my @files = grep { -f "$some_dir/$_" } readdir (DIR);`
	527	`#                         my @files = grep { -f "$some_dir/$_" } readdir (DIR);`

The first column gives the file where the occurrences are found. The second column is the occurrence line number. Finally, the third column shows the line with highlighting of the target.

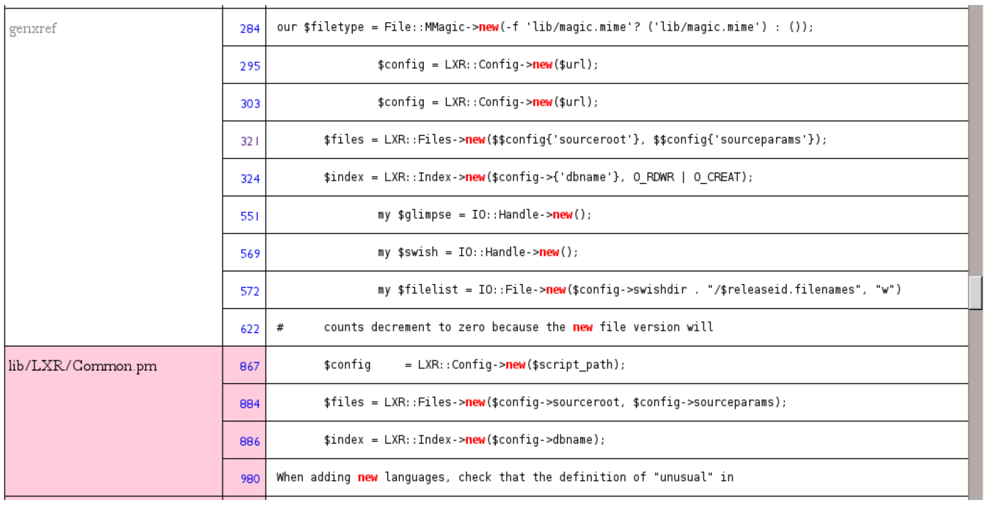
The file name and line number are hyperlinks to the corresponding locations.

As you can see, with a `read` target, this sequence is found in words such as `reading`, `readdir`, `README`, *etc*. Search is case-insensitive unless you check the *Case sensitive* box.

Note:

As expected, the search operates only on the selected version.

File name and line number are specifically highlighted to warn against possible inaccurate references to a modified-since-last-indexation file as:



genxref	284	`our $filetype = File::MMagic->new(-f 'lib/magic.mime'? ('lib/magic.mime') : ());`
	295	`$config = LXR::Config->new($url);`
	303	`$config = LXR::Config->new($url);`
	321	`$files = LXR::Files->new($$config{'sourceroot'}, $$config{'sourceparams'});`
	324	`$index = LXR::Index->new($config->{'dbname'}, O_RDWR | O_CREAT);`
	551	`my $glimpse = IO::Handle->new();`
	569	`my $swish = IO::Handle->new();`
	572	`my $filelist = IO::File->new($config->swishdir . "/$releaseid.filenames", "w")`
	622	`#       counts decrement to zero because the new file version will`
lib/LXR/Common.pm	867	`$config      = LXR::Config->new($script_path);`
	884	`$files = LXR::Files->new($config->sourceroot, $config->sourceparams);`
	886	`$index = LXR::Index->new($config->dbname);`
	980	`When adding new languages, check that the definition of "unusual" in`

2.6.b. *Swish-e* engine

The results are poorer with the ***Swish-e*** engine.

You get the file name where the occurrences were found and a “score” which is an indication of the relevance of the hit, roughly it is related to the number of occurrences in this file.

Swish-e is more oriented toward general information retrieval than toward technical text browsing. Consequently, its search is always case insensitive (the *Case sensitive* check box is effective only against file name matching).

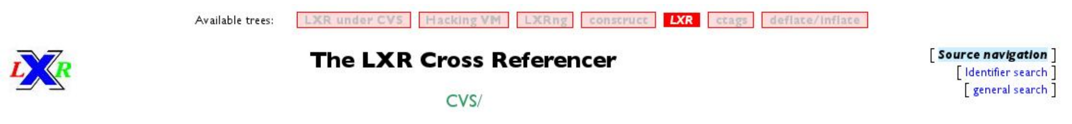
The *Containing* text may be a “bibliographic equation”. Consult the ***Swish-e*** manual for details.

Since ***Swish-e*** is only interested in file hits, there is no line sample.

2.7. LXR in multiple trees context

A single LXR installation may control several source-trees without duplication of its scripts.

If configured so, you can navigate from tree to tree with a single click. The header is modified as shown in the following picture:

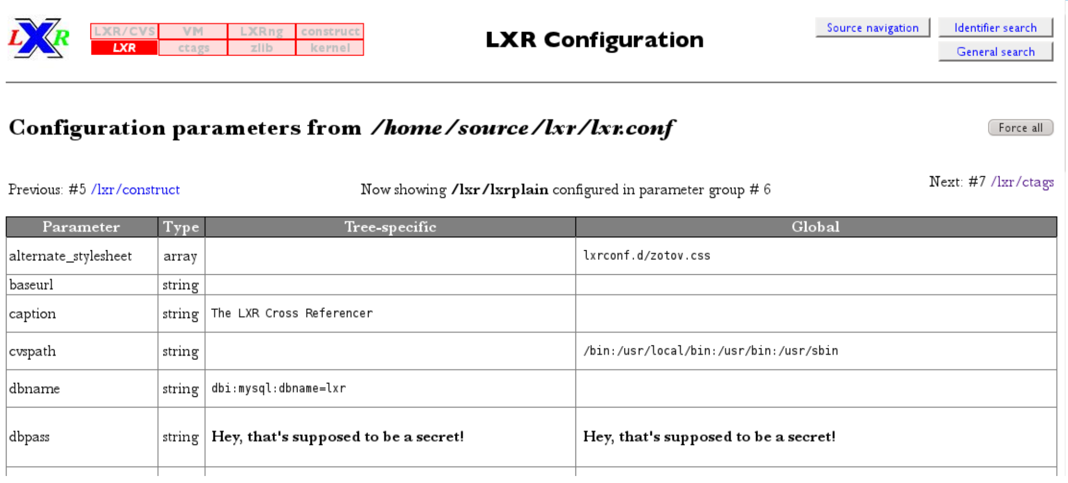


At the very top of the page, a row of clickable hyperlinks transports you to the root of the advertised tree. The current tree is specifically highlighted.

2.8. Checking configuration: showconfig

The tree-maintainer may sometimes be in trouble to correctly configure LXR. Script *showconfig* shows what LXR understood from configuration file *lxr.conf*. Type in the browser address bar:

```
http://localhost/lxr/showconfig
```



Configuration parameters from */home/source/lxr/lxr.conf*

Previous: #5 /lxr/construct   Now showing **/lxr/lxrplain** configured in parameter group # 6   Next: #7 /lxr/ctags

Parameter	Type	Tree-specific	Global
alternate_stylesheet	array		lxrconf.d/zotov.css
baseurl	string		
caption	string	The LXR Cross Referencer	
cvspath	string		/bin:/usr/local/bin:/usr/bin:/usr/sbin
dbname	string	dbi:mysql:dbname=lxr	
dbpass	string	**Hey, that's supposed to be a secret!**	**Hey, that's supposed to be a secret!**

This page gives the OS-absolute path of the configuration file and allows to navigate among the parameter groups: previous group hyperlink at left, `'virtroot'` for display group at the centre and next parameter group hyperlink at right.

The table shows the parameter name, its type, its value in the current parameter-group and in the global parameter-group. If two values are displayed, the tree-specific value overrides the global one.

Some lines may show no value, because the existing parameter is not defined in any parameter group. Think, for instance, of `'filetype'` or `'interpreters'` who may be fetched from a file. To force display, click on *Force all* button.

Note also that the database password `'dbpass'` is concealed. In this preliminary version, array content is not dumped if the array lies inside another structure.

Parameters are listed alphabetically, which may prove inconvenient to correlate with *lxr.conf*.

Finally, remove this script if you think it betrays information facilitating attacks against your site.

3 Indexing the source tree

This chapter is intended for the maintainer of a source tree. It describes the steps and tips to succeed in bringing a source tree to operational life.

3.1. What is indexation?

Indexation of a source tree feeds the internal LXR data base with information about the symbols in the files. It involves detecting and classifying the symbols (as variables, functions, …), collecting their occurrences and preparing the auxiliary data for free text search.

This task is done when the source tree is first installed and every time it is modified.

Note:

If the internal data base is not synchronised with the source tree, files can still be displayed with correct highlighting (with the exception of newer symbols). However, the references given by *Identifier lookup* are usually erroneous.

Indexation is done from the command line with script *genxref*.

3.2. How long does it take?

Quite difficult to answer a not so naive question.

It depends on many factors: the characteristics of the computer running *genxref*, the size of the source tree, its structure (few huge files or numerous small files in various subdirectories), …

As an example, indexing 0.10 LXR on a high-end computer with *MyISAM* storage[17] *MySQL* (3.3GHz 4-processor, 4 GiB memory, fast I/O channels) takes 3.8 seconds (elapsed time). Indexing the same 0.10 LXR on a low-end computer (650 MHz PIII, 512 MiB memory, standard PATA) takes about 39 seconds (elapsed time).

Always launch a sample job before the real big one to get an estimate of the needed time. Indexing a Linux kernel is known to necessitate many hours (with LXR 0.11: 4 hours 15 minutes for kernel 3.1 under *MySQL InnoDB*, 2 hours 33 minutes under *MySQL MyISAM* and 2 hours 51 minutes under *PostgreSQL* on the above fast computer – *see below release 1.0 figures*).

3.3. Genxref parameters

You invoke the indexing script with the command:

[17] Default storage engine changed from *MyISAM* to *InnoDB* between MySQL 5.1 and 5.5 with big impact on performance. It is thus necessary to explicitly specify *MyISAM* in the table descriptions.

```
$ ./genxref options
```

and the following options:

`--help`

Print a summary of the command and its options.

`--checkonly`

Verify critical configuration parameters and stop.

`--url=`*`URL`*

Defines the target source tree. The `URL` must match a combination of one of the `'host_names'` followed by `'virtroot'` as they are recorded in configuration file *lxr.conf*. Alternatively, for compatibility with older LXR releases, the `URL` may be one of `'baseurl'` or `'baseurl_aliases'`.

`--allurls`

Apply *genxref* to all URLs defined in the master configuration file.

Note:

This option should not routinely be used. You have a much better control over the indexing process with repeated applications of `--url=`. It is reserved for special circumstances where it is known that the whole process will terminate without errors in a definite time, such as general database reloading after a system upgrade.

`--version=`*`version_id`*

Generate the index only for the given version of the source tree. `version_id` is equal to one of the version contained in parameter `'range'` for variable `'v'`. It is also the name of a subdirectory in the source tree directory.

`--allversions`

Generate the index for all versions of the source tree (This is the default operating mode of *genxref*).

When *genxref* is applied to a CVS source-tree, it gathers all encountered versions in a file written in *custom.d/* with name `CVS`*virtroot*. *virtroot* is URL-encoded (*i.e.* all non-alphanumeric characters are replaced by an hexadecimal 3-character sequence `%xx`) to avoid potential conflict with path separators. This file may then be used to force variable `'v'` `'range'` attribute.

To suppress file creation, use `--allversions=noauto`.

`--reindexall`

Necessary to remove the previous index; otherwise, if *genxref* notices the index already exists for a file, it skips that file.

Note:

The default behaviour without this option was defined to allow to add a new version to a source-tree or a few files to a version and save time indexing only new material (remember, that step is very time-consuming, so why waste time on something which did not change?).

`--accept`

When a minor configuration error is detected during the initial tests, *genxref* suggests fixes. You may experiment the effect of these fixes with this option.

Note:

Since the fixes are applied on the memory copy of the configuration parameters, there is no guarantee that they will be effective: the configuration file is reread before processing a source tree and the parameters are restored with their saved values. The fixes work well for undefined or missing parameters.

3.4. Last petrol station before the desert

It is always wise to launch *genxref* with option `--checkonly` before the real indexation job. This is a very short task (less than 10 seconds) with a high value return.

The status information is not lost among the progress messages. You quickly identify the potential problems needing an eventual revision of the installation.

Genxref *output now uses colour to draw attention onto important information. To improve readability, set your terminal background colour to black or, better, to light grey (HTML* #BBBBBB *should do fine).*

```
$ ./genxref --url=//localhost/lxr --reindexall --version=1.0 --checkonly
[   OK  ]      Perl     version ... 5.14.2
[   OK  ]     ctags     version ... 5.8
Checked:    glimpse     version ... 4.18.5
Checked: glimpseindex version ... 4.18.5
Parameter 'swishbin' not defined - trying to find swish-e
swish-e not found, `command -v swish-e` returned a null string
genxref stopped without indexing by --checkonly option
```

genxref performs a test on the needed tools. If the tool is found, a line displays the test result. It may read:

- `Checked:` the tool exists and there is no version requirement;
- `[   OK   ]` the tool exists and satisfies the version constraint;
- `[FAILED]` the tool exists but does not pass the version constraint.

The last case is a fatal error. Indexing will not be done until you (or your system administrator) install the adequate version.

Note:

LXR runs also with privately installed tools provided the files are *world readable* and their locations are correctly given in configuration file *lxr.conf*. This is a work around if you cannot get your system administrator install ***ctags***, ***glimpse*** or ***swish-e***.

If a tool is not found, the printed message explains the cause: either a parameter is not defined or the tool does not exist in the system. This may or may not be a fatal error depending on the tool.

The last line reminds you you launched *genxref* with `--checkonly`, which is a good reason for it to stop now.

It is not an error to have only one of ***glimpse*** or ***Swish-e***. One must be present for free text search. But configuration parameters `'glimpsebin'` and `'swishbin'` cannot be both simultaneously defined.

3.5. Running the indexation

Genxref *output now uses colour to draw attention onto important information. To improve readability, set your terminal background colour to black or, better, to light grey (HTML* `#BBBBBB` *should do fine).*

The command to launch the indexation is similar to:

```
$ ./genxref --url=//localhost/lxr --reindexall --version=1.0
```

Use of the `--reindexall` option is recommended for "*small*"[18] trees since it involves an exhaustive scan of the sources.

If you do not use the `--reindexall` option, the existing index is kept and only the changed files are indexed, but the tree must be traversed anyway. There is a small penalty with the data base growing: the information related to the previous state of the changed files is not fully purged.

The *genxref* script does its jobs in four passes for every version.

The first optional pass erases LXR database base information (files, definitions and references). Depending on the source structure, it is either quasi-instantaneous or takes a time between those of the third and fourth passes since it must query the state of every file description in the LXR database.

This pass prints something meaningful only when indexing a single version without option `--reindexall`. In the others cases, everything happens in the database engine (usually running as a daemon).

It reports the files (and their revisions) which will be erased from the database:

```
%%% version path_to_directory
```

- *path_to_directory* is a pathname relative to `'sourceroot'` where the following files need processing

```
--- version filename revision status
```

- *version* is the version name as listed in `'range'`
- *filename* is the name relative to the current directory as printed in a %%% line

[18] "Small" is to be understood as a source-tree size giving an acceptable indexing time on your computer. Many factors compete to give the final time. If you are working on a huge project, such as the Linux kernel, running *genxref* every day with `--reindexall` is clearly unacceptable since it means several hours of processing (on high-end computers). On the other hand, if complete reindexing only takes a few minutes (up to, say, two minutes), it is much safer to use this option.

- *revision* is an internal code uniquely identifying the base file for this version
- *status* is one of the following:

```
not purgeable yet
purged
```

The first status means the base revision file is still referenced from an other version and cannot be removed. However all definitions and references pertaining to the current version are erased from the database.

The second status indicates that the base revision file has successfully been removed from the database. The following parsing passes will add symbols for a consistent result[19].

The second pass generates the search database with *glimpseindex* or *swish-e*. What is printed is the output of the relevant search engine.

With *swish-e*, the following line is printed before processing a directory or file:

```
&&& path_to_file_or_directory version
```

While *swish-e* needs a copy of each file with some header information added with its internal index files, *glimpse* undertakes a specific processing of the tree as a whole involving creating macro block indexes and arranging a private database.

The third pass collects symbol declarations in the LXR database with the help of *ctags*.

The following message is printed when entering a directory[20] [21]:

```
*** version path_to_directory
```

- *path_to_directory* is a pathname relative to `'sourceroot'`

Processing of a file is logged as:

```
--- version filename revision status
```

- *version* is the version name as listed in `'range'`
- *filename* is the name relative to the current directory[22] as printed in a `***` line
- *revision* is an internal code uniquely identifying the base file for this version
- *fileid* is a short number identifying the file
- *status* is one of the following:

```
fileid
was already indexed
```

[19] This solves a bug filed since 2002! Reindexing a file without `--reindexall` left previous definitions and references to a symbol as “ghost” occurrences resulting in false duplicates in cross-references reports.

[20] As long as no processing is done in the directory, this line is overwritten with the next directory. This is done to minimise scrolling. This results in unprocessed directories being erased from the log.

[21] This line is repeated every now and again, so that the current directory is kept in view despite of scrolling even in very populated directories.

[22] With source trees residing in a CVS repository, files in the *Attic* are not correctly attributed to the parent directory if it contains sub-directories. This will not be corrected because it would a negative impact on performance on all other cases.

```
FAILED
```

The first status means “normal” processing.

The second status indicates that this file has already been indexed in a previous run or version and is not parsed again. Reindexing may be forced with `--reindexall` option.

The third status is used if index files could not be created during initialisation.

Messages like `Warning: Unknown type` *letter* may follow the `---` line. This means you have not fully configured `'typemap'` for the language in *generic.conf* (discrepancy between this section and `'ectagsopts'`) preventing the symbols for the *letter*-category from being indexed.

The fourth pass parses the files to get references to symbols detected in the third phase. These references are entered in the LXR database. This pass is the one taking the longest time.

The following message is printed when entering a directory:

```
### version path_to_directory
```

Processing of a file is logged as:

```
--- version filename revision status
```

where *status* is one of the following:

```
fileid +++ number_of_lines_in_file
was already referenced
FAILED
```

The meaning of these statuses is the same as in the third phase.

Note:

fileid *is printed before entering the file and* +++ number_of_lines_in_file *when the parser reaches the end of file. You may then perceive a delay between these two data.*

What looks like an identifier and is not in the reserved symbol list is referenced if it has been seen during the third pass.

You may get messages `BTYPE was: some_category` after the `---` line. It means a fragment of the source file could not be classified as a managed category (like comment, string, include or code) and was ignored.

IMPORTANT NOTICE!

The internal LXR parser has been deeply changed between versions 0.9.8 and 0.9.9. You should no longer get this message with `atom` for `some_category`. If it ever happens[23], please report it immediately as a bug with as thorough context related to your source tree as you can.

If `some_category` is something else than `atom`, this means you made modifications to the language specifications in *generic.conf* without adapting the parser. If you did not, report it also as a bug with a copy of file *generic.conf*.

[23] Experience has taught the author you can never say never with computers.

3.6. *Troubleshooting*

With the *PostgreSQL* data base engine, the optional selective erasure pass prints `commit ineffective with AutoCommit enabled at` .... Ignore this warning. It is caused by the inability to change the *commit* mode dynamically[24].

Error messages issued by the free-text search engines during the second pass are often caused by non-existing private directories (those defined by configuration parameters `'glimpsedir'` and `'swishdir'`) or incorrect access permissions (*user and group writeable* and *world readable* is recommended).

During the third pass, `ctags: No files specified. Try "ctags --help"` means the directory defined by configuration parameter `'tmpdir'` cannot be written into. Check the access permission (*user and group writeable* and *world readable* is recommended).

Message `Can't call method "getline" on an undefined value at` ... during the fourth pass is also caused by incorrect access permissions on subdirectory `'tmpdir'`.

3.7. *Optimising resources usage*

3.7.a. File revision *vs.* file versions

A *file version* is what you see in the different directories of the tree. However the VCS's try to minimise the number of different versions they store. If two versions are identical, they will keep only one, named a *revision*, which will be known under two different *version* names.

Internally, LXR converts file *version* into *revision* and makes a note when a revision is processed. This is how *genxref* skips files with the *file* `was already indexed` message, even if you are indexing a seemingly brand new version whose files where never submitted to LXR. LXR simply determined that the file was already present in another version.

For plain files (as opposed to a VCS), there is no storage engine. The base revision factor associates last modification date and size of the file. Two files (with the same name) in two versions refer to the same revision if their factors are equal. Thus when copying files to open a new version directory, it is worth keeping the last modification date unchanged.

When option `--reindexall` is not given to *genxref*:

```
$ ./genxref --url=//localhost/lxr --version=current
```

changed files are recorded under a new revision (either defined by the VCS or a change in modification date and/or file size). These new revisions are indexed. The older revision information is deleted if no other file version refers to this revision. However, the database may grow a bit[25]. It is

[24] This seems to be a specific *Perl* issue when using object-oriented programming.

