Lire Developer’s Manual

Joost van Baal

Egon L. Willighagen

Francis J. Lacoste

Lire Developer's Manual

by Joost van Baal, Egon L. Willighagen, and Francis J. Lacoste

Revision History

Revision 20020214 $Date: 2002/02/10 23:03:49 $
$Id: dev-manual.dbx,v 1.27 2002/02/10 23:03:49 flacoste Exp $

Table of Contents

List of Figures

List of Examples

Preface

Log file analysis is both an essential and tedious part of system administration. It is essential because it's the best way of profiling the usage of the service installed on the network. It's tedious because programs generate a lot of data and tools to report on this data are often unavailable or incomplete. When such tools exist, they are generally specific to one product, which means that you can't compare e.g. your Qmail and Exim mail servers.

Lire is a software package developed by the Stichting LogReport Foundation to generate useful reports from raw log files of various network programs. Multiple programs are supported for various types of network services. Lire also supports various output formats for the generated reports.

What This Book Contains

This book is the *Lire Developer's Manual*. It describes the architecture and design of Lire. It contains comprehensive instructions on how to extend Lire. Its intended audience is system administrators or programmers that want to extend Lire or want to understand its internals.

There is another book, the *Lire User's Manual* which describes how to install, configure and use Lire. Its intended audience is system administrators that want to install and use Lire to gather informations about the services operating on their network.

How Is This Book Organized?

This book is divided in four parts. Part I gives an overview of the architecture and design of Lire.

You will find in Part II informations about how to extend Lire. In this part, you will learn on how to add a new DLF format to Lire, write log file converters and add reports for a superservice. There is also a chapter dedicated to the Report API which can be used to write complex reports.

Part III is targeted at developers that want to participate in Lire's development. It contains informations about CVS access, coding conventions, tools needed to build from CVS, release management and other aspects important to those part of the Lire development team.

Finally, Part IV is a reference section which gives comprehensive details about the various XML formats used by Lire and gives in-depth descriptions of its various APIs.

Conventions Used

If You Don't Find Something In This Manual

You can report typos, incorrect grammar or any other editorial problem to <`bugs@logreport.org`>. We welcome reader's feedback. If you feel that certain parts of this manual aren't clear, are missing

information or lacking in any other aspect, please tell us. Of course, if you feel like writing the missing information yourself, we'll very happily accept your patch. We will make our best effort to improve this manual.

Remember, that there is another manual, the *Lire User's Manual* which contains comprehensive information on how to install, use and configure Lire. It also contains reference informations about all of Lire's standard reports and supported services.

There are various public mailing lists for Lire's users. There is a general users' discussion list where you can find help on how to install and use Lire. You can subscribe to this list by sending an empty email with a subject of *subscribe* to <`questions-request@logreport.org`>. Email for the list should be sent to <`questions@logreport.org`>.

You can keep track of Lire's new release by subscribing to the announcement mailing list. You can subscribe yourself by sending an empty email with a subject of *subscribe* to <`announcement-request@logreport.org`>.

Finally, if you're interested in Lire's development, there is a development mailing list to which you can subscribe by sending an empty email with a subject of *subscribe* to <`development-request@logreport.org`>. Email to the list should be sent to <`development@logreport.org`>.

All posts on these lists are archived on a public website.