[25] As the erased file is supposed to be immediately reindexed, the symbol dictionary is not erased because it is anticipated it will be little different in the new version. This spares time needed to rebuild the dictionary. Some

recommended to periodically run the following command on a "live" working version to prune inaccessible stale data:

```
$ ./genxref --url=//localhost/lxr --version=current --reindexall
```

3.7.b. Processing time

3.7.b.1. *Purging the database*

There are two methods for purging the database: brute force or detailed analysis.

Brute force is very fast and can be used only when it is known to keep database integrity. This is the case when both `--reindexall` and `--allversions` options are simultaneously given to *genxref*. It is also used without `--allversions` where there is only one version in the tree:

```
$ ./genxref --url=//localhost/lxr --allversions --reindexall
$ ./genxref --url=//localhost/lxr --version=sole_version --reindexall
```

In the other cases, *genxref* must carefully analyse the file descriptors to identify the purgeable records:

```
$ ./genxref --url=//localhost/lxr --allversions
$ ./genxref --url=//localhost/lxr --version=one_version_among_others
```

Purge time is closely related to the number of file versions and number of symbol occurrences.

The Linux kernel 3.x is composed more than 36 800 files out of which 32 000 are indexed, contains roughly 1 300 000 different symbols for 2 500 000 definitions and 18 500 000 usages. Delicately purging a single version may take 1 hour 45 minutes on the aforementioned high-end computer. Indexing is not included in this figure. It is worth considering twice on such projects before deciding for `--reindexall`.

Not specifying `--reindexall` allows to parse only the changed files since the last indexation.

If only one version needs to be indexed, explicitly spare file state analysis on the other versions:

```
$ ./genxref --url=//localhost/lxr --version=one_version
```

The same is true if you have few versions to update. *genxref* them individually one after the other.

3.7.b.2. *Changing database engine*

All database engines are not equivalent and their performance make a big difference on indexing time. The difference is less noticeable on browsing where time is dominated by HTML generation.

From "quick and dirty" comparisons on a medium size test case (7 versions of LXR), *MySQL* is a bit faster than *PostgreSQL* (about 20%). *MySQL* is roughly 4 times as fast as *SQLite*, while *PostgreSQL* is about 3 times. This differs little from previous estimates[26].

symbols will not be present in the new text but their dictionary entry is kept. The result is a marginal database size growth.

[26] Version 0.11 stated *"default MySQL storage engine (InnoDB) is approximately twice as fast as SQLite and*

The database interface has been heavily worked upon in release 1.0, but the difference in *MySQL* between *InnoDB* and *MyISAM* remains the same: *MyISAM* is still twice as fast as *InnoDB*.

On the aforementioned high-end computer, the *real* times (in minutes:seconds) are 1:48 *SQLite*, 0:55 *MySQL InnoDB*, 0:32 *PostgreSQL* and 0:28 *MySQL MyISAM*.

The 1.2 ratio increases to 1.6 on large-scale projects: Linux 3.1 indexing times are 4:09:47 with *PostgreSQL*[27] and 2:39:30 with *MySQL MyISAM*.

3.7.c. Database size

The main contributions to database size are in decreasing order usages, symbol names and definitions. Depending on language parser analysis depth, total stored information may be several times bigger than the source-tree itself (counting the access indexes). Project programming style is also an important factor (comment density, symbol use frequency, …).

> *The Perl scanner (in present* ***ctags****) captures only procedures. Database has roughly the same size as source code.*
>
> *On the opposite, C-family parsing is rather exhaustive. The expansion factor is always greater than one. It reaches 5 on complex projects.*

When the number of files is important, the files dictionary may also be a figure to take into consideration. The stored file names contain the full path starting from `'sourceroot'`. With deep directory nesting, these paths can be quite long (maximum length around 100 in the Linux kernel tree). There is also at least one *version* string per *revision* file.

Index size is roughly the same as data size, but for the symbol dictionary where indexes may need up to twice as much space as data.

> *All put together, under MySQL InnoDB, an empty LXR database is 256 kbytes. A small user project (about 400 kbytes) results in a 1.4M database. At the other end of the scale, a Linux 3.1 kernel (450 Mbytes) needs a 2.7G database.*
>
> *Under MySQL MyISAM, an empty LXR database is 77 kbytes. The small 400k project results in a 630k database and Linux 3.1 kernel leads to a 915M database.*
>
> *Under PostgreSQL, an empty LXR database is 6.0 Mbytes. The small 400k project results in a 7.8M database and Linux 3.1 kernel leads to a 1.8G database.*
>
> *Under SQLite, the small 400k project results in a 500k database. And do not ever try to index a Linux kernel under SQLite.*

Once these figures are collected, it is possible to optimise database organisation for size through tuning the *initdb*-x-*template.sql* (after copying them from *templates/* to *custom.d/*) or the *initdb.sh* script if it has been kept. This requires SQL and database knowledge.

PostgreSQL is in turn twice as fast as MySQL InnoDB or as fast as MySQL MyISAM". The times were 1:48 *SQLite*, 1:08 *MySQL InnoDB*, 0:32 *PostgreSQL* and 0:30 *MySQL MyISAM*. Note that the newer test case is bigger than for 0.11 and 1.0 does a lot more things.

[27] Time can be marginally improved by playing with the internal *PostgreSQL* sequence generator cache. Considering there are 1.3 million symbols, setting cache factor in `symnum` sequence to 1 000 makes sense and resulted in a 10 minutes gain. However using higher values is likely to have a very limited impact.

Notes:

Being too conservative on table column sizes may hamper future version addition if names (files and symbols) are longer than the new limit.

1a After tuning an *initdb*-x-*template.sql* template, a full re-installation of the tree into LXR must be done, but you can spare some pain in multiple-trees context to avoid having to describe all the other trees: take care not to modify your existing *lxr.conf* if you want to tune only one database. Run *configure-lxr.pl* as:

```
$ ./scripts/configure-lxr.pl --tmpl-dir=custom.d --conf-out=mylxr.conf
```

Option `--tmpl-dir` tells to take templates in *custom.d/* and `--conf-out` gives a user name to this configuration file, which will be discarded anyway.

1b Tuning *initdb.sh* requires to purge the script from commands and SQL statements not related to the relevant tree. Make the changes in the database description.

2 Recreate the database with:

```
$ ./custom.d/initdb.sh
$ ./genxref --url=//localhost/lxr/tree
```

4 Configuring LXR

This chapter is an introduction to fine tuning LXR to satisfy one's personal taste or need. The intended audience is the advanced user, the tree maintainer and the developer.

LXR needs to know its environment to find all the required auxiliary tools and the various directories. It must be fed with the parsing rules for the supported languages. Its raw results must be edited before being displayed for the user.

This sums up what can be changed to obtain dramatic differences with the same base engine.

4.1. *Understanding file references in LXR*

File references are passed LXR to serve different purposes. LXR is an hybrid "object" running in three environments.

CAUTION!

Remember to check that all the paths are "*world readable*" because LXR is run by the web server under its own user-id. Most difficulties arise from incorrect file permissions.

4.1.a. LXR is mainly a set of scripts written in *Perl*

The scripts use service routines grouped in *Perl* packages[28]. Seen from the OS, a package is simply a file. To avoid maintenance problems when moving LXR from a directory to another, the packages are referenced using a *relative* form. The only constraint is to define the *root* of this relative form.

When an LXR script is launched, it defines its storage directory as the working directory. This working directory is the root of all file references from within the script. Since the service packages are shared among all the LXR scripts, the scripts must be stored in the same directory, called the *LXR root directory*.

Note:

Well, it is not that simple. *Apache 1.x* `mod_perl` module has some limitation which prevents calling the service packages with this uniform mechanism. Under *Apache 1.x*, the service packages must be copied in the *site_perl* directory of the standard library.

To summarize, files needed by the scripts must be located in the *LXR root directory* or one of its subdirectories.

4.1.b. LXR uses auxiliary tools or accesses non-specific files

These tools, such as ***ctags*** or ***glimpse***, or files, like a source-tree, are totally unrelated to LXR. They

[28] Packages here means those written specifically for LXR, not the standard library packages which are supposed to be always available.

can reside anywhere in the computer.

There is no other choice than naming them with an *OS absolute path*.

4.1.c. LXR emits HTML code

This HTML code may reference utility files, *e.g.* CSS style sheets. The rules for HTML environment are applied. These files, to be accessed, must reside in the `DocumentRoot` directory or one of its subdirectories.

On first sight, HTML-relative and HTML-absolute path forms can be used. However, the HTML-relative form is based on the current page position in the `DocumentRoot` hierarchy. Consequently, this relative form cannot be used because it would be different for pages in different branches of the hierarchy.

In simple cases, the `DocumentRoot` directory and the *LXR root directory* are the same and the distinction between HTML and LXR does not matter.

In complex cases, such as LXR service being integrated in a wider site or site-specific security rules[29], `DocumentRoot` is totally separated from the *LXR root directory*. This implies a copy of some files from the distribution LXR directory to some directory under `DocumentRoot`.

4.2. *Configuration files*

First thing first, LXR reads its master configuration file *lxr.conf* from the *LXR root directory*.

Note:
At this stage, LXR has yet no information on its user-environment. It is kind of bootstrapping. The file name *lxr.conf* cannot be user-customised without script modification.

This file contains the paths for all the other files needed for operation. To select the correct designation as stated above, one must understand which use LXR has for the file.

The principal other files to have an impact on LXR behaviour or appearance are *generic.conf* for language definitions, the templates for the HTML page architecture and the CSS style sheets.

Note:
The default CSS style sheet has always been named *lxr.css* but its name is defined in *lxr.conf* and can be changed to whatever you like.

These files are covered in the next chapters.

Golden rule reminder:
Always copy original files from the *templates/* directory to the *custom.d/* directory before editing. *templates/* has been made “read-only” to avoid accidental loss of reference files.

Note:

[29] For instance, *SourceForge.net* forbids CGI scripts in “normal” `DocumentRoot`. All scripts are segregated in *cgi-bin* directory.

Do not forget to reset permissions on the copy, otherwise it stays “read-only”.

4.3. Configuration scripts

The scripts in the *scripts/* directory are provided to facilitate the configuration process. For these scripts to be successful, the *current working directory* must be the *LXR root directory*.

- *ANSI-escape.sh*, *LCLInterpreter.pm*, *QuestionAnswer.pm, Tagger.pm* and *VTEscape.pm*

  contain definitions and procedures to be used by other scripts.

- *configure-lxr.pl*

  Main configuration script, see 1.3 Configure your installation for usage and 1.3.f for options.

- *kernel-vars-grab.sh*

  This script is a companion to template *lxrkernel.conf*. It scans a Linux source tree (only when stored as plain files – does not work with VCS's – to discover values to set in `'range'` attribute of variables (`'v'` version variable, `'a'` architecture variable and others to drive the `#include` resolution engine). The output is a set of files with suffix `_list.txt` in *custom.d/* directory.

  ```
  $ ./scripts/kernel-vars-grab.sh kernel_directory
  ```

  The output is added to the end of these files. To restart from scratch, not keeping values from other directories, add option `--erase`:

  ```
  $ ./scripts/kernel-vars-grab.sh --erase kernel_directory
  ```

  To change the default suffix, use option `--suffix` and edit *custom.d/lxrkernel.conf* to reflect the change:

  ```
  $ ./scripts/kernel-vars-grab.sh --suffix=_my_suffix.txt kernel_directory
  ```

- *lighttpd-init*

  This is an */etc/init.d/*-like script to control the *lighttpd* web server. See 1.8.b Lighttpd server.

- *recreatedb.sh*

  This script recreates database descriptions from the master configuration file (see 4.5 Reloading LXR after system upgrade).

- *set-lxr-version.sh*

  When you modify LXR (its internal *Perl* scripts or templates), use this script to identify your copy as a customised one. The command is:

  ```
  $ ./scripts/set-lxr-version.sh --user "-ajl-1.0"
  ```

  The string is added at the end of the release number.

The `--user` *option is mandatory because this script is also used during the release procedure to set the release version number. If you do not specify* `--user`*, your custom identification will not be added (because the Perl version subroutine looks different in the development stage – it is a template – and in the public release) and you get no error indication (blame* sed*).*

4.4. Multiple-trees context

Configuring for multiple trees is hardly more difficult than configuring for single tree.

Installation (1.2) is independent from usage. You can proceed with the following steps at any time.

In configuration step (1.3), answer `m` (for multiple) instead of `s` (for single) and proceed along with the questions.

Depending on the database engine, you have the choice of creating a database per tree (*MySQL*, *PostgreSQL* and *SQLite*) or creating tables dedicated to a tree in a single database (*MySQL*, *Oracle*, *PostgreSQL* and *SQLite*).

Be careful when adding the trees in several sessions not to erase the already existing databases: this involves deleting *initdb.sh* script or moving it under a different name after use.

Run *genxref* (see 1.7) for every tree.

Configuring the web server is the hardest step. Read carefully chapter 7 Web server configuration and its sections on multiple-trees operation.

4.5. Reloading LXR after system upgrade

Some system upgrades may lead to LXR databases loss. Among others, consider installing a new LXR version or fresh installation of the next system distribution release.

There is a solution to avoid the painful reconfiguration of LXR, above all when you manage many, many trees.

In the first case, save the master configuration file *lxr.conf* and its context file *custom.d/lxr.ctxt*. Save also the web-server related files (*.htaccess* and in *custom.d/ apache2-require.pl*, *apache-lxrserver.conf* and *lighttpd-lxrserver.conf*). Install the new LXR version (step 1.2 Create LXR installation directory) and reload the saved files. Do not configure this new installation, proceed to the procedure below.

In the second case, before upgrading your computer, save the *LXR root director*y and the source directories (more generally, save */home* content – easily done if it is a separate partition). Do the upgrade and reload what you have saved. Then follow the procedure below.

Note:

Ideally, you could also save the databases but that requires knowledge of the directories where they are stored and of the companion configuration files. But you have no guarantee that after reload, the data will still be compatible with the database engine.

- Step 1.3 Configure your installation is replaced by the invocation of script *recreatedb.pl*:

```
$ ./scripts/recreatedb.pl --verbose
*** LXR DB initialisation reconstruction  (version: 1.1) ***

LXR root directory is /home/source/lxr
Configuration read from lxr.conf

Initial context custom.d/lxr.ctxt is reloaded
Your DB engine was: MySQL
Configuration file lxr.conf loaded

*** scanning global configuration section ***

*** scanning /lxr/tree1 tree configuration section ***
*** scanning /lxr/tree2 tree configuration section ***
```

  Note:
  Advanced uses of this script can be hinted by option `--help`.

- Do step 1.4 Create a database normally, executing script *custom.d/initdb.sh*.
- Skip steps 1.5 Edit the LXR configuration file and 1.6 Copy configuration since the master configuration file was already operational.
- Step 1.7 Generate index is replaced by:

```
$ ./scripts/genxref --allurls
```

  Note:
  This is the only instance where the use of `--allurls` *is recommended because indexation is supposed to have been previously done with success. However, since it can be very lengthy, it is wise to launch this command in a separate terminal so that it runs in the background. Of course, indexation can also be split into independent sub-tasks with* `--url=`.

- Copy the web-server related files to their final destination. Restart the web server and you are done.

5 Master configuration file *lxr.conf*

This chapters covers in detail the content of the master configuration file. Its content is central to flawless operation.

The master configuration file *lxr.conf* must be located in the *LXR root directory*. It contains generic parameters needed for any LXR operation.

5.1. Master configuration file syntax

The master configuration file contains a list of **key/value pairs**.

Note:

If you are familiar with *Perl*, it is a list of anonymous *hashes*. It is read in at initialization time and evaluated. What this means is: you get all the power of *Perl* and all its idiosyncrasies.

Comments can be added to the file to give explanations or take note of modifications. A comment starts with a pound sign (#) followed by a blank (space or tab) and extends to the end of line.

Examples:

```
# This a comment starting in col. 1
'relevant' => 'data' # This comment follows meaningful configuration data
```

The configuration parameters are enclosed in a (parenthesised) **list**:

- opened by a left-parenthesis `(`
- separated by a comma `,`
- closed by a right-parenthesis `)`

```
( parameter-group-0 , parameter-group-1 ... , parameter-group-n )
```

or, with a nicer presentation[30]:

```
( parameter-group-0
, parameter-group-1
 ...
, parameter-group-n
)
```

[30] The concept of *nice* is a matter of personal taste, of course. However, the author of this manual has found by experience that programming in languages where **separators** are used, it is safer to write them at the **beginning** of the line, so that they are not omitted by mistake. Moreover they align neatly with the opening and closing symbols. **Delimiters** (or 'terminators') are written at the **end** of the line. In *Perl*, commas are separators and semi-columns are traditionally used as delimiters, though they really are separators.
The wise rule is: in lists, always write a symbol at start of line: the opening one of the list, the continuation separator or the closing symbol of the construct on its own line.

Notes:

The ellipsis … above only denotes the presence of more parameter-groups with their preceding comma.

Each parameter-group is list of key/value pairs (with a *hash* syntax for the technically minded):

- opened by a left-curly-brace `{`
- separated by a comma `,`
- closed by a right-curly-brace `}`

```
{ key/value-0
, key/value-1
 ...
, key/value-m
}
```

In these pairs, the **key** is a string (delimited with single-quotes `'` to avoid funny things from happening when evaluating the parameter). The **value** may be a (single-quote delimited) **string**, an **array** or even another **list of key/value pairs**. Each pair contains:

- the quote-delimited key
- the operator `=>` (equals sign followed by right-angle-bracket, also named *greater than*, without any intervening space)
- the value as one of:
  - a single-quote delimited string for single value parameters
  - an array of values opened with a left-square-bracket `[` separated by a comma and closed with a right-square-bracket `]`
  - a list of key/value pairs (see above)

Example:

```
(  # First parameter group
    { 'number_0' => '0'
    , 'array' =>
          [ 'elem0'
          , 'elem1'
          , 'elem2'
          ]
    , 'secondary_hash' =>
          { 'sec_key_a' => 'a'
          , 'sec_key_b' => 'b'
          }
    }
,  # Second parameter group
    { 'another_key' => 'as_an_example'
    }
)
```

Note:

This gets evaluated by *Perl*. Thus, there is a more elegant and less error-prone way of writing the array above by using the *Perl* function `qw()` (meaning *quote word*):

```
, 'array' => [ qw(elem0 elem1 elem2) ]
```

We are thus spared the trouble of writing the quotes and forgetting one. `qw()` splits its argument list at white space.

5.2. Rationale of parameter grouping

Now we know how to write the configuration file. Let us see how it is structured.

The first parameter-group (at index 0) is always read in and contains general parameters needed to have LXR functional. This group is **required**.

The subsequent parameter-groups (at indices 1 and above) are akin to a source tree and one is read in only when that source-tree is displayed. At least, one such parameter-group must be present and matched to the tree lest LXR dies (silently in the standard version, leaving you with a blank screen).

That's the original story. But you can improve on it.

1. The parameters are all stored in a single *dictionary* whether they are read from group 0 or others. It means that, if you have several trees with identical tree-parameters, you can “common-factor” the identical tree-parameters by writing them only once in parameter-group-0.
2. The tree-parameters are read **after** the group-0 parameters. It means you can override a general value with a tree-specific value by issuing the same key with a different value in a parameter-group-1 or above.

For instance, these two tricks can be used if you want to customise one of the HTML templates for a tree only, instead of listing them repetitively (with the risk of a typing error).

5.3. Sections in provided lxr.conf template

Directory *templates/* located in the *LXR root directory* contains a commented template for the master configuration file *lxr.conf* which can no longer be directly tailored to one's needs due to the expansion directives interspersed with the parameters. As explained in chapter 4 Configuring LXR (see mostly 4.1.a and 4.2), the resulting expanded file MUST be located in the *LXR root directory*. You can of course customise this template but make a copy of it:

```
$ cp templates/lxr.conf custom.d/my_lxr_template.conf
$ chmod u+w custom/my_lxr_template.conf
```

Note:

This command is valid if your working directory is the LXR root directory.

You can now adapt it to your needs with your favourite text editor. When you are done, generate the effective configuration file with (on a single line):

```
$ ./scripts/configure-lxr.pl -vv --tmpl-dir=custom.d --conf-out=lxr.conf
my_lxr_template.conf
```

Eventually tune the resulting file. Copy it to its final location:

```
$ cp custom.d/lxr.conf .
```

Comments in the file are used to provide reference marks. Parameter group 0 follows the *Global configuration section* header. Parameter group 1 after the *Tree configuration section* header is a parameter group dedicated to a tree. If you want to manually add another source tree, duplicate this section (including its comma, from the opening left-curly-brace to the closing right-curly-brace) just before the `#@here_tree:` tag and final right parentheses (see 5.1 Master configuration file syntax). Edit this new section as necessary.

Note:

It is easier and more reliable to run script configure-lxr.pl *with option* `--add` *which also generates directives to create the associated database.*

Global parameters define the global environment and are independent of the source trees, save for eventual common factoring of tree parameters.