I. Lire Architecture

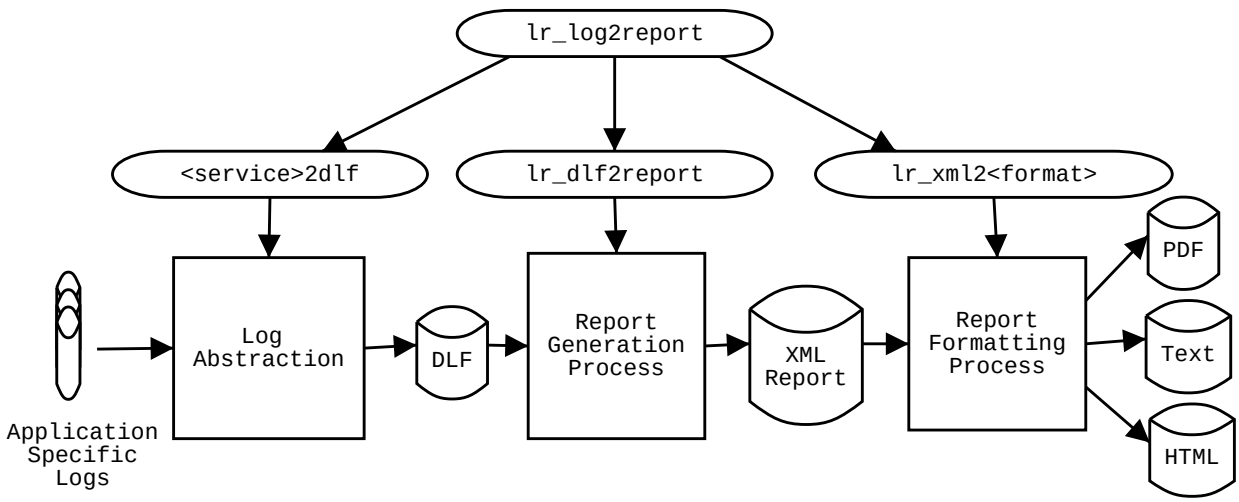
Chapter 1. Architecture Overview

Lire's intend to be the universal log reporting tool. It should be able to process logs from any products and generate useful reports from it. To be useful in the heterogeneous networks that are common nowadays, reports from different products accomplishing similiar functionalities should be comparable. To this end, Lire is designed around a three processes architecture.

Lire's architecture contains three processes:

1. **Log Abstraction.** The first process abstracts logs from different products into a generic format (DLF) that can be shared by all products that have similar functionality. For example, log files from products as different as Apache and Microsoft Internet Information Server will be transformed into an identical format.
2. **Report Generation.** The second process generates a report from the generic log. This report is based on the user's configuration. This process is a generic report engine that compute the report based on specifications that describes the operations that are need to create the report. The report is generated in a generic XML format.
3. **Report Formatting.** The last process converts the generic XML report's format into a more standard output format for human reading like HTML, text or PDF.

Figure 1-1. Lire's Design



The mapping of those three processes can be seen in the Figure 1-1 figure which present the intermediary products of those processes and their mapping to the Lire's component that implement them. Each of these processes is detailed in a later chapter.

Definitions

This section defines more precisely some terms that will be used often in the rest of this manual.

DLF

Example 1-1. DNS DLF Excerpts

```
1010912574 10.0.0.2 121.68.134.195.in-addr.arpa PTR recurs
1010912574 10.0.0.2 121.68.134.195.in-addr.arpa PTR recurs
1010912592 10.0.0.2 120.67.123.212.in-addr.arpa PTR recurs
1010912600 10.0.0.2 207.7.178.212.in-addr.arpa PTR recurs
1010912600 10.0.0.2 tr16.kennisnet.nl A recurs
1010912616 10.0.0.2 120.67.123.212.in-addr.arpa PTR recurs
1010912630 10.0.0.2 207.7.178.212.rbl.maps.vix.com ANY recurs
1010912630 10.0.0.2 NLnet.nl ANY recurs
```

DLF stands for “Distilled Log Format”. This is the generic log format used by Lire to abstract the different products log files. This is a really simple ASCII format where each event is represented by one line. The information about the event is represented by fields separated by spaces. All non-printable ASCII characters are replaced by `?`. Spaces in field’s value are replaced by _ (the underscore). Each lines must have the same number of fields. A DLF file doesn’t contains any header information. Example 1-1 shows an excerpt of a DNS DLF file.

DLF Schema

Information about the order of the fields in a DLF file, their types and what they represent is specified in the DLF’s schema. Schemas are defined in XML files using the Lire DLF Schema Markup Language (LDSML). Lire’s offers an API (only in Perl for now) to programmaticaly access the information of the schema.

It’s the fact that several different products’ log files can share a common DLF schema that makes Lire’s reports easily comparable.

Report

A report is what is generated by Lire. It is made of several subreports. Those subreports can be grouped into sections. The report is computed from the DLF file (and not the native log file) based on a configuration file which describes the subreports that make up the report along with their parameters. (Consult the *Lire User’s Manual* section *Customizing Lire* for more information.)

Service

Put simply, a service is a specific application that produce log. Altough it is usually the case, one application will be equivalent to one service. For example, the mysql service is used to process MySQL's log files.

But more precisely, a service is a specific log format. For example, the common service can be used for all web servers that supports the Common Log Format. Similarly, the welf service can be used to process the firewalls' log files written using WebTrends Enhanced Log Format.

In order to generate a report on it, the native log will be converted to the appropriate superservice's DLF schema

Subreport

A subreport is a particular view on the DLF log's data. Subreports are defined in XML files using the Lire Report Specification Markup Language (LRSML). (Altough it defines subreports, it is called a Report Specification because several subreports makes up the report.) Example of a subreport would be *Requests by Hours of the Day*.

Subreport are defined for a particular DLF schema.

Superservice

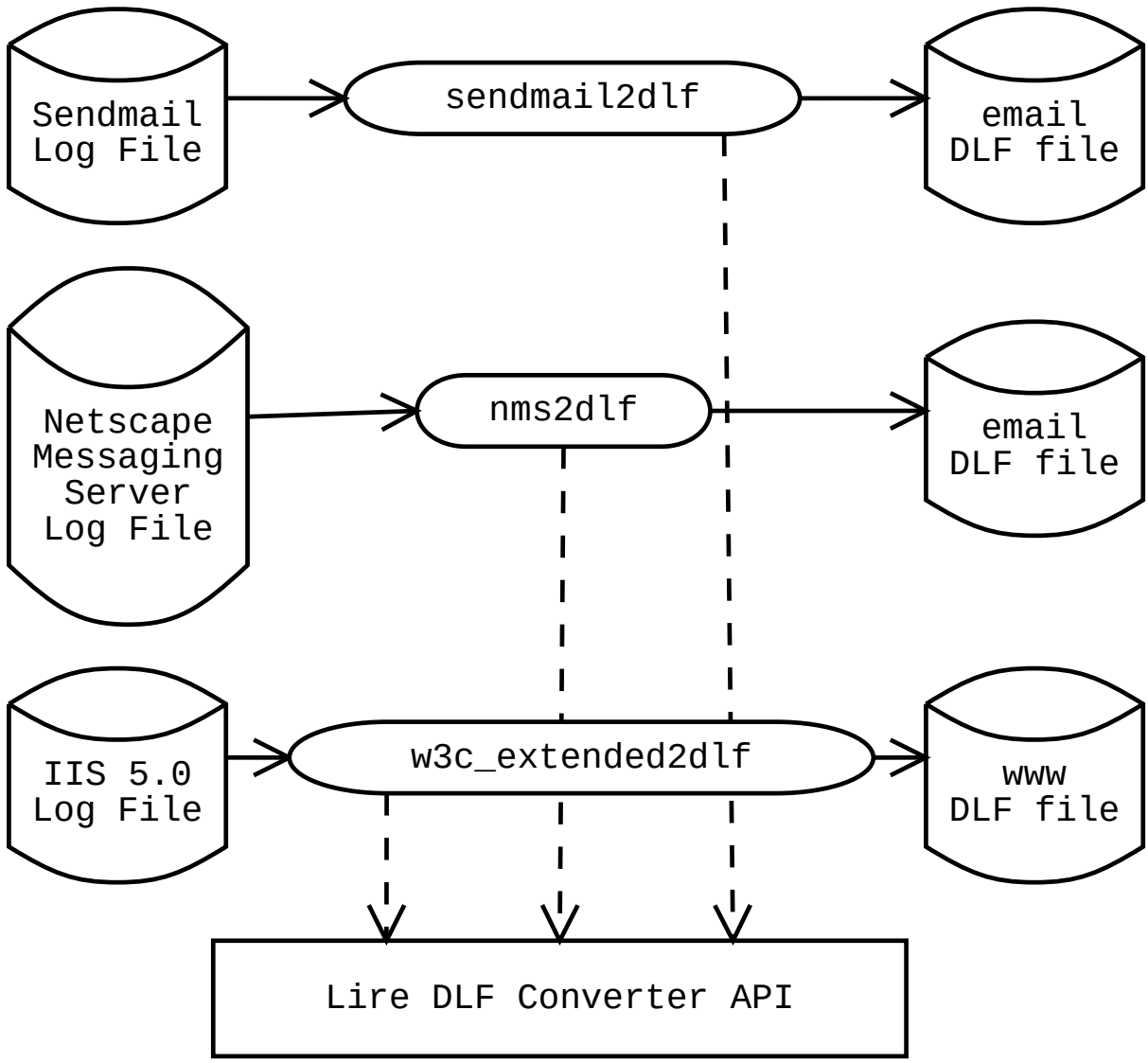
A superservice is a collection of service that shares the same DLF schema and report. It is used to group together applications (services) that offers the same kind of functionalities.

Lire currently supports 8 superservices: database, dns, email, firewall, ftp, print, proxy, and www.

Chapter 2. Logs Abstraction into DLF

Native log files for different applications are converted into the appropriate DLF file through log format specific converter (called DLF converters). Figure 2-1 shows a simple picture of the convertion process. On this diagram, you'll see that the **sendmail2dlf** DLF converter is used to transform a Sendmail's log file into a DLF file using the email DLF schema. Similarly, the convertion processes for the nms and w3c_extended services are depicted.

Figure 2-1. DLF Converter Process



A DLF converter is simply a filter that reads on input the service's native log file and outputs a DLF file in the appropriate superservice's DLF schema. That filter uses the Lire DLF Converter API to generate the DLF file correctly.

Information about writing new DLF converter can be found in Chapter 8.

Chapter 3. Report Generation

Chapter 4. Report Formatting

XSLT Transformation

Perl Text Formatting

Chart Generation

Chapter 5. Responder Architecture

Chapter 6. Source Tree Layout

Service specific scripts should reside in $CVSROOT/service/<service>/script/. Configuration data should be in <service>/etc/. Service specific documentation in <service>/doc/.