Tree parameters are focused on characteristics of a tree.

CAUTION!

In the following parameter descriptions, the **comma** separating two consecutive parameters is not shown but **MUST** be written to avoid syntax errors.

5.4. Global parameters

These parameters are found in parameter group 0 after the *Global configuration section* header at the top of the file.

They are thought to have a rather universal or permanent value independent of a specific tree. This is not a marble-carved rule. You can deviate without harm. As explained above, you can override their value by repeating them in a tree section.

Adapt the values to reflect your configuration.

5.4.a. Auxiliary tools

Notes:

1. Parameters in this section are **mandatory** (meaning LXR will not execute if their value is not defined). This is checked by *genxref*. During operation, you may have a diagnostic in the error log of the web server.

2. All paths must be given in **OS absolute** form.

The first auxiliary tool is the search engine. Your choice is between ***glimpse*** and ***Swish-e*** through the following parameters. Use only OS absolute paths.

```
'glimpsebin'     => '/usr/local/bin/glimpse'
'glimpseindex'   => '/usr/local/bin/glimpseindex'
'glimpsedirbase' => '/home/myself/glimpse_databases
```

`'glimpsebin'` and `'glimpseindex'` are paths to the components of ***glimpse***. `'glimpsedirbase'` is a path to a directory where sub-directories named after `'virtroot'` will be created to store the search database for this tree. This directory must grant write permission to the user launching *genxref*.

```
'swishbin'       => '/usr/local/bin/swish-e'
'swishdirbase'   => '/home/myself/swish_databases
```

`'swishbin'` is the path to ***Swish-e*** binary. `'swishdirbase'` is a path to a directory where sub-directories named after `'virtroot'` will be created to store the search database for this tree. This directory must grant write permission to the user launching *genxref*.

CAUTION!

Only one of `'glimpsebin'` or `'swishbin'` must be defined. Comment out or erase the other.

Note:

The free-text search engines also uses parameters `'glimpsedir'` and `'swishdir'` set in the tree parameter groups if `'glimpsedirbase'` and `'swishdirbase'` are not defined.

The second tool is the language parser ***ctags***.

```
'ectagsbin'      => '/usr/bin/ctags'
'ectagsconf'     => '/absolute_LXR_root_dir/templates/Lang/ectags.conf'
```

You must also define several paths to allow LXR to achieve its job.

```
'tmpdir'         => '/tmp'
```

Directory for temporary files (**needs write access**, of course)

The third tool (optional) is the ***CVS*** utility.

```
'cvspath'  => '/bin:/usr/local/bin:/usr/bin:/usr/sbin'
```

Path for CVS module (may be omitted if CVS not used)

5.4.b. Computer DNS names

This is where you tell LXR the host names used in the URL to access your computer. This will form part of the key to identify the requested source-tree. The other part is found in tree parameter `'virtroot'`.

```
'host_names'     =>    [ 'http://mycomputer.example.com:8080'
                       , '//localhost'
                       , '//127.0.0.1'
                       , 'https://myPC.localdomain'
                       ]
```

The form of the parameter is:

protocol`://`*host-name*`:`*port*

protocol`:` is optional if it is `http`. `:`*port* can also be omitted if it is the standard port for the *protocol* (80 for `http`, 443 for `https`).

The web server hands over to LXR whatever URL was used, without translation nor DNS magic. If a numeric IP is used, you get a numeric host name. This is why you must list all the anticipated names to access your computer.

5.4.c. HTML parameters

LXR fills in HTML templates with variable data extracted from source documents and its internal database. These templates can be adapted to satisfy personal taste and needs. A certain number of them are provided in the standard release. The following parameters are used to map the internal feature names to the templates.

```
'htmlfatal'      => path to optional template for 'tree not found' error
'htmlhead'       => path to generic template for page header
'htmltail'       => path to template for page footer
'htmldir'        => path to template for directory display
'htmlident'      => path to template for ident inquiry form
'htmlsearch'     => path to template for search inquiry form
'htmlconfig'     => path to template for configuration display
```

These files are read only by LXR scripts. Their paths may be expressed as relative to the *LXR root directory* or as OS absolute paths. The first form is preferred when the templates are located in a subdirectory in LXR (like the release *templates*), the second when the templates reside in a directory unrelated to LXR.

The *templates/html* directory that ships with each release contains files with names *html-*.html* corresponding to parameter `'html*'` above. The exception is `'htmlsearch'` since the free-text search engines produce different results. You must then choose between *html-search-glipmse.html* and *html-search-swish.html*.

If you use these templates without modifications, you can write parameters like this:

```
'htmlhead'       => 'templates/html/html-head.html'
```

Note:

The standard release also provides an *html-head-btn.html* with *buttons-and-menus* interface which is a replacement for *html-head.html* using the more traditional *link* interface.

If you feel like changing some part of these default templates, it is highly recommended you copy them first and work on the copy.

You can use "specialised" version of `'htmlhead'` and `'htmltail'` for the different LXR modes by providing user-customised templates where `html` in the parameter names is replaced by `source`, `sourcedir`, `diff`, `ident`, `search` or `showconfig` as in `'sourcehead'`. These specialised parameters (and templates) are optional. LXR reverts to `'htmlhead'` and `'htmltail'` if they are not defined.

Note:

In the standard *lxr.conf* release, `htmlhead`, `sourcehead` and `sourcedirhead` all point to the same template.

To give maximum flexibility to source text display, LXR outputs its HTML without any embedded style. The HTML elements are `class`-attributed. The display decoration is achieved through a CSS style sheet defined by:

```
'stylesheet'    => HTML path to css style sheet
```

CAUTION!

This style sheet is served to your browser through HTML. This may involve copying this file to some location under your `DocumentRoot`. In the simple case where `DocumentRoot` is the *LXR root directory* and you are satisfied with the default style sheet, you may write:

```
'stylesheet'    => 'templates/lxr.css'
```

Other style sheets may optionally be defined. In a compatible browser, such as *Firefox*, the user can select the active style sheet from a menu.

```
'alternate_stylesheet' => [ list of css style sheet ]
```

Source trees may be written with character sets other than plain ol' ASCII. To have them correctly displayed, the character encoding must be transmitted to your browser in a `Content-Type` HTTP header for every HTML page with:

```
'encoding' => 'iso-8859-1'
```

Value may be any IANA-defined character set. Possible choices are, among others, `'iso-8859-15'` for Euro sign or `'utf-8'` for Unicode. This parameter is also frequently given in tree parameter groups.

If not defined, it defaults to the above character set to preserve backward compatibility.

The following parameters do not really configure LXR's HTML behaviour but they are passed as dynamic arguments in the URL with an underscore (_) preceding their name.

```
'diffleftwidth'  => '50'
```

Set it to the number of characters to display in the left column when comparing two files with *diff*.

If not defined, it defaults to the above width.

```
'identdefonly'   => '0'
```

Set it to `'1'` if you want to limit identifier search to definitions only. If not defined, it is equivalent to `'0'`. Values other than `'0'` or `'1'` lead to unspecified behaviour.

5.4.d. File management

The parameters of this subsection guide LXR in handling the files and their content. Modification of their value may have very adverse effects on LXR behaviour. As a fail-safe rule, **DO NOT** fiddle with them unless you ABSOLUTELY need it.

Source tree directories usually contain files besides the source text. These files may not be relevant for the purpose of displaying and studying the source tree. It is then worth to shadow them so that they do not appear in directory listing. Moreover, they can be excluded from indexing, which results in a performance improvement for this step. This is done by filtering files with a *regular expression* whenever LXR enters a directory:

```
'ignorefiles' => 'regular_expression'
```

The released value for the regular expression discards invisible (aka. “dot”) files, editor backups, binary files and core dumps:

```
'ignorefiles' => '^\\.|~$|\\.(o|a|orig)$|^core$'
```

It can be extended to fit your needs, preferentially overriding this global exclusion rule by a specific one in a tree section.

Note:

See the *Perl* manual for modification and pay special attention to the double backslashes (\) when you need one. This is a consequence of `eval`ing the string.

Some files in source trees may have graphic content. LXR will avoid processing them if the file extension matches its definition of a graphic file defined by:

```
'graphicfile' =>
     'bitmap|bmp|gif|icon?|jp2|jpe?g|pjpe?g|png|tiff?|xbm|xpm'
```

The value is a list of alternatives with *Perl* regular expression meta-characters. The outer parentheses are automatically provided by LXR. Pattern is case-insensitive and is constrained to match only at the end of the file name (both rules are hard-coded in LXR).

This allows to tag a file entry in the directory listing with an adequate default icon.

LXR also has a limited capacity to display graphic files. It relies on the source tree accessibility under `DocumentRoot` and the ability of your web browser to understand the file format. See parameter `'sourceaccess'` in the tree parameter group for the first condition. If one of the conditions is violated, the browser will display a text message instead (the `alt` attribute of the `<img>` element).

Note:

See the *Perl* manual for modification.

The default icons in directory listing are rather poor and ugly. You can replace them with your own icons with the following parameter:

```
'iconfolder'     => '/some_dir_containing_icons/'
```

CAUTION!

Do not forget the trailing slash!

The `'iconfolder'` value points to a directory under `DocumentRoot`. It is an **HTML-absolute path**. It contains graphic files sent as `<img>` elements to your browser.

You do not need to design your own icons. You may use the ones you like on your system. For instance, if you want to display the *Oxygen* theme icon, you only have to create a symbolic link to the appropriate directory (path below for *Fedora Linux*):

```
$ cd /DocumentRoot_directory      # often equal to LXR root directory
$ ln -s /usr/share/icons/oxygen/22x22/mimetypes/ small-icons-dir
```

and

```
'iconfolder'     => '/small-icons-dir/'
```

Which image is used is defined by an association list in parameter `'icons'` (small example for the *Oxygen* theme):

```
'icons' =>
     {     'c|pc'      => 'text-x-csrc.png'
     ,     'h|hh'      => 'text-x-chdr.png'
     ,     'c\+\+|cc|cpp|cxx'      => 'text-x-c++src.png'
     }
```

You list as many lines as needed. The key is a regular expression without the outer parentheses. Matching is case-insensitive and restricted to the file extension.

You must also provide default icons in case no key matched the file extension, notably for directories:

```
'graphicicon'    => 'image-x-generic.png'
'defaulticon'    => 'unknown.png'
'diricon'        => 'inode-directory.png'
'parenticon'     => 'go-up.png'
```

Note:

The feature involving `'iconfolder'` is optional. If `'iconfolder'` is not defined, LXR reverts to the historical display based on default *Apache* icons. If `'iconfolder'` is defined, you must define the other parameters otherwise LXR errors out.

When LXR is instructed to list a file, it passes its content to a *parser*. How to associate a parser with a file is defined by a set of rules located in a file pointed to by parameter `'filetypeconf'`:

```
'filetypeconf'   => 'templates/filetype.conf'
```

Note:

This file is used by the *Perl* scripts. Its path is relative to the *LXR root directory*. An OS-absolute form is all right also but less convenient if you move the LXR root directory:

```
'filetypeconf'   => '/path/to/templates/filetype.conf'
```

The parser role is to isolate what looks like an identifier. The candidate is then compared to the symbols in the data base. If found, its display will be enhanced in a special way. Remember that symbols are collected during the *genxref* step only and classified by ***ctags***. If you modify the source file afterwards or ***ctags*** did not label correctly the symbol, you will not see this enhancement.

The internal LXR generic parser is configured by a file pointed to by parameter `'genericconf'`:

```
'genericconf'    => 'lib/LXR/Lang/generic.conf'
```

Note:

This file is used by the *Perl* scripts. Its path is relative to the *LXR root directory*. An OS-absolute form is all right also but less convenient if you move the LXR root directory:

```
'genericconf'    => '/path/to/lib/LXR/Lang/generic.conf'
```

5.4.d.1. *Content of file filetype.conf*

In releases prior to 0.11, this file did not exist. Its content was inlined in lxr.conf. *But this content was enriched release after release and it grew to a size such that readability and navigation convenience was lost. It was then decided to move the content to an independent file. You may remove the* `'filetypeconf'` *parameter and replace it with* `'filetype'` *and* `'interpreters'` *parameters. LXR knows how to handle both cases.*

The list defined by parameter `'filetype'` associates a file extension with a language and a parser:

```
'filetype' => list of key/value pairs
```

Each pair in the list has the following form:

```
lxr_lang =>
     [ ctags_lang
     , file_selector
     , parser
     , tab hint
     ]
```

- *lxr_lang* is a unique language name. There is no reserved value, you can use whatever meaningful name you wish.
- *ctags_lang* is the language name used in file *generic.conf*. It is the same as the one used by ***ctags***. It creates the link between these two configurations. It is considered good practice to have the same name so as not to upset the reader.
- *file_selector* is a pattern (regular expression) to be matched for the files belonging to that language. It is not restricted to the extension only since some files may bear reserved names without extension.

- *parser* is the *Perl* module implementing the parser for that language (*Perl* syntax).
- *tab hint* may be omitted; in this case, an internal default of 8 is used. If LXR finds an *emacs* tab width specification in the first line of a source file, it will use it instead of the tab hint or default.

Example for a single language:

```
'filetype'       =>
     { 'C' =>
           [ 'C'                   # generic.conf name
           , '\.c$|\.h$'           # pattern with 2 targets
           , 'LXR::Lang::Generic'  # Perl module containing the parser
           , '4'                   # tab hint (optional)
           ]
     }
```

Source files may fail the `'filetype'` tests for various reasons (non standard file extension, operational convention like the LXR sources, …). In that case, the first line of the file is read. If it starts with `#!` or contains an *emacs* mode specification, it is considered a shell script and the "command" is used to associate the file to a `'filetype'` (which means this `'filetype'` exists). This is done with the help of the `'interpreters'` parameter:

```
'interpreters' => list of key/value pairs
```

where each pair is of the form

```
command => lxr_lang
```

Note:

command is used in a regular expression; you can then elaborate a rather intricate criteria if in need.

Example:

```
{ 'bash' => 'shell' , 'csh' => 'shell' }
```

and take care that `'filetype' => { 'shell' => ...}` exists.

5.4.e. "Common factor"

This is an optional section where parameters common to several trees can be listed. This allows to change these parameters in one location only and have them effective for all the trees (unless the parameter is overridden in a specific tree).

`'treeextract'` and database parameters are usually found here.

When you practice "black magic" to constraint several source-trees to share a single *LXR root directory*, a single instance of the scripts, LXR needs to know how to extract the part of the URL identifying a specific tree to give you meaningful information when something goes wrong. You do it through:

```
'treeextract' => '([^/]*)/[^/]*$'
```

This parameter provides a regular expression to extract the source tree name from the URL (considered up to the **script** component – for the technically minded, it is the string contained in environment variable `PATH_INFO`). The string captured in the first pair of parenthesis (known as `$1` in the substitution rules) becomes the value of the `$target` substitution marker for the `'htmlfatal'` template.

See the appropriate *Perl* manual for the details on pattern matching.

The example above gets `$target` from the second-to-last part of the script URL. Stated otherwise, it is the last component of `'virtroot'`. The example is the default regular expression used in case `'treeextract'` is not defined.

CAUTION!

`'treeextract'` must be consistent with web server configuration for multiple-trees operation. See chapter 7 Web server configuration, notably 7.2.d for ***Apache*** and 7.3.d for ***lighttpd***.

Overriding this parameter in tree sections is possible but needs careful design because of the complex impact on web-server configuration.

5.5. Tree parameters

The parameters below define the source tree and what processing LXR should apply to it. Some are very tricky; so, at first, you should stick with their default value, the better off if you are not familiar with *Perl*.

Duplicate this section if you serve several trees.

5.5.a. Server configuration

First of all, you must tell LXR how this tree is referenced from the web. This may include some URL demangling to separate strict LXR-service addressing from tree designation in case your server may manage several trees. A generic URL looks like:

protocol`://`*host_name*`:`*port*`/`some`/`directory`/`**script**`/`*path*`/`*for*`/`*file*`?`*var1*`=`*val1*`&`*var2*`=`*val2*

- *protocol*`://`*host_name*`:`*port*
  is covered by parameter `'host_names'` (see 5.4.b Computer DNS names)
- `/`some`/`directory`/`**script**
  is the HTML-absolute designation of the script to launch (*i.e. source*, *ident*, *diff, search* or *showconfig*). If your *LXR root directory* is the same as your `DocumentRoot`, you simply write `/`**script**.

  Note:

  `/`some`/`directory`/` may be more than a mere path; it may involve some web server rewriting magic to allow invoking a single LXR instance for several trees. In that case, there is no longer one-to-one correspondence between this HTML path and the OS path to the script.

- */path/for/script*
  has meaning only for the script. In the *source* case, it names the file to display.
- *?var1=val1&var2=val2*
  are optional arguments for the script.

To uniquely identify the source script, LXR uses the *protocol*`://`*host_name*`:`*port*`/`some/directory part as a key. *protocol*`://`*host_name*`:`*port* is described by parameter `'host_names'` in the global parameter group. The tree specific part is defined by parameter `'virtroot'`:

```
'virtroot' => '/some/directory'
```

Note:

In the simplest case (a single source-tree and a server dedicated to LXR only, *i.e.* `DocumentRoot` equal to the LXR root directory), `'virtroot'` is reduced to `'/'`.

The string formed by concatenation of `'host_names'` and `'virtroot'` is used to match the initial part of the URL. The match is considered successful if the value of this string is an exact **prefix** (case sensitive) of the browser request.

The value of this parameter is used to build the requests associated to the different links in the LXR-synthesised pages. It is also present in the `<base>` tag of the `<head>` section of every page.

Note:

An alternate identification of the source-tree is kept for backward compatibility with former versions of LXR. It is deprecated because it is more error-prone and requires more user checks due to the duplication of the virtual root. The new identification method with `'host_names'` and `'virtroot'` is strongly recommended.

The older identification is based on parameters `'baseurl'`, `'baseurl_aliases'` and `'virtroot'`. Of course, `'virtroot'` has the same semantics as above.

```
'baseurl'  => 'http://site.url.example/lxr'
```

`'baseurl'` is the primary URL from which LXR is served. You are strictly limited to `http:`. Due to the present programming, you cannot use `https:`.

CAUTION!

The full URL from the browser request (including the port if it is different from :80) is used in the selection of the source-tree. You must consequently also write here the full URL starting with the protocol. The match is considered successful if the value of the `'baseurl'` is an exact **prefix** (case sensitive) of the browser request.

In case your server can be reached under different names (*e.g.* with the above URL from the Internet and with `localhost` on your computer or `site.url` for short on your local network, you can identify these aliases with the following key `'baseurl_aliases'` (incurring the same constraints as `'baseurl'`).

Of course, if you are using a numeric IP address instead of a host name, *e.g.* on a small local network, you must provide either `'baseurl'` or one `'baseurl_aliases'` with this numeric

IP address.

```
'baseurl_aliases' =>
          [ 'http://localhost/lxr'      # local computer
          , 'http://site.url/lxr'       # local network
          , 'http://another.url.example:43210/lxr' # with unusual port
          ]
```

`'baseurl'` and/or `'baseurl_aliases'` select the tree parameter-group to activate. More on that later to uncover some tricks to have several trees under (apparently) the same URL and a single directory for LXR.

CAUTION!

Since the links built by LXR does not use the host part of the request URL, you'd better care that the `'virtroot'` is the same among `'baseurl'` and all the `'baseurl_aliases'`, otherwise LXR will die when presented a synthesised request containing `'virtroot'` from an inconsistent alias.

Look-up for a tree ends on first match. If no match is found among all tree parameter groups, LXR displays the `'htmlfatal'` page.

The following two parameters are not really related to server configuration, but they are so general that it is better to list them early in a description of a tree in order not to forget them.

```
'caption'        => string
'shortcaption'   => very_short_string
```

`'caption'` is a title displayed in the header area of every page. Be descriptive, so every reader will know at once the purpose of the software contained in the source-tree.

`'shortcaption'` is used only when you have several trees and you want a “button” to quickly jump to the root of the tree. If `'shortcaption'` is present, a hyperlink with this name will be created. If there is no `'shortcaption'`, you must type the URL of your tree in the address bar to access your tree. You can then offer public and “private[31]” trees through coding a `'shortcaption'` or intentionally omitting it.

5.5.b. Tree location

Mandatory parameter `'sourceroot'` defines both the method to access the tree and its location.

This location is an OS-absolute path. Check that it is “*world readable*” (remember that LXR is executed from the HTTP server under its own user-id).

5.5.b.1. *Plain files*

```
'sourceroot' => 'directory_with_version_subdirectories'
```

[31] Do not take the word *private* too literally here. The feature offers no security. Only, you must know the tree URL instead of clicking on a speed link. Your tree is as public as your web server.

5.5.b.2. CVS repository

```
'sourceroot' => 'cvs:path_to_CVS_directory'
```

The *CVS* directory is real directory on your computer. Remote access does not work.