Futhermore, in each subdirectory, there should be a Makefile.am.

II. Extending Lire

Chapter 7. Writing a New Superservice

Writing a new superservice involves several things:

1. Making new directories in CVS:
   - `/service/<superservice>/`
   - `/service/<superservice>/script/`
   - `/service/<superservice>/reports/`
2. Adding several files:
   - `/service/<superservice>/Makefile.am`
   - `/service/<superservice>/reports/Makefile.am`
   - `/service/<superservice>/script/Makefile.am`
   - `/service/<superservice>/<superservice>.cfg`
   - `/service/<superservice>/<superservice>.xml` This file specifies the superservice's DLF format. Ideally, it should offer a place for each and every snippet of information which will ever be found in a logfile from a program which offers functionality defined by the superservice.
3. Writing service plugins (2dlf scripts):
   - `/service/<superservice>/script/<service>2dlf.in`
4. Adapting several files:
   - `/service/all/etc/defaults.in` (to add a TODLF declaration)
   - `/service/all/lib/Lire/DataTypes.pm` (adjust the check_superservice function.)
   - `/service/configure.in` (add the Makefiles and 2dlf script to AC_OUTPUT, to get them converted from <service>2dlf.in to <service>2dlf.)
   - `/service/Makefile.am` (add the superservice directory to SUBDIRS, so that make gets run there too, when called from the root source directory.)
5. Update Documentation:
   - User Manual: Chapter "Supported Applications".
   - Add manpages for scripts
6. Update **lr_config**

DLF Design

The DLF Schema

Chapter 8. Writing New Service

Writing a Log File to DLF Converter

API for 2DLF Scripts

Chapter 9. Writing a New Report

Writing a new report involves writing a report specification, e.g. `/service/<superservice>/reports/top-foo-by-bar.xml`, and adding this report along with possible configuration parameters to `<service>/<service>.cfg`. Beware! The name of the report generally consists of alphanumerics and '-', but the name of parameters may *not* contain any '-'. It generally consists of alphanumerics and '_'.

Report Informations

Report's Display Specification

Filter Specification

Calculation Specification

Chapter 10. Writing Advanced Reports

Using a Derived Schema

Writing Extension Reports

III. Lire Developers’ Conventions

Chapter 11. Developers' Toolbox

Required Tools To Build From CVS

In order to be able to build the program from the CVS tree and make a tarball distribution the following tools are needed:

- **docbook2manxml** & **man_xml** from *docbook2x* (http://docbook2x.sourceforge.net/) (v. >0.6.1)
- DocBook XML 4.1.2
- DocBook DSSSL stylesheets
- autotools
- (open)jade
- lynx
- GNU make
- Perl's XML::Parser module

For Debian woody the packages are: docbook-utils (http://packages.debian.org/testing/text/docbook-utils.html), docbook-xml-stylesheets, autoconf (http://packages.debian.org/testing/devel/autoconf.html), automake (http://packages.debian.org/testing/devel/automake.html), autotools-dev (http://packages.debian.org/testing/devel/autotools-dev.html), jade (http://packages.debian.org/testing/text/jade.html), lynx (http://packages.debian.org/testing/web/lynx.html), make (http://packages.debian.org/testing/devel/make.html) and libxml-parser-perl.

Accessing Lire's CVS

Make sure you've got an account on *SourceForge* (http://www.sourceforge.net). Get yourself added to the logreport project. (Joost van Baal joostvb@logreport.org can do this for you.) Make sure your ssh public key is on the sourceforge server.

Weekly, a full backup of the complete LogReport CVS as hosted on SourceForge is made, and written to `hibou:/data/backup/cvs/`.

CVS primer

If you've got a Unix like system, make sure you've got this

```
CVSROOT=:ext:cvs.logreport.sourceforge.net:/cvsroot/logreport
CVS_RSH=ssh
```

in your shell environment.

Of course, you could do something like

```
$ eval ‘ssh-agent‘
$ ssh-add
```

to get a nice ssh-agent running.

Now do something like

```
$ cd ~/cvs-sourceforge/logreport
$ cvs co service
```

There are also repositories called ’docs’ and ’package’. In the former the webpages are located and in the latter the package files for Debian GNU/Linux and other distributions are kept.

Files can then be edited and commited:

```
$ vi somefile
$ cvs commit somefile
```

and get flamed ;)

Subscribe yourself to the commit list (commit-request@logreport.org), to get all commit messages, along with unified diffs.

SourceForge

Mailing Lists

Chapter 12. Coding Standards

Indentation should be four spaces. No tabs please.