LXR offers an experimental feature to access the *CVS* history of the currently displayed file. This requires defining two parameters:

```
'cvswebprefix'   => string
'cvswebpostfix'  => string
```

The strings are concatenated in order `'cvswebprefix'`, file name, `'cvswebpostfix'` as a link under the “button” *View CVS Log*. Together they constitute a query to a *CVS* browser (*cvsweb*, *viewcvs* or other). This feature is rather rudimentary: the only host name used is the one coded in `'cvswebprefix'`. Presently, there is no mechanism like `'host_names'` above to access the *CVS* browser under different names (own computer, local network or web). Moreover, this service could be hosted in a different server than the LXR server.

The strings cannot contain variable substitution.

Examples:

```
, 'cvswebprefix'  => 'http://cvs.myhost.com/cgi-bin/cvsweb.cgi'
, 'cvswebpostfix' => '?cvsroot=rootname'
```

```
, 'cvswebprefix'  => 'http://cvs.myhost.com/cgi-bin/viewcvs.cgi/myroot'
, 'cvswebpostfix' => ''
```

5.5.b.3. GIT repository

```
'sourceroot' => 'git:path_to_Git_directory'
```

The *Git* directory hosts the *objects*, *refs*, *index*, … directories. *Git* access needs to pass parameters when initializing the library (see `'sourceparams'` below).

5.5.b.4. Subversion repository

```
'sourceroot' => 'svn:path_to_Subversion_directory'
```

The *Subversion* directory hosts the database for the source-tree.

Note:

This preliminary implementation limits access to local directories. Do not try to define something like `svn:http://svn.remote.net/project-A`. The initialisation code is not prepared to handle the `http:` prefix.

5.5.b.5. Mercurial repository

```
'sourceroot' => 'hg:path_to_Mercurial_directory'
```

The *Mercurial* directory hosts the database for the source-tree.

Note:
This preliminary implementation limits access to local directories.

5.5.b.6. BitKeeper repository

```
'sourceroot' => 'bk:path_to_BK_repository'
```

BitKeeper[32] access needs to define an auxiliary working directory when initializing the library (see `'sourceparams'` below).

CAUTION!
This access method has not been tested for a long time. You are on your own. The present maintainer does not even know if it works or not. Development stopped at the end of 2005.

5.5.b.7. Optional initialisation parameters

Some access methods need extra parameters when initialising their library. Parameters are passed through `'sourceparams'`:

```
'sourceparams' => list of key/value pairs
```

See chapter 10 Using LXR with SCMs for the parameters related to the chosen access method.

5.5.c. Other parameters

```
'sourcerootname' => 'text to display for the source root'
```

All file paths displayed in the header area of every HTML page are relative to `'sourceroot'`. The text provides a reminder for this root. It can be a simple string (such as `'/'` or even `''` but in the latter case you can then no longer navigate directly to root since you are unable to click on an invisible link). It can also contain variable substitution (one declared in the `'variables'` list only) like `'$v'` to have the version name (or more precisely the name of the directory containing the version).

```
'sourceaccess' => symbolic link inside LXR directory
```

This optional parameter is needed only if you want LXR to display graphic files. It points to the same directory as `'sourceroot'` but it is relative to `DocumentRoot`. It cannot contain any `..` (parent directory) partial path (which is removed for security reasons), thus the need for a symbolic link from `DocumentRoot` to the `'sourceroot'` directory.

Example:

[32] *BitKeeper* is a **proprietary software**. Since April 2005, there is no more free version for community developed open software. The interface developed for LXR is left as it was at that date in case someone needs it.

```
, 'sourceroot'   => '/home/source/tree'
, 'sourceaccess' => 'bypasslink'
```

if you have defined:

```
$ cd /DocumentRoot_directory      # often equal to LXR root directory
$ ln -s /home/source/tree bypasslink
```

The `'sourceaccess'` link is necessary to access files from HTML which is rooted at `DocumentRoot` in the web server. The scripts, written in *Perl*, have access to any "*world readable*" file through an absolute path. On the other hand, the web server is granted access only to its `DocumentRoot` hierarchy through a relative path. LXR adds a further security restriction that this path cannot contain any `/../` part. When issuing `<img` …> HTML tags, the image is retrieved under the server paradigm and must reside somewhere under `DocumentRoot`.

Note:

Since you activate this feature only when you need it, no care is taken to make sure that the parameter is defined when issuing an `<img` …> HTML tag. If it is not, the tag link will be faulty and you will get a "filename *cannot be displayed from this browser*" message as if the browser did not support the graphics format, whereas the link does not contain the valid file name.

CAUTION!

There is presently no way to display graphic files when the source tree is managed by an SCM because there is no real file.

5.5.d. Version selection

The exact version of a file you display in LXR is defined through a set of so-called "variables". Since they offer a choice, they will appear on the screen as a set of clickable fields in the top of the page. The LXR script (*diff*, *ident*, *search* or *source*) uses their values if it has been programmed to do so.

```
'variables'     => list of key/value pairs
```

A variable definition has the form:

```
'variable' =>
           { 'name'    => text to display
           , 'when'    => valid context for variable
           , 'range'   => array containing the set of accepted values
           , 'default' => default value
           }
```

A *variable* name may contain alphanumeric characters (`a` through `z`, `A` through `Z`, `0` through `9`) and underscores (_). The first character must not be an underscore.

The `'when'` key is optional. If omitted, the variable is always displayed in the version selection area. The value is a *Perl* boolean expression whose effect is to show (if *true*) or hide (if *false*) the variable setting buttons or menu. The expression may contain *strings* or reference variables by prefixing their name with a dollar sign ($).

```
'when' => '"$a" eq "arm"'
```

CAUTION!

Use *single quotes* (') to delimit the value and surround the operator arguments with *double quotes* (") in the expression. Odd as it may seem, double quotes are **mandatory** around variable names because their value is substituted **before** *Perl* evaluation: the expression is effectively computed against string constants, which require delimiters.

The `'when'` key is useful to hide irrelevant choices under defined circumstances. For instance, the kernel source-tree allows to choose between different architectures which, in turn, offer sub-architectures. The choice of these sub-architectures makes sense only when their primary architecture is active.

Evaluation of `'range'` results in an array of strings. These strings are the names of sub-directories in `'sourceroot'`. In its simplest form, this key is expressed as:

```
'range'    => [ "1.0" "1.1" "2.0" ]
```

or, more user-friendly:

```
'range'    => [ qw(1.0 1.1 2.0) ]
```

Advanced users (*Perl* gurus) might initialise `'range'` dynamically, like:

```
'range'    => [ readfile('/OS/absolute/path/versions_file') ]
```

`readfile` is an LXR-provided function to read any file in the system. Text file `versions_file` (or any other name you like) is best located in the source tree. It consists of the names of the versions, separated by spaces or newlines, without quote characters (unless the name contains spaces). Reading from a file reduces the necessity to update *lxr.conf* but does not remove any *genxref* processing.

You can even write a function to compute the array:

```
'range' => sub
     { return    grep {/(release|head)/}
            ($files->allreleases($LXR::Common::pathname)
            ,$files->allrevisions($LXR::Common::pathname)
            )
     }
```

This example is frequently used to extract all CVS tags and keep only those containing *head* or *release*.

The `'default'` key is optional. It defines the initial value displayed for the variable. If omitted, the variable defaults to the first value in `'range'`.

CAUTION!

`'default'` is mandatory if `'range'` is a function and your tree is stored in an SCM repository because `'range'` is no longer a real array.

You can define as many variables as you want. Unless you modify the scripts, do not expect any change of behaviour due to these new variables.

Variable `'v'` is **mandatory**. It is the main vehicle for choosing the version. Its name is hard-coded in all the scripts.

```
'v' =>
     { 'name'    => 'Version'
     , 'range'   => [qw(1.0.0 1.0.2 1.1)]      # example
     , 'default' => '1.1'    # if omitted, takes 1.0.0 from 'range'
     }
```

Some source-trees (such as the Linux kernel tree) have a variable `'a'`. It is not managed by the LXR scripts directly but may be used in `'maps'` rules to do wonderful things on links to files.

```
'a' => { 'name => 'Architecture', 'range' => … }
```

You can freely define any other other variables if you have a use for them in `'maps'`.

5.5.e. Exclude directories (Subdirectory)

```
'ignoredirs' => array of partial names for directories
```

These directories are not displayed nor indexed thus speeding up the process. They might be the directories where your Integrated Development Environment stores the intermediate forms of the sources or the binary objects.

A directory is ignored if **any** part of its path is **equal** to one of the strings. Stating it otherwise, the test is case sensitive, cannot contain a path separator (/ on *Unix*-like OS's) to designate a sub-directory[33], is not level specific in the hierarchy of directories.

Example:

```
'ignoredirs' => [ 'CVSROOT'
                , 'build'
                , 'CmakeTemp'
                ]
```

Notes:

If for any reason, LXR processes `DocumentRoot` and this directory contains `'sourceaccess'` links, add them in the `'ignoredirs'` list to avoid funny things from happening.

“Dot” directories (such as `.git`) are always ignored; this is an internal hard-coded rule which cannot be overridden by a carefully crafted `'ignoredirs'` parameter.

Tip:

Technically, `'ignoredirs'` is applied only on the **last** segment of the path when traversing the source tree. Consequently, if you manually provide an URL to your browser (you do not traverse the tree, you jump straight to the final node), you can bypass the exclusion rule and

[33] The “path separator” will be considered as part of the name as if it had been escaped.

display a file or subdirectory from the excluded directory. However, no highlighting of proper procedures or variables will be exhibited since indexing was skipped.

`'ignoredirs'` excludes all directories sharing the advertised fragment. For instance, if `include` is listed, `include/specific` cannot be kept. In case you need very detailed control on the excluded directories, parameter `'filterdirs'` acts on the full path:

```
'filterdirs'  => array of regexp for directories
```

Rules:

- **DO NOT USE IT** unless there is no simpler way with other parameters.
- Directory names are NOT terminated by a separator (`/`). If you want to match exactly your pattern (*i.e.* not as a prefix), use the end anchor `$` to terminate the pattern.
- Paths always begin with a separator (`/`). To match at the beginning of the path, force the first separator with the start anchor `^` as `^/`.
- Regular expressions have a cumulative effect: what has not been excluded by the first is filtered by the second, *etc*.
- Use regular expressions defined with *Perl* operator `qr//`. Beware of the path separator!
- Read carefully the *Perl* manual since first-shot regular expressions can very easily end up in excluding everything.

Example:

```
'filterdirs'  => [ qr!/build/CMakeFiles$!
                 , qr!^/include\b!
                 , qr{/include/(?!specific)$}
                 ]
```

The first pattern excludes all subdirectories ending in */build/CMakeFiles*.

The second one excludes directory */include* at the root of the tree, but also */include%other* because of the `\b` delimiter. To strictly exclude only */include*, use the `$` anchor instead.

The third one excludes any directory in */include* which is different from */include/specific*. Note that if this subdirectory is at the root of the tree, it cannot be kept because of the second pattern.

Tip:

Like `'ignoredirs'`, `'filterdirs'` is applied only when traversing the source tree. Consequently, if you manually provide an URL to your browser (you do not traverse the tree, you jump straight to the final node), you can bypass the exclusion rule and display a file or subdirectory from the excluded directory. However, no highlighting of proper procedures or variables will be exhibited since indexing was skipped.

IMPORTANT NOTICE:

Use of parameter `'filterdirs'` is strongly discouraged because of performance consideration, mostly on huge projects such as the Linux kernel. Depending on the number of regular expressions and their complexity, application of this parameter may have adverse effect on indexing time.

5.5.f. Exclude files (Subdirectory)

```
'ignorefiles' => regular expression
```

This exclusion rule (for file names) complement `'ignoredirs'` to provide fine-grained control about what is indexed and displayed.

Reminder:

A tree-section parameter overrides the corresponding global parameter. Consequently, you must copy the global rule in the tree-specific rule or else the previously discarded files reappear.

Tip:

Technically, `'ignorefiles'` is applied only on the **last** segment of the path when traversing the source tree. Consequently, if you manually provide an URL to your browser (you do not traverse the tree, you jump straight to the final node), you can bypass the exclusion rule and display an excluded file. However, no highlighting of proper procedures or variables will be exhibited since indexing was skipped.

`'ignorefiles'` excludes all files matching the pattern in any directory. For instance, if `bad` is listed, `specific/bad` cannot be kept. In case you need very detailed control on the excluded files, parameter `'filterfiles'` acts on the full path:

```
'filterfiles'  => array of regexp for files
```

Rules:

- **DO NOT USE IT** unless there is no simpler way with other parameters.
- If you want to match the *filename* fragment of the path, use the end anchor `$` to terminate the pattern.
- Paths always begin with a separator (`/`). To match at the beginning of the path, force the first separator with the start anchor `^` as `^/`.
- Regular expressions have a cumulative effect: what has not been excluded by the first is filtered by the second, *etc*.
- Use regular expressions defined with *Perl* operator `qr//`. Beware of the path separator!
- Read carefully the *Perl* manual since first-shot regular expressions can very easily end up in excluding everything or have undesirable side-effects.

Note:

`'filterfiles'` is targeted towards specific files; it is a tree-specific parameter by essence. It makes little sense to specify it in the global section of the configuration file.

Example:

```
'filterfiles' => [ qr!/specific/asm-.*$!
                 , qr!/m68k/(.*/)?[^/]*\.S$!
                 ]
```

The first pattern excludes all files starting with *asm-* in any */specific* directory.

The second one excludes all `.S` assembler files in the */m68k* subtree.

Tip:

Like `'ignorefiles'`, `'filterfiles'` is applied only when traversing the source tree. Consequently, if you manually provide an URL to your browser (you do not traverse the tree, you jump straight to the final node), you can bypass the exclusion rule and display a file otherwise excluded. However, no highlighting of proper procedures or variables will be exhibited since indexing was skipped.

IMPORTANT NOTICE:

Use of parameter `'filterfiles'` is strongly discouraged because of performance consideration, mostly on huge projects such as the Linux kernel. Depending on the number of regular expressions and their complexity, application of this parameter may have even worse effect on indexing time than `'filterdirs'`.

5.5.g. Include directories (Subdirectory)

The following parameters allow LXR to build direct links to files mentioned in “include” sentences (such as C `#include` or *Perl* `use`). For the link to be successfully built, the “included” file must exist, be located in an accessible subdirectory and have been indexed by *genxref*. If any of these conditions is not satisfied, the file name will be treated as a string, a symbol, … according to its appearance.

When parsing an include sentence, LXR prefixes the file name found in the text with each directory from the optional `'incprefix'` list in turn, starting implicitly with the current directory. The resulting file name is subject to transformations from `'maps'` before determining if the file really exists. The process succeeds when a match is found or fails at the end of the list.

```
'incprefix' => [ directories where to look for included files ]
```

Directory names are relative to `'sourceroot'`.

If `'incprefix'` is omitted, it defaults to the current directory only.

```
'maps' => [ reg-exp => replacement , … ]
```

CAUTION!

Though => is used as separator between *reg-exp* and *replacement*, the whole list of rules is contained in an **array**, not a *Perl* **hash**. This change has been introduced in release 1.0 to allow a precise order for `'maps'` rule application.

This optional parameter is an attempt to mimic compiler behaviour where include directory locations are given by command line options. It contains a list of substitution rules written as pairs pattern to match/replacement (in the syntax above, => is a synonym for comma (`,`) to insist upon the fact that the array must contain an even number of strings). All rules are applied sequentially in the given order and their effect is cumulative.

replacement may contain *variable* substitution requests in the form $*variable*. This symbol is replaced by the current value of the designated variable. If there is an ambiguity risk with what follows the request, use form ${*variable*} to strictly delimit the name.

See the *Perl* manual for regular expression writing.

Example taken from the kernel introducing variable a:

```
, 'incprefix' => [ '/include', '/arch/%-ARCH-%/include' ]
, 'maps' =>      [ '\/arch\/%-ARCH-%\/' => '/arch/$a/',
                 ]
```

With #include 'inc.h' directive, the list of include prefix produces successively the following names *current_directory*/inc.h, /include/inc.h and /arch/%-ARCH-%/include/inc.h which are then transformed. The first and second ones are used as is since no 'maps' rule matches. The third will give /arch/ppc/include/inc.h if the current value of variable a is ppc.

Note:

%-ARCH-% is used as a temporary surrogate for the effective architecture name inserted by the 'maps' rule. If %-ARCH-% happens to be a real directory, change the name to something not existing otherwise there may exist cases in which the 'maps' rule will be wrongly applied.

This is where you can find a use for custom variables without modifying LXR.

5.5.h. Auxiliary data storage

LXR needs to store its internal data somewhere for retrieval. Your choice is between *MySQL*, *Oracle*, *PostgreSQL* and *SQLite* database engines.

```
'dbname' => identifier of the database
'dbuser' => username for the database
'dbpass' => user password for the database
```

These **mandatory** parameters tell LXR which database to use and how to connect to it.

For *MySQL*, the format is	dbi:mysql:dbname=*data_base_name*
For *PostgreSQL*, it is	dbi:Pg:dbname=*data_base_name*;host=*localhost*[34]
For *SQLite*, it is	dbi:SQLite:dbname=*file_name*
For *Oracle*, it is	dbi:Oracle:host=*localhost*;sid=DEVMMS;port=1521

CAUTION!

The driver name after dbi: is case-sensitive and must be written as above.

```
'dbprefix' => 'lxr_'
```

34 host=localhost forces connection to the database through TCP socket (seemingly more reliable than default).

This optional parameter defines the prefix for the tables in the database. It allows to store separate data for different trees in a single database. The example above displays the default prefix.

This parameter is relevant mainly for *Oracle* which apparently serves a single database. It is also useful in multiple-trees context.

CAUTION!

Changing the prefix is best done with *initdb-config.pl* script.

The search engines can store their index files in a tree-specific location. This avoids mixing symbols from different trees. Unless you defined `'`*xxx*`dirbase'`, you must create a read/write directory for the search engine and tell LXR its name with:

```
'glimpsedir' => directory
'swishdir'   => directory
```

Note:

Having both specified causes no harm.

Reminder:

From release 1.0 on, it is easier to define `'glimpsedirbase'` and `'swishdirbase'` in the global section (auxiliary tools) without defining `'glimpsedir'` nor `'swishdir'` in a tree-specific section.

6 Generic parser configuration file

This chapters covers the content of the embedded parser configuration file. Its content describes the essential features, from LXR point of view, of each supported language.

The generic parser contained in *Generic.pm* can parse a wide variety of languages, at least for the purpose of cross-referencing symbols. Its sole purpose is to split the input file into sequences representative of categories of the language. These categories are rather coarse-grained but a sequence contains tokens pertaining to a single category.

Next, finding the names (*i.e.* strings) inside a sequence is a simple matter of comparing the strings to a dictionary.

How *Generic.pm* is driven is a matter of tuning *generic.conf*. The path to this file is stored in parameter `'genericconf'` of *lxr.conf*.

6.1. Parser configuration file syntax

The parser configuration file contains a list of **key/value pairs**.

Note:
If you are familiar with *Perl*, it is an anonymous *hash*. It is read in at initialization time and evaluated. What this means is: you get all the power of *Perl* and all its idiosyncrasies.

Comments can be added to the file to give explanations or take note of modifications. A comment starts with a pound sign (#) followed by a blank (space or tab) and extends to the end of line.

Examples:

```
# This a comment starting in col. 1
'relevant => 'data' # This comment follows meaningful configuration data
```

The list of key/value pairs (with a *hash* syntax for the technically minded) is:

- opened by a left-curly-brace {
- separated by a comma ,
- closed by a right-curly-brace }

```
{ key/value-0
, key/value-1
 ...
, key/value-m
}
```

In these pairs, the **key** is a string (delimited with single-quotes `'` to avoid funny things from happening when evaluating the parameter). The **value** may be a (single-quote delimited) **string**, an

array or even another **list of key/value pairs**. Each pair contains:

- the quote-delimited key
- the operator => (equals sign followed by right-angle-bracket, also named *greater than*, without any intervening space)
- the value as one of:
  - a single-quote delimited string for single value parameters
  - an array of values opened with a left-square-bracket [separated by a comma and closed with a right-square-bracket]
  - a list of key/value pairs (see above)

Example:

```
{ 'number_0' => '0'
, 'array' =>
           [ 'elem0'
           , 'elem1'
           , 'elem2'
           ]
, 'secondary_hash' =>
           { 'sec_key_a' => 'a'
           , 'sec_key_b' => 'b'
           }
}
```

Note:

This gets evaluated by *Perl*. Thus, there is a more elegant and less error-prone way of writing the array above by using the *Perl* function qw() (meaning *quote word*):

```
, 'array' => [ qw(elem0 elem1 elem2) ]
```

We are thus spared the trouble of writing the quotes and forgetting one. qw() splits its argument list at white space.

The configuration file has the following structure:

```
{ 'ectagsopts'         => [ list of options ]
, 'eclangnamemapping'  => { list of key/value pairs }
, 'langmap'            => { list of key/value pairs }
}
```

The first two parameters tune *ectags* behaviour. The last one configures the parser for a language.

6.2. *ectags parameters*

When asked to parse the tree files by *genxref*, LXR launches ***ectags*** with the command:

```
ectagsbin ectagsopts –excmd=number --language-force=lg -f - path to file
```

`ectagsbin` comes from *lxr.conf*. `lg` is the language forced name (see below). `ectagsopts` is defined as:

```
'ectagsopts' =>
           [ "--options=" . $config->ectagsconf
           , "--c-types=+plx"
           , "--eiffel-types=+l"
           , "--fortran-types=+L"
           ]
```

Any needed option can be defined. The above example is the default configuration. All options from this array are concatenated before being written to the command.

Note:

You see on the first line an example of *Perl* eval'ing the whole configuration: global configuration parameter `'ectagsconf'` is concatenated to string `"--options"` without pain.

See the ***ectags*** manual if you need more information.

```
'eclangnamemapping' =>
           { 'C'      => 'c'
           , 'C++'    => 'c++'
           , 'Python' => 'python'
           , 'SQL'    => 'SQL2'
           , 'shell'  => 'sh'
           }
```

This “table” defines the mapping between the LXR language names in the *key* and those used by *ectags* in the *value* column when option `--language-force` is used. That is, always, since LXR always launches *ectags* with this option to make sure *ectags* will not infer a different language name. The table is rather short because it contains only the differences between the two environments. When a language name is not in the table, it is used as is.

6.3. Language descriptions

`'langmap'` is a table where each managed language has an entry. Each entry is key/value pair. The key is the language name. It must be identical to the value of first sub-parameter of `'filetype`' in *filetype.conf* (or *lxr.conf* if parameter `'filetypeconf'` is not used) from which it is referenced.

```
'langmap' =>
     { 'C'       => { … }
     , 'C++'     => { … }
     , 'Perl'    => { … }
     , 'shell'   => { … }
     …
     }
```

The entry for a language defines the list of reserved keywords, how to “parse” a file, the names of symbol classes and an internal numeric id:

```
{ 'identdef'     => string
, 'reserved'     => [ list of reserved keywords ]
, 'flags'        => [ list of flags ]
, 'spec'         => [ list of parsing rules ]
, 'include'      => { list of key/value pairs }
, 'typemap'      => { list of key/value pairs }
, 'langid'       => 'numeric id'
}
```

Let us have a look to each one.

6.3.a. Language tagging

```
'langid' => numeric string
```

This gives a numeric id to the language for indexing purpose. Every language must have a unique `'langid'`. This allows to discriminate two variables or functions with the same “names” in two languages so that they appear as two different entities.

What happens if `'langid'` is not unique has not been tested.

6.3.b. Identifier detection

```
'identdef' => pattern
```

This partial pattern defines the lexical form of identifiers **and** reserved words. A default pattern of `'[-\w~\#][\w]*'` is built inside LXR to cover as many languages as possible. It has the adverse effect of capturing part of expressions (if operators - and ~ precede an identifier without intervening space) which would prevent the bare identifier from being recognised in the database. You must then design a pattern to match the strict language definition.

Note:

It is said **partial** because the LXR engine surrounds this string with ancillary sub-patterns to “navigate” inside the source text. In particular, the `'identdef'` pattern is immediately followed by `\b` (symbol boundary). Do not bother to enclose the whole pattern in parentheses, it will be done for you.

Examples:

```
'identdef' => '([\w~]|\#\s*)[\w]*'       # C and C++
'identdef' => '[\w]+'                    # Pascal
'identdef' => '[-\w][\w]*'               # Perl
```

The first example captures strings formed of letters, digits or underscores optionally preceded by a tilde (for destructors) or #directives with optional spaces between the pound sign and the name. *Perl* identifiers may have an initial dash.

6.3.c. Reserved keywords

```
'reserved' => [ list of reserved names ]
```

These names will not be considered as identifiers and will be tagged as reserved words. They can thus be displayed with nice decoration if you feel like playing with *lxr.css*. The pattern for `'identdef'` must be able to describe them. Any form not captured by `'identdef'` will not be recognised.

Note:

Strictly speaking, it is useless to list the *include* keywords because they are captured in the independent category `'include'` (see below). Anyway, it is harmless to include them in the list and that avoids questions about listing them or not and puts aside the risk of omitting an otherwise needed keyword in another context.

May be empty, meaning no reserved word.

Example:

```
'reserved' => [ 'for', 'do', 'done', 'case' ]
```

CAUTION!

String comparison is case-sensitive unless modified by flags.

6.3.d. Languages attributes

```
'flags' => [ list of flags ]
```

The flags change the standard behaviour of the language parser. Adding a flag in the list puts it in the "on" state; omitting it is equivalent to the "off" state.

The implemented flags are:

- `'case_insensitive'` causes comparisons to be done independent of case (look-up in symbol dictionary, keywords).

Reminder:

Regular expressions for `'identdef'` and `'spec'` must capture both case variants.

6.3.e. Source file fragment categories

```
'spec' => [ list of key/value pairs ]
```

Each element of the list defines a rule for breaking the file into fragments belonging to a category. The rule consists of:

```
{ 'category' => [ list_of_patterns ] }
```

1. the name of the category,
2. a pattern (regular expression) matching the beginning of the fragment,
3. a pattern (regular expression) matching the end of the fragment,

4. an optional pattern (regular expression) matching any token which should be kept inside this fragment, though it could be identified as the end delimiter without this rule.

As of version 0.9.8, recognized categories are `'atom'`, `'comment'`, `'include'` and `'string'`. Category `'code'` is reserved for future extension.

Note:
`'atom'` is very specific and should be defined only when no other solution is feasible with the present parser. In versions prior 0.9.8, it was used to prevent leaving the current parsing state when matching construction was encountered. The parser has been restructured to have a more reliable and flexible behaviour. The feature will change without notice in relation with `'code'` category.

A category is normally described by at least two patterns: one for the “start” delimiter, one for the “end” delimiter. It contains anything between the delimiters. Sometimes, it is difficult to design neatly separated delimiter pattern because they are interdependent. In that case, the category may be described by a single pattern capturing the whole fragment; there is no “end” nor “stay' pattern.

The parser follows the following algorithm:

- The state-machine starts in the *no-category* state. It leaves this state on any “start” delimiter. At this point, the already parsed fragment is passed for generic processing (tagging reserved words and identifiers by *processcode*).
- When a “start” delimiter is found, the state-machine enters the *category* state. It remains in this state even if *category* “stay” tokens (described by the optional pattern 4 above) are encountered until its *category* “end” delimiter is found. The parsed fragment, including the delimiters, is passed to the *category* processing.

  The state-machine is then restarted in the *no-category* state.

Dispatching is based solely on the “start” delimiters. But, as is common in language grammars, you may meet ambiguities (some delimiters may be prefix of others). To solve this difficulty, you should sort your categories with the most specific and longest “start” delimiters first, followed by more and more generic and shorter delimiters. The patterns are used in the order in which they appear in `'spec'`.

CAUTION!
Remember that *generic.conf* is eval'ed by *Perl*. Consequently, if you need to backslash-escape characters in the pattern, backslash \ must be repeated twice.

The categories are defined are as:

- `'comment'` is used to capture comments in the file.

  It will be tagged as such and its content will not be scanned for reserved words nor identifiers.
- `'include'` is used to capture references to other files.

  It will be processed with the `'include'` specification before being manipulated with *lxr.conf* parameters `'incprefix'` and `'maps'` to associate the file name with an hyperlink to the file.

- `'string'` is used to capture strings.

  They will be tagged as such and the content will not be scanned for reserved words nor identifiers.

- `'atom'` is used to “lock” the state-machine in the *no-category* case if that effect cannot be obtained otherwise. It has no “start”, nor “stop” delimiters. To insist upon its difference, its syntax is not the same as the others. It contains a single pattern, not an array.

  ```
  'atom' => pattern
  ```

  A more precise and controlled effect is obtained with pattern 4 in the other categories. Stated otherwise, you do not release control from the other categories until you know for sure you must leave them.

  Example:

  ```
  'spec' =>  [ { 'comment' => [ '/\*', '\*/' ] } # See note
             , { 'comment' => [ '//',  "\$"  ] ]
             , { 'string'  => [ '"',   '"', '\\\\.' ] }
             , { 'string'  => [ "'",   "'", "\\\\." ] }
             , { 'include' => [ '#\s*include', "\$" ] }
             ]
  ```

  Note:

  \ is not repeated, because we want * (and not the pattern operator), which is what we get after *eval*. A frequent pattern in the standard file is `'\\\\.'`; after *eval*, we get `'\\.'` which means “match a \, then any character”, quite tricky to get it right!

  The `'string'` definitions exhibit how to stay within a string when encountering `\"`. Without the “stay” pattern, the string `"A string with \" inside"` would have been split as a string `"A string with \"`, a code fragment containing a variable `inside` and the beginning of a new string with the last `"`. The rest of the source text would have been out of sync.

  IMPORTANT NOTICE!

  In case you need to describe alternatives in some patterns, use `(?:…|…)` instead of `(…|…)`. The latter form “captures” the alternative item inside its parentheses and breaks the internal sequencing of the source file. In the former form, the parentheses are made “transparent” by `?:` which does not cause spurious capture. For an example, see the HTML description in file *generic.conf*.

  For some languages, “standard” parentheses must be used. This has required modification to the parser to make this sub-pattern independent from the pattern in which it is incorporated. The patch brought better parsing correctness but caused incompatibility with Perl version 5.8 because the needed feature was introduced in Perl 5.10.

Do not be overconfident with the “stay” pattern. There are still shortcomings because pattern-matching will never compete with a true finite state automaton.

Sometimes, categories need to be anchored at the beginning of lines. But since the LXR parser considers the source file as a simple stream of characters, the start of the internal buffer does not coincide with the start of line. A specific processing is needed to "understand" the ^ anchor at the beginning of the pattern. Due to the simplistic nature of this processing, there can be only one anchor as the first character in the pattern. If the pattern describe alternatives, you must then "common factor" the anchor.

Example for *Fortran 77* comment:

F77 comments are characterised by `C` or `*` in column 1 (beginning of line).

```
'spec' =>  [ { 'comment' => [ '(^C|^\*)', '\$' ] }   # won't work
           , { 'comment' => [ '^(C|\*)',  '\$' ] }   # correct
           ]
```

In the first definition, the anchors are not in first position and are not detected by pre-processing. The second definition will be successfully transformed to allow comment parsing.

6.3.f. Describing include syntax

```
'include' =>     { 'directive' => directive_syntax
                 , 'separator' => string
                 , 'pre'[35]     => [ substitution_pair ]
                 , 'global'    => [ substitution_pair ]
                 , 'post'[36]    => [ substitution_pair ]
                 }
```

This group is used against an `'include'` fragment to break it into components (instruction name, target file, *etc.*) and transform the language form for the target file into an OS path when the default processing is not fit.

`'directive'` must provide the following 5 components through pattern-matching:

1. the instruction name,
2. the spacer between 1 and 3,
3. the left delimiter for the target file,
4. the target file name,
5. the right delimiter.

Any of these components may be the empty string if it does not exist in the language syntax. The pattern is internally constrained to match at the start of the fragment; you do not need to specify it with an initial `^`.

Examples:

```
'directive' => '([\w]+)(\s+)()([\w:]+\b)()'   # Perl
#                -----         --------
#                  1     2  3    4      5
```

`use` or `require` is captured by the first pair of parentheses, spaces by the next. There is no delimiter, hence the empty pair of parentheses. We find next the target module with the colon separator. Finally, there is no right delimiter.

[35] Prior to 1.2 release, this parameter was named `'first'`.
[36] Prior to 1.2 release, this parameter was named `'last'`.

Note:

Starting in 0.11 release, ***Perl*** is processed by a dedicated parser with built-in *include* rules. This is also the case for ***Python*** and ***Ruby*** from 1.0 release, for ***Java*** from 1.1 release. They no longer use `'include'` specification.

```
'directive' => ([\w\#]\s*[\w]*)(\s+)(?| (\")(.+?)(\")|(\0<)(.+?)(\0>))
#                --------------       with            or
#                      1         2        3    4   5      3    4   5
```

This complex example covers both forms of C/C++. Note the usage of `(?|` to keep the same numbering of components across alternatives. `\0<` and `\0>` are used instead of `<` and `>` because LXR marks all characters which have a special meaning in HTML to distinguish them from the tags LXR adds to the source text. The following shorter form:

```
'directive' => ([\w\#]\s*[\w]*)(\s+)(\"|\0<)(.+?)(\"|\0>)
#                --------------       ------       ------
#                      1         2      3      4      5
```

is not correct from the language point of view because it does not guarantee a match between the corresponding delimiters. However, you can use it if you are sure the source text compiles without error[37] **and** a file name will not contain a non matching delimiter.

CAUTION!

`(?| … )` is *Perl* 5.10 syntax; it allows to keep the same number for corresponding parenthesised groups in alternatives. If you use it, check the version of your *Perl* interpreter. Dedicated C and *Perl* parsers have been created in 0.11 release to work around this dependency.

`'pre'` is an optional transformation applied first on the original target file name. Then the optional `'global'` is repeatedly applied until it matches no more. Finally, if it exists, `'separator'` causes all language-specific separators to be replaced by the default OS path separator (`/`) and the optional `'post'` is applied once.

Example (*Perl*):

```
# no 'pre'
, 'separator' => '::'
, 'post'      => [ '$', '.pm' ]
```

`'separator'` will replace all `::` *Perl* delimiters by the OS `/` delimiter. The last step by `'post'` adds file extension `.pm` at the end of the resulting path.

An equivalent result can be obtained with a “global” substitution rule, but less efficiently:

```
# no 'pre'
, 'global' => [ '::', '/' ]
, 'post'   => [ '$', '.pm' ]
```

`'global'` explicitly cites the `'::'` substitute. `'global'` is more general than `'separator'` and can operate on something different from the path separator.

[37] LXR is not a compiler; it does not aim to detect language errors. Source text under analysis is supposed to be language-compliant. Consequently, LXR parsing need not mimic compiler parsing accurately.

When no `'include'` description is provided and an *include* region must be processed, the following defaults apply:

- `'directive'` is equivalent to an identifier, parseable by `'identdef'`, followed by white space, followed by the bare file name, *i.e.* without delimiters such as quotes or double quotes,
- `'separator'` is the OS path separator,
- `'pre'`, `'global'` and `'post'` are omitted.

6.3.g. Bridging *ectags* and LXR

```
'typemap' => dictionary
```

This parameter is used to associate the classification of symbols as determined by *ectags* to a human readable class. The dictionary is a list of key/value pairs. The key is a letter and the value a string for the symbol class name. This string is displayed in the identifier and search pages.

See the ***ectags*** *man* page for the list of letters and their meaning.

Example:

```
'typemap' =>
     { 'c' => 'class'
     , 'd' => 'macro (un)definition'
     , 'e' => 'enumerator'
     }
```

7 Web server configuration

This chapters gives hints on how to smoothly integrate LXR into a web-site respecting local conventions. This starts with individual single-tree LXR and tops with multiple-trees subsite within a wider server.

7.1. A note on the interaction between URLs and LXR configuration

How do we translate a URL into an LXR edited display of a file?

The answer to this question addresses two issues.

1. How do we transfer control to the LXR script?
2. How do we designate the source root?

Let's have a look at a typical LXR URL:

```
http://site.url.example/hierarchical_path/script/path_to_file?query
```

The last part `/path_to_file?query` is usually generated by LXR and contains the file to manage (path relative to `'sourceroot'`) and the parameters for the script in `?query`. That part is highly variable and is the way to drive LXR into expressing its valuable power. This is LXR estate and we can do nothing with it. Or rather, we must not interfere with it, lest LXR goes into trouble.

The part `/script` selects the script (*source*, *diff*, *ident*, *search*) to execute. No fancy to play with it.

The first part `http://site.url.example/hierarchical_path` must give an answer to the above two issues. But the answers are intermixed.

Huge organisations could devote individual site names to pairs server/source root. All there is to do is to configure `DocumentRoot` (in *Apache*) or `server.document-root` (in *lighttpd*) to point to the per-organisation unique LXR directory. *lxr.conf* will take care of the rest.

Individual users with a single site name can play a similar game by adding a prefix to their host name. For instance, if they own `site.url.example`, they can use `tree.site.url.example`. But they will have to create a `VirtualHost` for each tree to point to the unique directory. Relying on the default `VirtualHost` in order to handle undefined host names (catch-all) is not a good practice and could route non-LXR request to LXR by mistake.

An *Apache* alternative is to use the `AliasMatch` directive (which scans only the `/hierarchical_path`) to send a matching path to a different directory than one resulting from literal reading of the URL. The *lighttpd* equivalent alternative uses `alias.url` but requires more manual configuration.

The author of this note uses the URL prefix

```
http://site.url.example/lxr
```

to route the request to LXR. It is followed by the tree name as in:

```
http://site.url.example/lxr/lxr
http://site.url.example/lxr/kernel
```

`http://site.url.example` is coded in `'host_names'`. `/lxr/lxr` and `/lxr/kernel` are put into `'virtroot'` for two trees.

Yet another alternative needing no change in the web server configuration makes use of symbolic links in `DocumentRoot`. This may be the only alternative if you have no administrator access to the server configuration directory, *e.g.* if only `~user` web sites are allowed on the system.

7.2. *Apache*

File *.htaccess* as copied by script *configure-lxr.pl* does not need to be customised.

7.2.a. LXR as the main site

`DocumentRoot` points to the LXR root directory which is usually */var/www/html/*. Parameter `'virtroot'` reduces to `'/'`. Access LXR with URL:

```
http://site.url.example/source
```

Some site administrators may restrict CGI scripts to */var/www/cgi-bin/* directory. In that case, install everything in */var/www/cgi-bin* which will behave as `DocumentRoot`. You may need to manually merge LXR-specific file *.htaccess* with existing *.htaccess*. Parameter `'virtroot'` is equal to `'/cgi-bin'`. Access LXR with URL:

```
http://site.url.example/cgi-bin/source
```

If file *robots.txt* is tentatively interpreted as a script, remove it.

7.2.b. LXR as a sub-site

The LXR root directory is a sub-directory of `DocumentRoot`, *e.g.* *lxr/*. Parameter `'virtroot'` becomes `'/lxr'`.

When CGI scripts are restricted to */var/www/cgi-bin* directory, install everything there if LXR is the only CGI service or in a subdirectory, *e.g.* *lxr/*, if several services must coexist. In the latter case, there is no need to merge *.htaccess*. Then create symbolic links from `DocumentRoot`:

```
$ ln -s /var/www/cgi-bin /var/www/html/lxr
```

or

```
$ ln -s /var/www/cgi-bin/lxr /var/www/html/lxr
```

In both cases, access LXR with URL:

```
http://site.url.example/lxr/source
```

7.2.c. LXR as an independent site

LXR service appears to be issued from another site coexisting with the "normal" site. Instead of being served from `//site.url.example`, it comes from, say, `//lxr.url.example`. For that, you need to create a `VirtualHost` in file *apache-lxrserver.conf*:

```
<VirtualHost *:80>
    ServerName    lxr.url.example
    DocumentRoot  /OS_absolute_path_to_LXR_root_directory
</VirtualHost>
```

The `Directory` record is already configured by script *configure-lxr.pl*.

```
<Directory  /OS_absolute_path_to_LXR_root_directory
    Options FollowSymLinks
    AllowOverrride AuthConfig FileInfo Limit Options
        # up to Apache 2.2
    Order  allow,deny
    Allow  from all
        # from Apache 2.4
#   Require all granted
</Directory>
```

The LXR root directory can now be located anywhere and is no longer restricted to */var/www/html/*.

If you are running under a security enhanced OS, like *Security Enhanced Linux* (*SELinux*), you may get security alerts preventing LXR scripts execution. In this case, type the following command:

```
$ chcon --reference /var/www/cgi-bin/ -R /LXR/root/directory/
```

Parameter `'virtroot'` reduces to `'/'`. Access LXR with URL:

```
http://lxr.url.example/source
```

Having LXR as a part of an independent site is only a matter of playing with `'virtroot'` as in the previous section.

7.2.d. Multiple trees operation

The first trivial solution creates a site per tree with the technique exhibited in the previous section. However, this leads to have as many copies of the LXR root directory as there are trees. In fact, you duplicate single-tree operation.

The second solution uses symbolic links inside `DocumentRoot`. Install LXR into subdirectory *lxr/tree1/*[38]. Create as many links to this subdirectory as you need:

[38] Using a subdirectory allows to have other HTML pages besides LXR service without mixing them with CGI scripts which could cause problems. Of course, you can have LXR in `DocumentRoot`*/tree1/* instead of

```
$ cd /path/to/DocumentRoot
$ mkdir lxr
$ mkdir lxr/tree1
$ cd lxr/tree1
$ #  --  install LXR in tree1 --
$ cd ..     # back to lxr
$ ln -s tree1 tree2
$ ln -s tree1 tree3
```

lxr.conf will contain three source-tree descriptions with `'virtroot'` equal to `'lxr/tree1'`, `'lxr/tree2'` and `'lxr/tree3'`. Source-tree *tree1* is accessed with URL:

```
http://site.url.example/lxr/tree1/source
```

The drawback of this second solution is the necessity to change the `DocumentRoot` architecture whenever you add or remove a tree: you must add or remove a link.