Shell Coding Standards

Shell scripts should run -e. Shell script should be portable. Refer to *http://doc.mdcc.cx/doc/autobook/html/autobook_208.html* (http://doc.mdcc.cx/doc/autobook/html/autobook_208.html).

Perl Coding Standards

Perl scripts should use strict, and run -w.

Chapter 13. Commit Policy

Make sure your changes run on your own platform before commiting. Try not to break things for other platform though. Currently, Lire supported platforms are GNU/Linux (Debian GNU/Linux, Red Hat Linux, Mandrake Linux), FreeBSD, OpenBSD and Solaris.

Documentation should be updated ASAP, in case it's obsolete or incomplete by new commits.

CVS Branches

When doing major architectural changes to Lire, branches in CVS are created to make it possible to continue to fix bugs and to add small enhancements to the stable version while development continues on the unstable version. This applies mainly to the service repository. The doc and package repositories generally don't need branching.

Hands-on example

A branching gets announced. Be sure to have all your pending changes commited before the branching occurs. After a branch has been made, one can do this:

```
$ cd ~/cvs-sourceforge/logreport
$ mv service service-HEAD
$ cvs co -r lire-20010924 service
$ mv service service-lire-20010924
```

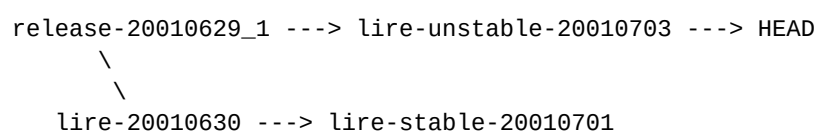
or (with the same result)

```
$ mv service service-HEAD
$ cvs co -r lire-20010924 -d service-lire-20010924 service
```

Now, when working on stuff which should be shipped in the coming release, one should work in service-lire-20010924. When working on stuff which is rather fancy and experimental, and which needs a lot of work to get stabilized, one should work in service-HEAD.

Naming, what it looks like

Here is what branches schematically look like:



In this diagram a branch named `lire-20010630` was created from the `release-20010629_1` tag. `lire-unstable-20010703` is another tag on the *trunk* (the *trunk* is the main branch). `HEAD` isn't a real tag, it always points to latest version on the trunk.

Creating a Branch

To create a branch, one runs the command **`cvs rtag -b -r release-tag branch-name module`**. Note that this command doesn't need a checkout version of the repository. For example, to create the `lire-stable` branch in the service repository, one would use **`cvs rtag -b -r release-20010629_1 lire-stable service`**.

Accessing a Branch

To start working on a particular branch, you do **`cvs update -r branch-name`**. For example, to work on the `lire-stable` branch, you do in your checked out version, **`cvs update -r lire-stable`**. This will update your copy to the version `lire-stable` and will commit all future changes on that branch.

Alternatively, you can also specify a branch when checking out a module using **`cvs co -r branch-name module`**. For example, you could checkout the stable version of Lire by using **`cvs co -r lire-stable service`**.

To see if you are working on a particular branch, you can use the **`cvs status file`** command. For example, running **`cvs status NEWS`** could show :

```
===================================================================
File: NEWS              Status: Up-to-date

   Working revision: 1.74
   Repository revision: 1.74 /cvsroot/logreport/service/NEWS,v
   Sticky Tag: lire-stable
   Sticky Date: (none)
   Sticky Options: (none)
```

The branch is indicated by the `Sticky Tag:` keyword. If its value is `(none)` you are working on the `HEAD`.

To work on the `HEAD`, one removes the sticky tag by using the command **`cvs update -A`**.

Merging Branches on the Trunk

One can bring bug fixes and small enhancements made on a branch into the unstable version on the trunk by doing a merge. You do a merge by using the command **`cvs update -j branch-to-merge`** in your working directory of the trunk. Conflicts are resolved in the usual CVS way. For example, to merge

the changes of the stable branch in the development branch, you would use **`cvs update -j lire-stable`**.

You should tag the branch after each successful merge so that future changes can be easily merged. For example, after merging, you do in a checked out copy of the `lire-stable` branch : **`cvs tag lire-stable-merged-20010715`**. In this way, one week later, we can merge the week's changes of the stable branch into the unstable branch by doing **`cvs update -j lire-stable-merged-20010715 -j lire-stable`**.

Chapter 14. Testing

One week before release, the software should be tested on all supported platforms. In between releases, the system gets tested on various platforms on an ad hoc basis. When testing, use the to-be-released tarball. Run `make dist` to generate such a tarball. Releases are done about every month.

Chapter 15. Making a Release

Before making an official Lire release, it should have been tested on all supported platforms. A release shouldn’t be made unless Lire builds, installs and generates an ASCII report from all supported log files on all supported platforms. If this is not the case, the release should be delayed untill this is fixed.

Making a new release of Lire involves many steps :

1. Writing final version number in NEWS.
2. Tagging the CVS.
3. Building the "Standard" Lire tarball.
4. Building the "Full" Lire tarball.
5. Building the Debian GNU/Linux package.
6. Building the RPM package.
7. Uploading the tarballs and packages available.
8. Advertising the release.

Setting version in NEWS file

In between releases, the NEWS file generally reads "version in cvs". This should of course be changed to e.g. "version 20011205".

Tagging the CVS

Run e.g. **`cvs tag lire-20011017`**.

Building The "Standard" Tarball

The "Standard" tarball is the one that contains only the code needed to build and install Lire. It doesn’t contain required libraries like expat or XML::Parser. There is also a "Full" version of the tarball that includes those libraries.

1. Start from a fresh copy by running the command **`make maintainer-clean-recursive`** in the directory where you checked out Lire’s source code.
    a. Make sure that there are no tarballs in the `extras` subdirectory.
2. Set the version and prepare the source tree by running the command **`./bootstrap`**. (You can overwrite the pre-cooked version by doing e.g. **`echo ‘date +%Y%m%d‘-R-f-jvb-1 > VERSION`** . Make sure your version hasn’t got too many characters. Non-GNU tar chokes on too long pathnames in the archive.)

3. Generate Makefiles

    a. Run **`./configure`**

4. Build Lire and create the tarball by running the command **`make distcheck`**.

    This will build a tarball `lire-`*`version`*`.tar.gz` and then makes sure that the content of this tarball can be build and installed. If that command fails, Lire isn't ready to be released. Fix the errors before making the release.

5. Sign Lire's tarball with you public key. To do this with GnuPG, run **`gpg --detach-sign --armor lire-`*`version`*`.tar.gz`**.

    A file `lire-`*`version`*`.tar.gz.asc` will get created. Publish this file, together with the tarball. Now, people downloading the tarball can verify its integrity by downloading the .asc along with it, as well as your public key, and by running **`gpg --verify lire-`*`version`*`.tar.gz.asc`** .

Building The "Full" Tarball

The "Full" tarball is the one that contains the required Perl and XML libraries along with Lire's source code. This tarball should be called `lire-full-`*`version`*`.tar.gz`.

1. If you have built the "Standard" tarball, you should move it someplace else along with its signature, because this procedure will overwrite it.
2. Start from a fresh copy by running the command **`make maintainer-clean-recursive`** in the directory where you checked out Lire's source code.
3. Add the required libraries' tarball in the `extras` subdirectory. Those tarballs can be downloaded using **wget**.

    a. **`wget http://www.cpan.org/modules/by-module/XML/XML-Parser.2.30.tar.gz`**

    b. **`wget http://prdownloads.sourceforge.net/expat/expat-1.95.2.tar.gz`**

4. Set the version and prepare the source tree by running the command **`./bootstrap`**.
5. Build Lire.

    a. Run **`./configure`**

    b. Run **`make`**

6. Create the tarball by running the command **`make`** followed by the command **`make distcheck`**.

    This will build a tarball and then make sure that the content of this tarball can be build and installed. If that command fails, Lire isn't ready to be released. Fix the errors before making the release.

7. Rename the generated tarball to `lire-full-`*`version`*`.tar.gz`.
8. Sign Lire's tarball with you public key. To do this with GnuPG, run **`gpg --detach-sign --armor lire-full-`*`version`*`.tar.gz`**.

A file `lire-full-`*`version`*`.tar.gz.asc` will get created. Publish this file, together with the tarball. Now, people downloading the tarball can verify its integrity by downloading the .asc along with it, as well as your public key, and by running **`gpg --verify lire-full-`*`version`*`.tar.gz.asc`** .

Building The Debian Package

This is a raw unformatted dump of what we did to build and upload the Lire .deb.

```
              $ cd ~/cvs-sourceforge/logreport/package/debian
              $ vi changelog

:r !date --rfc

              $ cd /usr/local/src/debian/lire/debian/20010219
```

Run 'debian-install-build woody'. This does:

```
              $ cd /usr/local/src/debian/lire/debian/20010219
              $ cp \
  ~/cvs-sourceforge/logreport/service/lire-20010219.tar.gz .

              $ tar zxf lire-20010219.tar.gz
              $ cd lire/20010418
              $ mv lire-20010418 lire-20010418.orig
              $ tar zxf lire-20010418.tar.gz
              $ cd lire-20010418
              $ mkdir debian
              $ cp \
   ~/cvs-sourceforge/logreport/package/debian/[^C]* debian/
```