The third solution installs LXR in subdirectory *lxr/* and will have the URL processed by *Apache* to route all requests with URL like

```
http://site.url.example/lxr/tree1/source
```

to `DocumentRoot`*/lxr/*.

The needed directive in *apache-lxrserver.conf* is `AliasMatch`:

```
AliasMatch ^/lxr/[^/]+/(.*) "OS_absolute_path_to_DocumentRoot/lxr/$1"
```

The net effect of this `AliasMatch` directive is to strip off the tree name from the server routing information, but the LXR scripts still receive the original URL and can retrieve the target tree from the fragment not used for routing.

This directive is activated when you answer `m` to the `Configure for single/multiple trees?` question and `yes` to `Use built-in multiple trees management?` in script *configure-lxr.pl* .

With this scheme, no modification is needed in the `DocumentRoot` architecture no matter how many trees are present. Adding a tree or deleting a tree resolve to modifying only *lxr.conf*. With the addition of `'htmlfatal'`, a nice message can be displayed if someone references a non-existent tree.

CAUTION!

This scheme requires that you can copy file *apache-lxrserver.conf* into */etc/httpd/conf.d*. If you have no administrator rights, you are limited to the second solution.

To implement an alternate scheme, replace the above `AliasMatch` directive in *apache-lxrserver.conf* and write the corresponding `'treeextract'` parameter in *lxr.conf*.

```
AliasMatch ^/lxr/[^/]+/(.*) "OS_absolute_path_to_DocumentRoot/lxr/$1"
#                 -----   <- tree designation removed from routing
```

`DocumentRoot`*/lxr/tree1/* if you do not need HTML pages.

AliasMatch is applied to the URL path. Remember to capture the rest of the path after the match and to copy it into the rewritten path. This explains the parentheses (…) in the pattern and the $1 in the replacement to transfer the script name and its arguments.

```
     , 'treeextract'  =>   '([^/]*)/[^/]*$'
# captured tree designation>------- ===== < script name at URL path end
```

This is the mirror of AliasMatch. The pattern is applied to SCRIPT_NAME. It is anchored at the end of the path. It ignores the script (*ident*, *diff*, *search* or *source*) and captures the preceding part representing the tree identification.

A subtle difference is the use of * instead of + so that the pattern matches even when LXR is the only service in the server (single tree, LXR at root of server), but this is not necessary because, in this case, routing should always succeed unless you damaged your *lxr.conf*.

7.3. *Lighttpd*

Since this support is experimental, configuration file *lighttp-lxrserver.conf* needs to be manually tailored. Apart from the final LXR section, it is fairly identical to "standard" *lighttpd.conf* with the exception that directories have been suffixed with /. This could cause incompatibility if files from *conf.d/* or *vhosts.d/* are included.

var.state_dir has been changed to */var/run/lighttpd* to solve a compatibility issue with script *lighttpd-init*.

mimetype.assign has been extended to allow *.shtml* files as text/html.

In the final LXR section, the OS-absolute path of LXR root directory has been inserted by script *configure-lxr.pl*.

Note:
Remember that you can run *lighttpd* as a private task if you have no administrator privilege to launch it as a daemon or to access the configuration directory. Do not forget to fix the configuration to reference only directories with sufficient permissions, notably state_dir.

7.3.a. *Lighttpd* as the main site

server.document-root points to the LXR root directory. Parameter 'virtroot' reduces to '/'. Access LXR with URL:

```
http://site.url.example/source
```

Add as many $HTTP["host"] == or $HTTP["host"] =~ (with pattern matching) paragraphs to describe all the aliases for the primary host name.

Note:
The primary host name is written by the configurator with initial double slashes // (URL form). If this causes trouble, remove the //.

7.3.b. *Lighttpd* as a sub-site

The LXR root directory is a sub-directory of `server.document-root`, *e.g.* *lxr/*. Parameter `'virtroot'` becomes `'/lxr'`. It is necessary to wrap `cgi.assign` in:

```
$HTTP["url"] =~ "^/lxr" {
    cgi.assign += ( "/source" => "", "/ident" => "", … )
}
```

Access LXR with URL:

```
http://site.url.example/lxr/source
```

Note:

The generated `server.document-root` may conflict with the definition for the effective site root. If this is the case, remove the line. The same is true for the `$HTTP["host"]` descriptions. They are likely to have already been given in the site root definitions.

7.3.c. *Lighttpd* as an independent site

LXR service appears to be issued from another site coexisting with the “normal” site. Instead of being served from `//site.url.example`, it comes from, say, `//lxr.url.example`. For that, you tell *lighttpd* how to route the URL in file *lighttpd-lxrserver.conf*:

```
$HTTP["host"] == "lxr.url.example" {
    server.document-root = "/OS_absolute_path_to_LXR_root_directory"
}
```

This replaces the unnested `server.document-root` for the main site case. You do not need to modify `cgi.assign`.

Add as many `$HTTP[“host”] ==` or `$HTTP[“host”] =~` (with pattern matching) to describe your aliases.

`'virtroot'` is `'/'`. Access LXR with URL:

```
http://lxr.url.example/source
```

Having LXR as a part of an independent site is only a matter of playing with `'virtroot'` and `$HTTP["url"]` as in the previous section.

7.3.d. Multiple trees operation

The first trivial solution creates a site per tree with the technique exhibited in the previous section. However, this leads to have as many copies of the LXR root directory as there are trees. In fact, you duplicate single-tree operation.

The second solution installs LXR in subdirectory *lxr/* and will have the URL scanned by *lighttpd* to route all requests with URL like

```
http://site.url.example/lxr/tree1/source
```

to *lxr* with `alias.url` directives.

There is already an example commented out `alias.url` directive (corresponding to the built-in policy) in *lighttpd-lxrserver.conf*. Remove the pound sign (#) and adapt them to fit your needs:

```
alias.url += ( "lxr/tree1" => "/OS_absolute_path_to_LXR_root_directory"
             , "lxr/tree2" => "/OS_absolute_path_to_LXR_root_directory"
             , "lxr/tree3" => "/OS_absolute_path_to_LXR_root_directory"
             )
```

In this case, *lxr.conf* contains three source-tree descriptions with `'virtroot'` equal to `'lxr/tree1'`, `'lxr/tree2'` and `'lxr/tree3'`.

To implement an alternate scheme, replace the `$HTTP["url"]` pattern and `alias.url` rules in *lighttpd-lxrserver.conf* and write the corresponding `'treeextract'` parameter in *lxr.conf*.

```
$HTTP["url"] =~ "^/lxr/[^/]+/ {
#                      -----  <- tree designation here, but ignored
    alias.url += { "/lxr/tree_designation/"
                     => "/OS_absolute_path_to_LXR_root_directory/"
}
```

There is no URL or path rewriting here, only a match to decide how to route. Consequently, it is not necessary to copy anything beyond what is matched. The pattern above for the built-in policy is redundant, only to illustrate the similarities with *Apache*. It could be reduced to `^/lxr/` since the tree identification is not used in routing.

Unhappily, there is no *wild card* rule for `alias.url`. Every tree must be listed. The configurator takes care of that. Remember to enumerate all the trees in manual configuration.

Note the trailing slash after the tree designation. It is used to force matching only when it is followed by a script name.

```
     , 'treeextract'  =>   '([^/]*)/[^/]*$'
# captured tree designation>------- ===== < script name at URL path end
```

This is the mirror of `$HTTP["url"]`. The pattern is applied to `SCRIPT_NAME`. It is anchored at the end of the path. It ignores the script (*ident*, *diff*, *search* or *source*) and captures the preceding part representing the tree identification.

A subtle difference is the use of `*` instead of `+` so that the pattern matches even when LXR is the only service in the server (single tree, LXR at root of server), but this is not strictly necessary because, in this case, routing should always succeed unless you damaged your *lxr.conf*.

8 Customising page architecture

LXR provides a very flexible way of adapting its display to one's taste through two features: templates *and* CSS styles.
Pages synthesized by LXR are based on filling templates fields. They all share a common structure: a header, a body and a footer.
Composing a new template requires HTML knowledge.

Templates are merely HTML fragments of a page. An LXR page is divided in a head, a content and a tail. Head and tail are composed from a template while the content may or may not be under template control. The head template contains the beginning of the synthesised page with the `<html>` tag, the `<head>` section, the `<body>` tag and the beginning of the content for the top of the page (the header). The tail template closes the page with the footer row and the `</body>` and `</html>` tags.

Template content is a valid HTML fragment. Comments can be inserted. Two forms are supported:

- regular comments, which are removed from the result to preserve bandwidth when page is sent to the browser

  ```
  <!-- regular comment, note the space before the first word -->
  ```

  This is how the template licence statement is removed from the generated page: the licence applies to the template, not to the result since the sent HTML page essentially contains user data which might be covered by a specific intellectual property right.

- *sticky* comments, which are kept in the final result (amongst them are SSI directives)

  ```
  <!--sticky comment: no space before first word -->
  <!--#include virtual="/some/file.shtml" -->
  ```

The templates that ship with each release present two user interfaces: the classical one based on links and the new one providing buttons and menus.

When your tree has lots of versions, links consume an excessively large space in the header area, which gives a cluttered aspect to the header. In this case, it is nicer to use *buttons-and-menus*. You may however mix *links* and *buttons-and-menus* without harm.

The example header for *buttons-and-menus* style is *html-head-btn.html*. You must then configure your *lxr.conf* as:

```
, 'htmlhead' => 'templates/tpml/html-head-btn.html'
```

CAUTION!

Buttons-and-menus style composition is very tricky. Study thoroughly the provided examples before attempting to create new ones. You must also review *lxr.css* to achieve the desired layout. *And, remember, always work on a copy in* custom.d/.

The templates are identified by a conventional name which is transformed into a file name through the parameters in *lxr.conf* HTML subsection. The components of LXR send a prefix to the template engine: `diff`, `ident`, `search` or `source`. A suffix is added and the result is the key for the parameter. If it does not exist in *lxr.conf*, prefix `html` replaces the provided one and a new lookup is attempted. If it does not exist, a very simple internal template (without any substitution capability) is used.

Some components of LXR do not use body templates, namely file display and difference markup.

The template definition syntax in *lxr.conf* is:

```
template_key => path_to_template
```

path_to_template is either a path relative to the *LXR root directory* or an OS absolute path. If you want to use directly the templates from the *templates/* directory, which is not recommended, you must use a directive like:

```
, 'htmlhead' => templates/tmpl/html-head.html
```

Note:
As a fail-safe rule, always customise a **copy** of the examples provided, so that you can revert to a working template if something goes wrong.

8.1. Variable text in templates

Insertion of LXR enhanced data is done through substitution of special *markers*. They are recognised by the appearance of $*simple_name* in the HTML code where *simple_name* is composed of alphanumeric or underscore characters (`a` through `z`, `A` through `Z`, `0` through `9` or `_`). If the marker does not exist (really does not exist or is used in a wrong context), nothing is substituted and the marker is left unchanged in HTML code.

There are two types of markers:

- simple markers
- block markers

Tip:
A simple marker might need to be followed by alphabetic text without any intervening space after the marker name. This situation prevents correct detection of the marker name. The solution is to use a block marker syntax with an empty block.

8.1.a. Simple markers

Simple markers are replaced by a single value. They are associated with a scalar LXR data.

Note:
There is inherently no difference between a simple marker and a block marker with an empty

block. It is only a convenient short-hand notation.

Example:

```
<tr>
     <td class="banner">$banner</td>
</tr>
```

8.1.b. Block markers

Block markers are followed by a block delimited by curly braces ({}) without any intervening spaces. They are usually associated with an iterative function allowing for repetitive substitution according to some rule.

The content of the block between the curly braces is arbitrary. It may even contain markers. The only constraint is that block markers be properly *nested*. Their blocks are fully contained inside the outer pair of curly braces. Substitution starts from the outermost block, thus giving complete freedom to the associated function to expand or not an inner block.

CAUTION!
The rule of balanced braces prevents insertion of isolated braces anywhere in a template. The workaround for HTML code is the use of *character references* `&#123;` and `&#125;` (or hexadecimal `&#x7B;` and `&#x7D;`).

Example:

```
<table class="header">
$variables{      <tr>
           <td class="leftmost">
                 $varname:
           </td>
           <td class="rightmost">
$varlinks{[&nbsp;$varvalue&nbsp;]}
           </td>
     </tr>
}</table>
```

This results is a line for each variable, thanks to the `<tr>` tag. All the values defined in `'range'` are edited in this line.

8.2. *Markers for headers*

The default header template is defined by:

```
'htmlhead' => 'templates/tmpl/html-head.html'
```

But you can have more: `sourcehead` (for file header), `sourcedirhead` (for directory), `diffhead` (for differences between files), `identhead` (for identifier search), `searchhead` (for free text search), `showconfighead` (for configuration display).

These templates MUST open an HTML-compliant page with a `<!DOCTYPE>`, the opening `<html>`, the complete `<head>` section, the opening `<body>` and any desired fixed header decoration.

The following paragraphs describe the available markers with typical application.

8.2.a. `<head>` section

Marker	Provides	Replaced by
	`Example`	
`$title`	Content of the `<title>` element	Current path for *source* and *diff*, the concatenation of `'sourcerootname'` and the search target for *ident* and *search*
	`<title>$title</title>`	
`$encoding`	Character set name of `charset=` attribute in a `<meta>` tag	Value of configuration parameter `'encoding'`
	`<meta http-equiv="content-type" content="text/html; charset=$encoding">`	
`$baseurl`	Reference URL in a `<base>` tag to resolve relative links	Concatenation of `'host_names'` and `'virtroot'` or the real `'baseurl'`
	`<base href="$baseurl">`	
`$stylesheet`	Location of the CSS style sheet in `<link rel="stylesheet">` tag	Value of configuration parameter `'stylesheet'`
	`<link rel="stylesheet" type="text/css" href="$stylesheet">`	

Note:

Parameter `'stylesheet'` is not transformed in any way. It can be written HTML-absolute or HTML-relative in *lxr.conf* according to the global site configuration. HTML-relative is convenient for single-tree operation with *LXR root directory* equal to `DocumentRoot`. HTML-absolute is preferred when LXR is only a sub-site.

8.2.b. `<body>` section

8.2.b.1. *Titling*

Following markers are intended for composing an informative titling section for the page.

Marker	Provides	Replaced by
	`Example`	
`$caption`	Fixed general title for the page	Value of configuration parameter `'caption'`
	`<h1 class="main">$caption</h1>`	
`$banner`	Path to the current file for pages dealing with files (*i.e. source* or *diff*)	Clickable hyperlinks in every component of the path or an empty string
	`<p class="banner">$banner</p>`	
`$pathname`	Path to the current file for *source* or *diff*, reduces to `/` otherwise	Value of *Perl* variable `$pathname` (see marker `$banner` if hyperlinks desired)
	`<p>Current target is $pathname</p>`	
`$LXRversion`	LXR release version id	LXR version as set by script *set-lxr-version.sh*
	`<p>LXR version is $LXRversion</p>`	

If your tree is stored in a **CVS repository**, you can add a "link" button to display files in the current directory that were put aside in the "Attic" (recall that CVS does not manage directory version; the "attic" feature is the nearest equivalent for directory versioning).

`$atticlink`	Toggling state "show/hide files in attic" hyperlink when current script is *source*	Empty string if source tree is not managed by CVS
	`<$div>$atticlink</div>`	

8.2.b.2. Navigating to other trees

When LXR manages several source trees, it is convenient to have hyperlinks to the other trees instead of manipulating the URL, with the risk of a typing mistake. The set of hyperlinks is generated by the `$forest{}`[39] block marker:

[39] Tip: a forest is made up of many trees.

Marker (context)	Provides	Replaced by
`$forest{}`	An area for the trees hyperlinks	The expanded block if there exists at least one tree with a `'shortcaption'` parameter
`$trees{}` (`$forest{}` *block)*	Repeated application of block to each tree	Concatenation of all expansions
`$treelink` (`$trees{}` *block)*	Hyperlink to the root of a tree	An `<a>` tag or a `<span>` block for the currently displayed tree
`$caption` `$link` (`$trees{}` *block)*	Reserved for future extension	

Example:

```
<td class="treelink">
$forest{   <span class="treelink compact">
$trees{$treelink&#x200B;}</span>
}</td>
```

8.2.b.3. Switching between modes

Headers usually offer a set of links or buttons to switch between the **LXR operating modes**. This avoids manipulating directly the URL, which is inconvenient and error-prone. The set is generated by the `$modes` block marker:

Marker (context)	Provides	Replaced by
	`Example`	
`$modes{}`	Repeated application of block to each mode	Concatenation of all expansions
	`$modes{``<!--individual mode layout--><br>`**`}`** Note the `<br>` tag to stack the links one above the other.	
`$modelink` *(note 1)*	A ready-to-use hyperlink	An `<a>` tag for the mode
	`$modes{``[&nbsp;$modelink&nbsp;]<br>`**`}`**	
`$modeaction`	URL to jump to if *submit* button is clicked	An `action=` attribute for `<form>` tag

Marker (context)	Provides	Replaced by
	`Example`	
(note 2)	`<form method="get" action="$modeaction">`	
`$modename` *(note 2)*	Mode name	Internally generated string naming the mode
`$modeoff` *(note 2)*	Mode current operating status	`disabled` or an empty string
	`<button type="submit" $modeoff>$modename</button>`	
`$modecss` *(note 2)*	A means to change button appearance according to mode current operating status	`modes`, `modes-sel` if this is the active mode or `modes-dis` if the mode is disabled
	`<form method="get" class="$modecss" … > … </form>`	
`$urlargs{}` *(note 2)*	A means to pass LXR state to the next script	*This common block marker is explained further below*

Context notes:

1. inside `$modes{}` block for *link* interface
2. inside `$modes{}` block for *buttons-and-menus* interface

Example:

```
$modes{<form method="get" class="$modecss" action="$modeaction">
     <div class="compact">
$urlargs{  <input      type= "hidden"
                       name= "$urlvar"
                       value="$urlval"
           >
}          <button type="submit" $modeoff>$modename</button>
     </div>
</form>
}
```

8.2.b.4. *Setting variables*

IMPORTANT ADVICE

Do not include this feature in *showconfig* header. Changing variable values has no meaning there and may have unexpected effects when switching mode or tree. Stated otherwise, do not use the shareable `'htmlhead'`, design a specific `'showconfighead'` without the markers described in this section.

Variables defined by `'variables'` in *lxr.conf* constitute the basis for file version selection. Headers usually offer a means of **manipulating variable values** through a set of links or menus. This avoids manipulating directly the URL, which is inconvenient and error-prone. The set is generated by the `$variables` block marker and two specific markers in case of *buttons-and-menus* interface:

Marker (context)	**Provides**	**Replaced by**
`$variables{}`	Repeated application of the block to each variable	Concatenation of all expansions
`$varname` (`$variables{}` *block)*	Purpose of variable	Value of attribute `'name'` in the `'variables'` definition

- For *link* interface, use the following inside the `$variables{}` block:

Marker (context)	**Provides**	**Replaced by**
`$varlinks{}`	Repeated application of the block to each value in `'range'` attribute	Concatenation of all expansions
`$varvalue` (`$varlinks{}` *block)*	Display of one value	One value of attribute `'range'` in the `'variables'` definition

Example:

```
<table class="header">
$variables{      <tr>
           <td class="leftmost">$varname:</td>
           <td class="rightmost">
$varlinks{[&nbsp;$varvalue&nbsp;]}
           </td>
     </tr>
}</table>
```

The possible values in `'range'` follow one another on a line and every variable is set on its own table row.

- For *buttons-and-menus* interface, use the following **OUTSIDE** the `$variables{}` block:

Marker	**Provides**	**Replaced by**
`$varbtnaction`	URL to jump to if submit button is clicked	Target URL *without* query string

Marker	Provides	Replaced by
`$urlargs{}`	A means to pass LXR state to the next script	This common block marker is explained further below

- For *buttons-and-menus* interface, use the following **INSIDE** the `$variables{}` block:

Marker (context)	Provides	Replaced by
`$varname`	Purpose of variable	Value of attribute `'name'` in the `'variables'` definition
`$varid` *(see note)*	The variable identifier	Name of the variable in the `'variables'` definition
`$varmenu{}`	Repeated application of block to each variable value defined in `'range'` attribute	Concatenation of all expansions
`$varvalue` (`$varmenu{}` *block)*	A possible menu choice for an `<option>` element	A value of attribute `'range'` in the `'variables'` definition
`$itemsel` (`$varmenu{}` *block)*	"Current" status for a possible variable value	`selected` if the choice matches the current value of the variable or an empty string
`$itemclass` (`$varmenu{}` *block)*	A means to change the menu item appearance according to its "current" status	`varlink` or `var-sel` if this is the "current" value

CAUTION!
Due to the LXR HTTP state transition rules, the `$varid` replacement MUST be preceded by an exclamation mark (`!`) which means "override the variable value on page load". This convention simplifies handling of conflicting variable values between `$urlargs{}` and `$varid`.