Export the shell environment variable EMAIL, it should hold your emailaddress, as it is to appear in the package's maintainers field. (One could use 'dh_make --copyright gpl -s' on first time debianizing.) Build the .deb by running:

```
$ debuild 2>&1 | tee /tmp/build
```

Check the .deb:

```
$ debc | less
```

After having *really* tested it (dpkg -i, purge, etc.), optionally install it on any local apt-able websites you might have (Joost has one on http://mdcc.cx/debian/) and upload it to hibou's apt-able archive:

```
$ scp lire_20010418-1_all.deb \
 hibou.logreport.org:/var/www/logreport.org/pub/debian/dists/local/contrib/binary-all/admin
```

```
$ scp lire_20010418*.gz \
 hibou.logreport.org:/var/www/logreport.org/pub/debian/dists/local/contrib/source/admin/
```

On hibou, update the Packages file by running

```
$ cd /var/www/logreport.org/pub/debian
$ make
```

Move the old debian stuff to hibou:/pub/archive/debian/ .

Upload it to the official debian mirrors:

```
vanbaal@gelfand:/usr...src/debian/lire/20010418% date; \
  dupload lire_20010418-1_i386.changes
Thu Apr 19 14:27:38 CEST 2001
Uploading (ftp) to ftp.uk.debian.org:debian/UploadQueue/
[ job lire_20010418-1_i386 from lire_20010418-1_i386.changes New dpkg-dev, announcement wil
 lire_20010418.orig.tar.gz, md5sum ok
 lire_20010418-1.diff.gz, md5sum ok
 lire_20010418-1_all.deb, md5sum ok
 lire_20010418-1.dsc, md5sum ok
 lire_20010418-1_i386.changes ok ]
Uploading (ftp) to uk (ftp.uk.debian.org)
 lire_20010418.orig.tar.gz 163.1 kB , ok (12 s, 13.59 kB/s)
 lire_20010418-1.diff.gz 32.6 kB , ok (3 s, 10.88 kB/s)
 lire_20010418-1_all.deb 222.4 kB , ok (16 s, 13.90 kB/s)
 lire_20010418-1.dsc 0.6 kB , ok (0 s, 0.60 kB/s)
 lire_20010418-1_i386.changes 1.2 kB , ok (1 s, 1.22 kB/s) ]
```

check ftp://ftp.uk.debian.org/debian/UploadQueue/

For a potato release:

```
vanbaal@gelfand:~/cvs-sourceforge/logreport/package% scp \
  ../service/lire-20010626.tar.gz stegun:/usr/local/src/debian/lire/

joostvb@stegun:/usr...sr/local/src/debian/lire% tar zxf \
   lire-20010626.tar.gz
joostvb@stegun:/usr...sr/local/src/debian/lire% mv \
   lire-20010626 lire-20010626.orig
joostvb@stegun:/usr...sr/local/src/debian/lire% tar zxf \
   lire-20010626.tar.gz
joostvb@stegun:/usr...sr/local/src/debian/lire% mkdir \
   lire-20010626/debian


vanbaal@gelfand:~/cvs-sourceforge/logreport/package% scp \
  debian/[^C]* stegun:/usr/local/src/debian/lire/lire-20010626/debian/
vanbaal@gelfand:~/cvs-sourceforge/logreport/package% scp \
  debian-potato/[^C]* stegun:/usr/local/src/debian/lire/lire-20010626/debian/
```

```
joostvb@stegun:/usr...sr/local/src/debian/lire% patch -p0 \
  < lire-20010626/debian/lire-20010626.patch

joostvb@stegun:/usr...ebian/lire/lire-20010626% debuild

-rw-r--r-- 1 joostvb src  14k Jun 28 15:21 lire_20010626-1potato2.diff.gz
-rw-r--r-- 1 joostvb src  625 Jun 28 15:23 lire_20010626-1potato2.dsc
-rw-r--r-- 1 joostvb src 208k Jun 28 15:22 lire_20010626-1potato2_all.deb
-rw-r--r-- 1 joostvb src 1.1k Jun 28 15:23 lire_20010626-1potato2_i386.changes
```

or use the debian-install-build script in cvs-sourceforge/logreport/package.

Building The RPM Package

Uploading The Release

To release a new distribution, publish the tarball on various places, and send an announcement to the <announcement@logreport.org> mailinglist, stating the most interesting new features. Furthermore, add a newsitem to the websites' news list. We'll describe how to upload the tarball to various places.