Example:

```
<form     method="get"
          class="vars"
          action="$varbtnaction"
>
     <p class=compact>
$urlargs{       <input type="hidden" name="$urlvar" value="$urlval">
}$variables{    &nbsp;<label>$varname:
```

```
           <select name="!$varid">
$varmenu{              <option class="$itemclass"
$itemsel>$varvalue</option>
}          </select>
           </label>
}          <button type=reset>Revert</button>&nbsp;&nbsp;
           <button type=submit>Change</button>
     </p>
</form>
```

Notes:

Do not forget the exclamation mark before `$varid`

The variables menus are all side-by-side since there is no `<br>` tag

Provide a submit button to activate the preselected changes

8.2.b.5. Passing state to the next script

Since the HTTP protocol is stateless, any attempt to implement clear state transitions must transfer the needed information through the *query string* part of the URL. This is how LXR passes its commands from one invocation of its scripts to the next. State transition is usually transparent to the user. However, creating buttons or menus is not as straightforward as coding a link.

Transition through a link is easy because `<a>` tags contain all the needed information in a single location, the `href` attribute.

On the other hand, buttons or menus are included in a `<form>` construct. The target script is designated in the `action` attribute of the `<form>` tag. The desired set of variable values is determined from `<select>` tags and `<option>` elements, while the present state of variables and query arguments is memorised in `<input type="hidden">` tags. This creates an intricate interaction between both sets. To solve it, one must examine the modified variables and filter the current state in order not to send back a variable with two values. There is a simpler and more elegant solution suggested by one of LXR design goals. LXR tries to stay “minimalist” and avoids as much it can scripting functions *e.g. Java* or *JavaScript*. The modified variable value is submitted with a marking character, a “sigil”, affixed to the variable name to mean “override this variable with this value”. That way, the two sets become independent and the programmer does not need to worry about the interaction.

As mentioned, the desired value is transmitted through `<select>`/`<option>`. The current state is summarised with `$urlargs` block marker.

Marker (context)	**Provides**	**Replaced by**
`$urlargs{}`	Repeated application of block to each query argument to report the present state	Concatenation of all expansions
`$urlvar` (`$urlargs{}` *block)*	Name of a query argument	Name string

Marker (context)	Provides	Replaced by
`$urlval` (`$urlargs{}` *block)*	Value of a query argument	Value string

Typical usage:

```
$urlargs{ <input type="hidden" name="$urlvar" value="$urlval"> }
```

8.2.b.6. *Other markers*

The `$dotdoturl` and `$thisurl` markers have probably no usage. They have been offered by LXR for ages but are not reliable.

Marker	Provides	Replaced by
`$dotdoturl`	URL for the parent directory of `'virtroot'`	Concatenation of `'host_names'` and `'virtroot'` with the last directory removed

CAUTION!

This marker is not protected: it removes blindly what it thinks to be the last directory. If your `'virtroot'` is `/`, it will strip the host name! It is also unaware of any DNS magic you may apply to your URL (see 7.1 A note on the interaction between URLs and LXR configuration), which may invalidate the URL.

There is also no guarantee that this “dot dot URL” may be accessed by the web server. No access is possible above `DocumentRoot`!

Marker	Provides	Replaced by
`$thisurl`	URL of current page	Complete copy (query string included) of the URL used to request this page

The last block marker is of interest only to developers:

Marker (context)	Provides	Replaced by
`$devinfo{}`	Repeated application of block for every LXR *Perl* module	Concatenation of all expansions
`$moduleid` (`$devinfo{}` *block)*	CVS information for this module	Value of *Perl* variable `$CVSID` from this module
`$modpath`	Path to module	*Perl* module path

Marker (context)	Provides	Replaced by
(`$devinfo{}` *block*)		
`$modtime` (`$devinfo{}` *block*)	Last modification time	UTC time and date string

8.3. *Markers for footers*

The default footer template is defined by:

```
'htmltail' => 'templates/tmpl/html-tail.html'
```

But you can have more: `sourcetail` (for file footer), `sourcedirtail` (for directory), `difftail` (for differences between files), `identtail` (for identifier search), `searchtail` (for free text search), `showconfigtail` (for configuration display).

These templates MUST close an HTML-compliant page with any desired fixed footer decoration, the closing `</body>` and the closing `</html>`.

Basically, the same markers as those for headers are available, with the exception of those related to the `<head>` section.

It has been thought unnecessary to make the `$atticlink` marker available.

8.4. *Markers for directory listing*

The body for listing directory contents with the *source* script is controlled by the template:

```
'htmldir' => 'templates/tmpl/html-dir-indexing.html'
```

or, without the *Last indexed* column:

```
'htmldir' => 'templates/tmpl/html-dir.html'
```

If not found, it defaults to showing an icon and the file name.

It should define the elements of the directory listing, usually with a `<table>`. Two block markers can be used:

Marker (context)	Provides	Replaced by
`$description{}`	Directory description	Expansion of block
`$files{}`	Repeated application of	Concatenation of all expansions

Marker (context)	Provides	Replaced by
	block to each file or subdirectory to report file characteristics	This marker is preferentially used in a `<table>` to generate one row per file.
`$dirclass` (`$files{}` *block)*	A means to change this file "line" appearance	Alternatively `dirrow1` and `dirrow2` every 3 files
`$iconlink` (`$files{}` *block)*	A clickable image representative of the file extension	An `<img>` tag
`$namelink` (`$files{}` *block)*	File name	An `<a>` link to the file
`$filesize{}` (`$files{}` *block)*	Template for file size	A dash for a directory or block expansion
`$bytes` (`$filesize{}` *block)*	File size	File size (in bytes or kilobytes depending on the size)
`$modtime` (`$files{}` *block)*	Last modification date and time for file	An UTC date string
`$indextime` (`$files{}` *block)*	Last indexation date and time for file	An UTC date string, *Not valid* warning or a dash; void for directories
`$dirindexclass` (`$files{}` *block)*	HTML class attribute allowing to decorate `$indextime` with CSS styles	Either `dirindex` or `dirindexinvalid` according to indexation state for file.
`$description{}` (`$files{}` *block)*	Description of subdirectory or file	Block expansion
`$desctext` (*both* `$description{}` *blocks)*	Descriptive text	Text extracted from *README* files for subdirectories, some guessed descriptive text for files or an empty string

Example:

```
<div class="dirdesc">
$description{$desctext}
</div>

<table class="dircontent">
     <thead>
           <tr class="dirheader">
                 <th class="diricon">&nbsp;</th>
                 <th class="dirname">Name</th>
                 <th class="dirsize">Size</th>
                 <th class="dirtime">Date (UTC)</th>
                 <th class="dirindex">Last indexed</th>
                 <th class="dirdesc">Description</th>
           </tr>
     </thead>
     <tbody>
$files{
           <tr class="$dirclass">
                 <td class="diricon">$iconlink</td>
                 <td class="dirname">$namelink</td>
                 <td class="dirsize">$filesize{$bytes bytes}</td>
                 <td class="dirtime">$modtime</td>
                 <td class="$dirindexclass">$dirindex</td>
                 <td class="dirdesc">$description{$desctext}</td>
           </tr>
}
     </tbody>
</table>
```

8.4.a. Directory description

LXR tries its best to help understand the meaning of a source-tree. It tries to find *README.txt*, *README* or *README.html* files in the displayed directory. The first file found is scanned in the hope of finding something “interesting”. Considering the developer freedom and the variety of comment formats, the heuristics is not guaranteed to hit a meaningful bit of text.

Once the possible licence paragraph is stripped from text *README* files, a small number of paragraphs[40] from the beginning of the file is kept. These paragraphs are returned as the directory description.

Note:

If you are fluent enough in *Perl*, you can customise the heuristics to your commenting habits by modifying *Local.pm*.

If *README.html* was selected, a `<span class="lxrlongdesc">` element is looked for. This element may contain a nested `<span class="lxrshortdesc">` element. If this latter element does not contain any other `<span>` element, its content (without the `<span>` and `</span>` tags) is used as the directory description. Otherwise, the content (without the tags) of the `"lxrlongdesc"` element is returned, once again if it contains no nested `<span>` element.

[40] A paragraph is defined as a sequence of consecutive lines preceded and followed by at least a blank line.

This feature may be used to provide some insight information to the reader of a public LXR server.

8.4.b. File description

Trying to extract descriptive information from files is even more arbitrary. The default *Local.pm* attempts this endeavour only on C, C++ or *Java* files.

The heuristics operates from the beginning of the file to roughly the first entity declaration. Licence paragraph and borders are stripped from this fragment. A preferred target is some text preceded by the file name or the word “Description”.

After pruning what is considered as “noise”, the remaining text is returned as the file description.

Notes:

Happily enough, you often get an empty string instead of garbage.

Once again, there is no guarantee to extract the real description.

8.5. *Markers for file listing*

File content listing is not driven by templates.

8.6. *Markers for difference markup*

Difference markup is not driven by templates.

8.7. *Markers for identifier search*

The body for identifier search with the *ident* script is controlled by the template:

```
'htmlident' => 'templates/tmpl/html-ident.html'
```

It provides the `<form>` for the identifier query. It uses method `GET` and script *ident* is invoked again when the submit button is clicked. It also contains the mask for the results.

8.7.a. Query form

The `<form>` returns the identifier name in query variable `_i` (note the underscore), the “definitions only” check box state in query variable `_identdefonly`. Query variable `_remember` is set to 1 if you want to preset the `<input>` fields to their previous value.

Marker	Provides	Replaced by
`$variables`	State passing data (version selection) from one script	A series of `<input type="hidden">` tags; to be used as is inside the `<form>`

Marker	Provides	Replaced by
	invocation to the other	
`$checked`	Check status of the "definitions only" check box	`<input>` tag attribute depending on the value of configuration parameter `'identdefonly'` and the previous value of this attribute
`$identifier`	The looked-for identifier; to be used outside the `<form>`	HTML-safe identifier name where the eventual angle brackets <> are replaced by HTML character references[41] (see the next one and the note)
`$identifier_escaped`	The unchanged looked-for identifier for the value of an `<input>` field only	Identifier name where non alphanumeric characters are replaced by their %-encoded equivalent, ready to be transmitted to the LXR server (see the previous one)

CAUTION!

The distinction between `$identifier` and `$identifier_escaped` results from the effort to protect against cross-site scripting (XSS) attacks. In the provided template, `$identifier_escaped` is locked inside quotation marks, so that a smartly crafted search name including some (*JavaScript* or other) script functions cannot get executed by your browser. Do not use `$identifier_escaped` in any other context.

`$identifier` is secure because the "trigger" characters, the angle brackets (needed to code tags) are replaced by HTML character references which are always considered as pure text.

Example:

```
<form method="get" action="ident">
     <p>
  $variables
     <label>
  <b>Identifier: </b>
        <input      id="focus"
                    type="text"
                    name="_i"
                    value="$identifier_escaped"
                    size="15"
        >
     </label>
     <label>
        <input      type="checkbox"
                    name="_identdefonly"
                    value="1" $checked
        >
```

41 This mapping cannot be reversed since no one can tell if `&lt;` was the intended 4-character string or if it was `<`.

```
   &nbsp;&nbsp;Definitions only
     </label>
     <input type="hidden" name="_remember" value="1">
     &nbsp;&nbsp;<input type="submit" value="Find">
     </p>
</form>
```

8.7.b. Results area

The markers offer three choices of editing a reference from a very simple combined string (but dangling from line to line) to a very sophisticated layout possibility.

Marker (context)	Provides	Replaced by
`$defs{}`	Identifier definitions	Expansion of block or empty string if no definition
`$uses{}`	Identifier references	Expansion of block or empty string if no reference
`$occurs` (`$defs{}`/`$uses{}`)	Number of occurrences	Total number of occurrences (definitions or references depending on context)
`$filehits` (`$defs{}`/`$uses{}`)	Number of files	Number of files where the identifier was found (as a definition or a reference according to context)
`$refs{}` (`$defs{}`/`$uses{}`)	Repeated application of block to each occurrence of the identifier	Concatenation of all expansions
`$type` (`$refs{}` *block)*	Identifier type, *e.g.* variable, class, procedure …	`'typemap'` string from a language description in *generic.conf*
`$rel` (`$refs{}` *block)*	Extra information for some categories	Entity name of which this identifier is a member
`$fileref` (`$refs{}` *block)*	Occurrence location **choice #1** (historical)	String combining the file name and the line number as a link with this occurrence as target
`$file` or `$fileonce` and `$line` (`$refs{}` *block)*	Occurrence location **choice #2** (recommended for definitions)	Respectively a clickable link to the file and the line number as a link to the occurrence
`$file` or `$fileonce`	Occurrence location	Respectively a clickable link to the file

Marker (context)	**Provides**	**Replaced by**
and `$lines` (`$refs{}` *block)*	**choice #3** (recommended for references)	and a string containing all the links to the lines of occurrence
`$refinvalid` (`$refs{}` *block)*	HTML class attribute allowing to decorate `$fileonce`, `$file`, `$line` and `$lines` with CSS styles	Expands to `identinvalid` if file has been modified since last indexation; empty otherwise.
`$indexwarning{}` (`$defs{}`/`$uses{}` *but must occur* **after** `$refs{}` *block)*	Warning block for inaccurate references	Expansion of block or empty string if no inaccurate reference *Currently, block is not expanded in its turn; it contains only fixed HTML text.*

Notes:

1. The `$type` and `$rel` markers are intended for definitions. As a safety measure, they return an empty string for a reference.
2. Usually, an occurrence “costs” one line. This can lead to a very long page where scrolling is necessary. Also the `$fileref` string has by nature a varying length. The file name portion is ragged and the line numbers do not line up. The visual effect of choice #1 is poor.
3. Separating file and line number information in choices #2 and #3 allows a neatly aligned tabular form. Using choice #2 for references will “cost” one line per occurrence.
4. `$fileonce` is the same as `$file` for the first reference in a file, then an empty string for the second and following references.
5. Choice #3 makes no difference between `$file` and `$fileonce` because the `$refs{}` block is expanded only once per file. It results in a very compact presentation, suitable for displaying in a `<table>` cell.

Example for definitions:

```
<h2>Results for <span class="identident">$identifier</span></h2>
$defs{
     <h3>Definitions</h3>
     <table class="identdef">
           <tr>
                 <th>Type</th>
                 <th>Member of</th>
                 <th class="identfile">File</th>
                 <th class="identline">Line</th>
           </tr>
  $refs{
           <tr>
                 <td class="identtype">$type</td>
```

```
                <td class="identrel">$rel</td>
                <td class="identfile $refinvalid">$file</td>
                <td class="identline $refinvalid">$line</td>
          </tr>
  }
     </table>
     <p>
$occurs declarations in $filehits files.
     </p>
  $indexwarning{
     <p class="error">One or more files need reindexing</p>
  }
}
```

Example for references:

```
$uses{
     <h3>References:</h3>
     <table class="identref">
          <tr>
                <th class="identfile">File</th>
                <th class="identline">Line</th>
          </tr>
  $refs{
          <tr>
                <td class="identfile $refinvalid">$fileonce</td>
                <td class="identline $refinvalid">$lines</td>
          </tr>
  }
     </table>
     <p>
$occurs references in $filehits files.
     </p>
  $indexwarning{
     <p class="error">One or more files need reindexing</p>
  }
}
```

8.8. Markers for free-text search

The body for free-text search with the *search* script is controlled by the template:

```
'htmlsearch' => 'templates/tmpl/html-search-engine.html'
```

where *engine* is replaced by the name of the search engine, *glimpse* or *swish*. Though they are basically the same, this allows to have the name of the search engine inserted and links to the adequate documentation.

It provides the `<form>` for the free-text query. It uses method `GET` and script *search* is invoked again when the submit button is clicked. It also contains the mask for the results.

8.8.a. Query form

The `<form>` returns the searched string in query variable `_string` (note the underscore), the file name in query variable `_filestring`, the “case sensitive” and “advanced search” check box states in query variables `_casesensitive` and `_advanced` respectively.

Marker	Provides	Replaced by
`$variables{}`[42]	State passing data (version selection) from one script invocation to the other	Concatenation of all expansions for each `'variables'` giving a series of `<input type="hidden">` tags
`$variable` (`$variables{}` *block)*	Variable name	Variable name
`$value` (`$variables{}` *block)*	Variable value	Current value of variable
`$advancedchecked`	Check status of the “advanced search” check box	A ready-to-use attribute for the `<input>` tag reflecting the checked state
`$casesensitivechecked`	Check status of the “case sensitive” check box	A ready-to-use attribute for the `<input>` tag reflecting the checked state
`$searchtext`	The looked-for text; to be used outside the `<form>`	HTML-safe text where the eventual angle brackets `<>` are replaced by HTML character references[43] (see the next one and the note)
`$searchtext_escaped`	The unchanged looked-for text; to be used in an `<input>` field only	Text where all non alphanumeric characters are replaced by their %-encoded equivalent, ready to be transmitted to the LXR server
`$filetext_escaped`	The unchanged looked-for file name; to be used in an `<input>` field only	File name where all non alphanumeric characters are replaced by their %-encoded equivalent, ready to be transmitted to the LXR server

Note:

42 This is not consistent with script *ident* which offers a simple marker instead of this block marker. This should be corrected in a future version after pondering on the best choice (trade-off between ease-of-use and flexibility).

43 This mapping cannot be reversed since no one can tell if `&lt;` was the intended 4-character string or if it was `<`.

There is no `$filetext` intentionally.

CAUTION!

The distinction between $*xxx* and $*xxx*`_escaped` results from the effort to protect against cross-site scripting (XSS) attacks. In the provided template, $*xxx*`_escaped` is locked inside quotation marks, so that a smartly crafted search string including some (JavaScript or other) script functions cannot get executed by your browser. Do not use $*xxx*`_escaped` in any other context.

$*xxx* is secure because the “trigger” characters, the angle brackets (needed to code tags) are replaced by HTML character references which are always considered as pure text.

Example:

```
<form method="get" action="search">
$variables{<input type="hidden" name="$variable" value="$value">}
     <table>
           <tr>
                 <td>
                       <p><b>Files named:</b></p>
                 </td>
                 <td>
                       <input      type="text"
                                   name="_filestring"
                                   value="$filetext_escaped"
                                   size="50"
                       >
                 </td>
           </tr>
           <tr>
                 <td>
                       <p><b>Or containing:</b></p>
                 </td>
                 <td>
                       <input      id="focus"
                                   type="text"
                                   name="_string"
                                   value="$searchtext_escaped"
                                   size="50"
                       >
                 </td>
           </tr>
           <tr>
                 <td></td>
                 <td>
                       <label>
                             <input      type="checkbox"
                                         name="_advanced"
                                         $advancedchecked
                                         value="1"
                             >
     Advanced (allows usage of regex)
                       </label>
     &nbsp;&nbsp;&nbsp;&nbsp;
```

```
                        <label>
                              <input      type="checkbox"
                                          name="_casesensitive"
                                          $casesensitivechecked
                                          value="1"
                              >
     Case-sensitive
                        </label>
                  </td>
                  <td>
     &nbsp;&nbsp;&nbsp;&nbsp;
                        <input type="submit" value="Search">
                  </td>
     </table>
     <br>
</form>
```

8.8.b. Results area

The markers offer two choices of editing a reference as a very simple combined string (but dangling from line to line) or a more flexible layout possibility.

Marker (context)	Provides	Replaced by
`$maxhits_message`	Error indication	Error message if the overflow condition is met, an empty string otherwise
`$resultcount`	Number of hits reported by the search engine (occurrences for ***Glimpse***, files for ***Swish-e***)	Hits number (limited to the overflow condition)
`$results{}`	Repeated application of block to each individual result	Concatenation of all expansions
`$text` (`$results{}` *block)*	Hit text from the search engine	***Glimpse***: line containing the string; ***Swish-e***: relevance factor for the whole file, roughly related to the number of occurrences or “density” of string in file
`$fileref` (`$results{}` *block)*	Occurrence location **choice #1** (historical)	String combining the file name and the line number as a link with this occurrence as target
`$file` or `$fileonce` and `$line` (`$results{}` *block)*	Occurrence location **choice #2**	Respectively by a clickable link to the file and the line number as a link to the occurrence

Marker (context)	Provides	Replaced by
`$tdfile` (`$results{}` *block)*	HTML class attribute allowing to decorate `$fileonce` with CSS styles	String `searchfile` or `searchfilevoid` depending on whether `$fileonce` is a file name or an empty string
`$searchinvalid` (`$results{}` *block)*	HTML class attribute allowing to decorate `$fileonce`, `$file` and `$line` with CSS styles	Expands to `searchinvalid` if file has been modified since last indexation; empty otherwise.

Notes:

1. Overflow is reported if the requested string is too common and results in excessive hits[44]. The maximum number of hits is hard-coded in LXR.
2. `$fileref` string has by nature a varying length. The file name portion is ragged and the line numbers do not line up. The visual effect of choice #1 is poor.
3. Separating file and line number information in choice #2 allows a neatly aligned tabular form.
4. `$fileonce` is the same as `$file` for the first reference in a file, then an empty string for the second and following references.
5. `$line` returns an empty string in the ***Swish-e*** case, since this search engine never returns line numbers.

Example:

```
<p>
     $maxhits_message
     $resultcount occurrences found.
</p>
<h2>Results for <span class="searchident">$searchtext</span></h2>
<table>
     <tbody>
  $results{
           <tr>
                 <td class="$tdfile $searchinvalid">$fileonce</td>
                 <td class="searchline $searchinvalid">$line</td>
                 <td>$text</td>
           </tr>
  }
     </tbody>
</table>
```

[44] This is unlikely to happen with swish-e since it reports only the files containing the requested text, not the occurrences. It may nevertheless happen if your source-tree contains thousands of files.

8.9. *Markers for configuration display*

The body for configuration display with the *showconfig* script is controlled by the template:

```
'htmlconfig' => 'templates/tmpl/html-config.html'
```

The template may provide a title area and should define the elements for the configuration parameters contained in a selected parameter group, usually with a `<table>`.

Marker (context)	Provides	Replaced by
`$conffile`	Name of master configuration file	OS-absolute path of master configuration file
`$virtroot`	Designation of tree under scrutiny	Value of parameter `'virtroot'`
`$parmgroupnr`	Index of parameter group under scrutiny	Index within configuration file (global parameter group is always excluded)
`$previous`	Switch to previous group	<a> link to show previous group
`$next`	Switch to next group	<a> link to show previous group
`$conf_parm{}`	Repeated application of block to each parameter	Concatenation of all expansions
`$parm` (`$conf_parm{}` *block)*	Parameter name	Name
`$type` (`$conf_parm{}` *block)*	*Perl* Parameter type	String equal to `hash`, `array` or `string`
`$val` (`$conf_parm{}` *block)*	Tree-specific value	Ready to use HTML sequence
`$global` (`$conf_parm{}` *block)*	Global value	Ready to use HTML sequence

The template may eventually provide a `<form>` to force display of all known parameters, even if not defined in *lxr.conf*. It uses method `GET` and script *showconfig* is invoked again when the submit button is clicked. The `<form>` returns the “force” flag in query variable `_confall` (note the underscore).

Example:

```
<p>
    Now showing <strong>$virtroot</strong>
    configured in parameter group # $parmgroupnr
</p>

<table>
    <thead>
          <th>Parameter</th>
          <th>Type</th>
          <th>Tree-specific</th>
          <th>Global</th>
    </thead>
$conf_parm{     <tr>
          <td>$parm</td>
          <td>$type</td>
          <td>$val</td>
          <td>$global</td>
    </tr>
}</table>
```

8.10. Markers for custom error page

When something goes wrong during initialization of LXR, if *lxr.conf* does not provide the required parameters, if the source root cannot be found, ... LXR dies silently. Oh well! Not so silently. Anyway, it is not user-friendly. You may apparently stay stuck on your present page or get a totally blank screen (which is worse). Your only resort is to have a look at the server error log. But if the server is not on your computer or you have no administrator rights, you stay in the blue!

The author of this note has a page with links to many trees. In fact, it is rather a catch-all page where all the links are present, whether the tree exists or not, so that the page is written once and not modified every day. But it is very frustrating to be confronted to a white screen instead of getting a proper message when you click on a “waiting” link. You do not get a 404-error because the links drive you into LXR and later LXR discovers that it cannot serve the request (non-existent `sourceroot`) and dies away.

To alleviate this bothersome event, LXR has been modified as of release 0.9.9 to display an error page with the name of the tree. The error page is described by an HTML template:

```
'htmlfatal' => 'templates/tmpl/html-fatal.html'
```

If omitted, a very simple page is internally generated.

Very few markers are available since this a response to a serious configuration mishap:

Marker	Provides	Replaced by
`$target`	Guessed desired tree	Tree name as extracted from the URL by parameter `'treeextract'`
`$stylesheet`	Location of the CSS style	Value of configuration parameter

Marker	Provides	Replaced by
	sheet	`'stylesheet'`
`$LXRversion`	LXR release version id	LXR version as set by script *set-lxr-version.sh*

Example:

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
<title>Error - no tree</title>
<link href="$stylesheet" rel="stylesheet" type="text/css">
</head>
<body>
     <h1 class="main">Unrecoverable error</h1>
     <h2>No known root for source-tree <em>$target</em></h2>
<hr>
<p>Page automatically generated by the $LXRversion LXR engine</p>
</body>
</html>
```

9 Customising LXR appearance

CSS styles allow for a precise control of the layout and display appearance of LXR output.

Composing a style sheet requires CSS knowledge.

When LXR outputs its HTML, it cares to tag important data with a `class="…"` attribute. Style decoration can then be applied with CSS style sheet `'stylesheet'`.

This section lists all CSS classes with their intended usage. A suggested style is given in the parentheses.

This list is not authoritative. You can define additional attributes in your custom templates. Simply, try to avoid conflict with the built-in attribute usages.

9.1. Ubiquitous classes

These so-called classes may be issued by any script under various circumstances. They are not tied to a specific context.

`compact`

Used to tag `<p>` or `<div>` when expecting a very compact layout (reduce margins and paddings).

`error`

Used to tag a message, usually a `<p>`, after an error condition (centred, red).

`warning`
`fatal`

When debugging enabled, used on an `<h4>` to pass debugging output.

9.2. Header and footer

`header`
`footer`

Used to tag a `<table>` as having this meaning (centred).

`leftmost`
`rightmost`

Used to tag the outer `<td>` cells of the `<table>` (respectively left and right aligned).

9.2.a. Titling

`treelink`

Used to tag a `<td>` cell containing links to other trees; marks an `<a>` tag as targeting another tree.

`tree-sel`

Used to tag a `<span>` element containing the name of the currently displayed tree.

`main`

Used to tag an `<h1>` element containing the main title of the tree from marker `$caption` (bigger sized bold characters).

`banner`

Used to tag a `<td>` element containing the path to the file or directory from markers `$banner` and `$target` (centred, moderate sized colour characters).

9.2.b. Mode buttons

`modes`

Used to tag an `<a>` or `<form>` element containing a link to another mode, *i.e.* a different script.

`modes-sel`

Used to tag a `<span>` or `<form>` element containing the name of the current mode.

`modes-dis`

Used to tag a `<span>` or `<form>` element containing the name of a disabled mode.

An example of disabled mode is General search *(free-text search) when no search text engine is present or when tree is stored in a VCS.*

9.2.c. Variable buttons

`varlink`

Used to tag an `<a>` or `<option>` element setting the variable to a different value than the current one.

`var-sel`

Used to tag a `<span>` or `<option>` element containing the current variable value.

9.3. *Directory listing*

`dirdesc`

Used to tag a `<div>` or `<td>` cell containing tentative descriptive text.

`dircontent`

Used to tag the `<table>` containing the directory listing.

`dirheader`

Used on the `<tr>` header line of the previous table (centred).

`dirrow1`
`dirrow2`

Used alternatively (every 3 lines) on the `<tr>` content line of the table (may provide different background colours to subdivide the table in small blocks where the eye can easily follow a line of information).

`diricon`
`dirname`
`dirsize`
`dirtime`
`dirindex`
`dirindexinvalid`
`dirdesc`

Used in the `<td>` row cells for the icon, the name of the file, its size, modification time, indexation time and description.

`dirfile`
`dirfolder`

Used in the `<a>` link to a file or folder.

`desctext`

Used to tag a `<div>` containing descriptive text extracted from *README.** files.

For *README* or *README.txt*, the text itself is contained in a `<p class="lxrdesc">` element to distinguish it from the “See also” plain `<p>` paragraph.

For *README.html*, the element may include a `<span class="lxrshortdesc">` element. If it does not, it is also tagged `lxrlongdesc`.

Note:

Descriptive text for files is contained in plain `<p>` elements, without `class` attribute. This may change in a future version.

9.4. Source text listing

`filecontent`

The text of the source file is edited inside a `<pre class="filecontent">` … `</pre>` block (fixed-pitch font).

`annot-cur`
`annot0`
`annot1`

Used to mark the `<span class= >` … `</span>` block for the line annotation (revision number and author). `annot-cur` highlights the latest revision. `annot0` and `annot1` are used alternatively for other revisions.

`fline`

Used for the source line number in an `<a>` tag, creating an anchor only, not a hyperlink.

`fid`

Used to mark the `<span class= >` … `</span>` block containing a known identifier.

`comment`
`include`
`reserved`
`string`

Used to mark the `<span class= >` … `</span>` block containing the corresponding category fragment of text.

`offshore`

Used for the `<a>` links to `http:`, `mailto:`, … addresses found in comments.

9.5. *Difference markup*

See 9.4 Source text listing for the attributes within the source text.

`diff-fref`

Used in the `<a>` links to the compared files in the header (colour, bold, moderate size).

`diff-mark`

Used in the `<span>` block containing the difference markers (bold, foreground and background colours).

`diff-left`
`diff-both`
`diff-right`

Used in `<span>` blocks to delimit the file portions which differ: in the right, left or both files (background colours).

9.6. *Identifier lookup*

`identident`

Used in a `<span>` block to highlight the looked-up identifier name.

`identdef`
`identref`

Used to tag the `<table>` for definition and use locations respectively.

`identtype`

Used to tag `<th>` and `<td>` cells containing the symbol type in the results table.

`identrel`

Used to tag `<th>` and `<td>` cells containing the symbol extra information in the results table.

`identfile`

Used to tag `<th>` and `<td>` cells containing the file name and `<a>` links to the file in the results table.

`identline`

Used to tag `<th>` and `<td>` cells containing the file line number and `<a>` links to the line in the results table.

`identapprox`

Extra style to augment `identline` when matching against a case-folded version of the identifier (*i.e.* they are case-insensitive).

`identinvalid`

Extra style to augment `identfile` and `identline` when file has been modified since last indexation.

9.7. Free-text search

`searchident`

Used in a `<span>` block to highlight the looked-up text.

`searchref`

Used to tag the `<table>` for the location results.

`searchfile`
`searchfilevoid`

Used to tag `<th>` and `<td>` cells containing the file name and `<a>` links to the file in the results table.

`searchline`

Used to tag `<th>` and `<td>` cells containing the file line number and `<a>` link to the line in the results table.

`searchinvalid`

Extra style to augment `$searchfile`, `$searchfilevoid` and `$searchline` when file has been modified since last indexation.

`searchtext`

Used to tag `<td>` cells containing the line text of a hit.

Used to tag the `<pre>` element, inside the previous cell, containing the hit line text.

Used to tag a `<span>` block within the line surrounding the searched text.

Note:

Due to discrepancy in regular expression syntax between *Perl* and free-text search engines, it is not guaranteed that every hit can be `<span>`-highlighted. However, all occurrences are exhaustively accounted for in the hits.

9.8. Configuration display

`conf_title`

Used to tag a `<table>` or `<div>` containing a title for the page.

`conf_section`

Used to tag a `<table>` describing the parameter group.

`conf_prev`
`conf_next`

Used to tag cells in this table containing hyperlinks to previous or next parameter groups.

`config`

Used to tag a `<table>` showing the parameters.

`conf_parm`

Used to tag the cell for the parameter name.

`conf_type`

Used to tag the cell for the parameter type.

`conf_val`

Used to tag a value cell, whether tree-specific or global.

`conf_force`

Used to tag a parameter name not used in the tree-specific nor in the global group.

10 Using LXR with SCMs

Replacing a plain directory by a version control system repository is simply a matter of adequate lxr.conf *configuration.*

Initial configuration is done as usual through script *configure-lxr.pl* (see 1.3.e Configuration for VCS's).

10.1. Limitations

Since the *source code management* systems do not store source trees as plain files, not all LXR features are available.

- No free-text search with ***Glimpse***
- Very limited free-text search under CVS with ***Swish-e***
- No versioning on directories with some VCS's.
- No display of graphic files

However, you have a drastic improvement on the size of the source tree since the differences between one version and another are very efficiently stored.

10.2. CVS

Prefix `cvs:` designates a CVS repository. If it is stored in `/path/to/cvs/repository` and the CVS module is called `project` then set

```
, 'sourceroot' => 'cvs:/path/to/cvs/repository/project'
, 'sourcerootname' => 'cvs:$v'
```

Change `'range'` parameter so that it becomes a subroutine instead of the explicit version list.

```
'range' =>
     sub   { return
                 ( $files->allreleases($LXR::Common::pathname)
                 , $files->allrevisions($LXR::Common::pathname)
                 )
           }
```

You may eventually keep only the line `allreleases` since the individual file revision numbers are not synchronised in a release.

You may also filter the versions returned to keep a smaller relevant number of these. Change the first line of the subroutine to:

```
'range' =>
     sub   { return    grep {/(some_pattern|head)/}
                 ( $files->allreleases($LXR::Common::pathname)
                 , $files->allrevisions($LXR::Common::pathname)
                 )
           }
```

Note:

some_pattern is a regular expression to match against the revision tags. Since revision tags are arbitrary, there is no general rule. Check the naming convention for the project, *e.g.* by running LXR unfiltered and displaying a file if you have no developer documentation. The `'version'` variable line or menu will list all available versions for THIS file. This will give you an idea on the release identifier structure.

Alternatively, `genxref --allversions` collects a cumulative list of all possible versions in text file *custom.d/CVS**%encoded_virtroot*** unless option `--allversions=noauto` is used. In the file name, ***%encoded_virtroot*** is replaced by the value of parameter `'virtroot'` with any non alphanumeric character %-URL encoded to avoid potential conflict with path separators. The content of this file will help you design a relevant filter.

CAUTION!

Always keep `head` in your filter, otherwise you will not be able to list directories, causing LXR to be totally useless!

You should also set the default version retrieved to a version that really does exist otherwise you will receive errors when generating your index.

A good value for the default version is `head`.

```
'default' => 'head'
```

CAUTION!

You MUST provide a `'default'` value because CVS does not manage directory versions. The above function returns an empty list in the directory case; therefore, LXR cannot recover its first value since it does not exist.

10.3. Git

Prefix `git:` designates a *Git* repository. If it is stored in `/path/to/git/repository` then set

```
, 'sourceroot' => 'git:/path/to/git/repository/.git'
, 'sourcerootname' => 'git:$v'
```

Change `'range'` parameter so that it becomes a subroutine instead of the explicit version list. The subroutine will collect all commit tags from the internal *Git* files:

```
'range' => sub
```

```
     {     my $git_dir = "/path/to/git/repository/.git/refs/tags";
           opendir (DIR, $git_dir)
           || die "cannot opendir $git_dir: $!";
           my @files = grep { -f "$git_dir/$_" } readdir (DIR);
           closedir DIR;
           unshift (@files, "HEAD");
           return sort @files;
     }
```

Note:

HEAD is added to guarantee one version exists at least.

You may eventually consider the branch heads. To add them, modify the subroutine as follows:

```
'range' => sub
     {     my $git_dir = "/path/to/git/repository/.git/refs/tags";
           opendir (DIR, $git_dir)
           || die "cannot opendir $git_dir: $!";
           my @files = grep { -f "$git_dir/$_" } readdir (DIR);
           closedir DIR;
           $git_dir = "/path/to/git/repository/.git/refs/heads";
           opendir (DIR, $git_dir)
           || die "cannot opendir $git_dir: $!";
           my @files = grep { -f "$git_dir/$_" } readdir (DIR);
           closedir DIR;
           unshift (@files, "HEAD");
           return sort @files;
     }
```

You should also set the default version retrieved to a version that really does exist otherwise you will receive errors when generating your index.

A good value for the default version is HEAD.

```
'default' => 'HEAD'
```

CAUTION!

Version names are case-sensitive.

You can limit the amount of service information in file display with two parameters:

```
'sourceparams' =>
     {     'git_annotations' => '1'
     ,     'git_blame'       => '1'
     }
```

'git_annotations' prints the revision tag (at least, some right-end characters) and 'git_blame' the revision author name. 'git_blame' equal to 1 automatically sets 'git_annotations' to 1.

10.4. *Subversion*

Prefix `svn:` designates a *Subversion* repository. If it is stored in `/path/to/svn/repository` then set

```
, 'sourceroot' => 'svn:/path/to/svn/repository'
, 'sourcerootname' => 'svn:$v'
```

Change `'range'` parameter so that it becomes a subroutine instead of the explicit version list. The subroutine will collect all releases from trunk, branches and tags lines from the database:

```
'range' =>
     sub   { return    grep {defined}
                 ( $files->allreleases($LXR::Common::pathname)
                 , $files->allbranches($LXR::Common::pathname)
                 , $files->alltags    ($LXR::Common::pathname)
                 )
           }
```

To decrease the number of displayed versions, you can remove some of the `allxxx` lines provided you leave at least one.

Note:

`allreleases` automatically adds `head` to provide a release number independent access to the latest revision of a file. If you remove `allreleases`, do not forget to list `head`.

A good value for the default version is `head`.

```
'default' => 'head'
```

CAUTION!

Version names are case-sensitive.

You can limit the amount of service information in file display with two parameters:

```
'sourceparams' =>
     {     'svn_annotations' => '1'
     ,     'svn_blame'       => '1'
     }
```

`'svn_annotations'` prints the revision tag and `'svn_blame'` the revision author name. `'git_blame'` equal to 1 automatically sets `'svn_annotations'` to 1.

10.5. *Mercurial*

IMPORTANT NOTICE!

To use a Mercurial repository, you need to configure lxr.conf *as required by this storage engine and also to provide some extension commands to Mercurial. The configurator takes care of both issues, but if you prefer to build manually your own configuration, do not forget*

Mercurial customisation.

Prefix `hg:` designates a *Mercurial* repository. If it is stored in `/path/to/hg/repository` then set

```
, 'sourceroot' => 'hg:/path/to/hg/repository'
, 'sourcerootname' => 'hg:$v'
```

Change `'range'` parameter so that it becomes a subroutine instead of the explicit version list. The subroutine will collect all releases from branches and tags lines from the database:

```
'range' =>
     sub   { return
                 ( $files->allbranches($LXR::Common::pathname)
                 , $files->alltags    ($LXR::Common::pathname)
                 )
           }
```

To decrease the number of displayed versions, you can remove some of the `allxxx` lines provided you leave at least one.

Note:

`alltagss` automatically adds `tip` to provide a release number independent access to the latest revision of a file. If you remove `alltags`, do not forget to list `tip`.

A good value for the default version is `tip`.

```
'default' => 'tip'
```

CAUTION!

Version names are case-sensitive.

You can limit the amount of service information in file display with two parameters:

```
'sourceparams' =>
     {     'hg_annotations' => '1'
     ,     'hg_blame'       => '1'
     }
```

`'hg_annotations'` prints the changeset number and `'hg_blame'` the change author name. `'hg_blame'` equal to 1 automatically sets `'hg_annotations'` to 1.

Since the built-in commands in *Mercurial* do not provide the required information to *LXR*, it is necessary to extend *Mercurial* capabilities. *templates/hg-lxr-ext.py* contains two new commands necessary for LXR operation. This file must be copied into configuration directory *custom.d/* since *Python* requires it to be located in a writeable directory (because it will be compiled).

File *hg.rc* acts as an enabling trigger for the commands. It must also be copied into *custom.d/* where it is customised to reflect your configuration. Two parameters must be adjusted: the file location of the new commands and a security attribute.

1. Command location

   With your favourite text editor, change line

```
lxr-ext = %LXRroot%/%LXRconfdir%/hg-lxr-ext.py
```

where `LXRroot` is your LXR root directory and `LXRconfdir` is *custom.d*. With the examples from 1.2 Create LXR installation directory, this line becomes one of the following:

```
lxr-ext = /home/myself/lxr/custom.d/hg-lxr-ext.py
lxr-ext = /usr/local/share/lxr/custom.d/hg-lxr-ext.py
```

2. Command security attribute

   Mercurial exhibit a defensive behaviour to prevent security breach. New commands are allowed only if the user executing the command and the command file owner match, unless the command file is declared trustworthy. With your favourite text editor, change line

```
users = %LXRconfUser%
```

   to the name of *custom.d/* owner (who is usually the same as the file). This is easy if you installed LXR in your home directory but check if you installed in */usr/local/share*. This leads to one of the following alternatives:

```
users = myself
users = root          # (or something else)
```

10.6. BitKeeper

You are on your own. The present maintainer has not checked this SCM.

BitKeeper is a **proprietary software**. The LXR interface is frozen in its 2005 state more or less.

The prefix for `'sourceroot'` is `bk:`.

Alphabetical Index