The LogReport Webserver

Upload the tarball to the LogReport webserver like this.

```
$ scp lire-20001211.tar.gz hibou.logreport.org:/var/www/logreport.org/pub/
```

On hibou, do:

```
               $ cd /var/www/logreport.org/pub
$ chown .www lire-20010525.tar.gz
$ chmod g+w lire-20010525.tar.gz

$ tar zxf lire-20001211.tar.gz
$ rm current && ln -s lire-20001211 current
$ rm current.tar.gz && ln -s lire-20001211.tar.gz current.tar.gz
$ rm -rf lire-20001205
$ mv lire-20001205.tar.gz archive
```

Update the README.txt file: Run

```
$ cd /var/www/logreport.org/pub
```

```
$ ( echo \
  'current is the latest official release'; echo; ls -lF c* ) > README.txt
```

Check the symlink to the documentation stuff in the tarball.

Check if the stuff in http://logreport.org/pub/docs is still up to date.

Advertising The Release

SourceForge

In order to release a distribution on SourceForge (SF), you login with your SF account on the SF website. Once logged in you go to the project webpage (https://sourceforge.net/projects/logreport/) and choose *Admin*. Down at the bottom of that page is a a *[Edit/Add File Releases]* link (click it (https://sourceforge.net/project/admin/editpackages.php?group_id=5049)).

You are able to edit packages, like the Lire package in the LogReport project. To add a new release, choose *[Add Release]*. As a release name uses the date, like 20010407, assign it to the Lire package and then use the *Create This Release* button to makes it effective.

The next page shows 4 steps of which only one (step 2) is not straightforward. In that step you assign files to a release (.tar.gz, .deb, .rpm). These files should be uploaded to SF's Upload anonymous FTP site at ftp://upload.sourceforge.net/incoming/. Make sure the file is placed in the `/incoming` directory. Click *Refresh View* in Step 2 to add the files you uploaded to the FTP site. Check the files belonging to the release and Click *Add Files*. In step 3, set Processor to any. Set file type to .deb and source.gz. Click update/refresh. Step 4: send notice. Done.

Freshmeat.net

On Freshmeat.net, releases are not released, but get announced only. These announcements attract a lot of attention. The webpage for the Lire package can be found at http://freshmeat.net/projects/lire/.

To announce a new release go to Lire - development branch (http://freshmeat.net/branches/14593/) webpage. Choose *Add Release* from the Project pull down menu in the light blue area. The rest is very straightforward.

Chapter 16. Website Maintenance

We give hints on how to upgrade the website: installing stuff from current CVS on http://logreport.org (http://logreport.org/).

If you wanna upload a complete new site:

```
vanbaal@gelfand:~/cvs-sourceforge/docs% tar --exclude CVS -zcf \
  htdocs.tar.gz htdocs

vanbaal@gelfand:~/cvs-sourceforge/docs% scp htdocs.tar.gz \
  hibou.logreport.org:

vanbaal@hibou:~% tar zxf htdocs.tar.gz
vanbaal@hibou:~% mv htdocs logreport.org

vanbaal@hibou:/var/www% rm -rf logreport.org.bak
vanbaal@hibou:/var/www% mv logreport.org logreport.org.bak
vanbaal@hibou:/var/www% mv ~/logreport.org .
vanbaal@hibou:/var/www% mv logreport.org.bak/pub logreport.org/
vanbaal@hibou:/var/www% mv logreport.org.bak/9* logreport.org/

vanbaal@hibou:/var/www% chown -R .www logreport.org
vanbaal@hibou:/var/www% chmod -R g+w logreport.org
```

or, if you've only changed some pages:

```
vanbaal@gelfand:~/cvs-sourceforge/logreport/docs% scp \
  htdocs/developers.phtml htdocs/toolbar.inc htdocs/news.inc \
  hibou.logreport.org:/var/www/logreport.org/
```

Documentation on the LogReport Website

Be sure the links to stuff under `/pub/current` are still alive. E.g. the files `TODO`, `dev-manual.html` and `user-manual.html` are linked to.

Publishing the DTD's

The DTD's are published as HTML on the website by using `hibou:/usr/local/src/dtdparse/dtdparse-2.0b2-LogReportPatched.tar.gz`, which is a patched version of Norman Walsh's dtdparse utility. Before the utility is run, make sure that the

DocBook DTD is not included in the parsing process, because the DocBook DTD should not be published. This is done by changing the line:

```
<!ENTITY % load.docbookx     "INCLUDE"                              >
```

into:

```
<!ENTITY % load.docbookx     "IGNORE"                               >
```

The webpages are then generated with:

```
perl ~/dtdparse-2.0b2-patched/dtdparse.pl --title "XML Lire Report Markup Language" --outpu
perl ~/dtdparse-2.0b2-patched/dtdformat.pl --html lire.xml
```

The resulting `lire` directory can be tar-ed, gziped and unpacked again on hibou in the directory `/var/www/logreport.org/pub/docs/dtd/`.

The other two DTD's are HTML-anized similarly, but keep in mind to change the title when running **dtdparse.pl**.

Chapter 17. Writing Documentation

Documentation of the Lire project is done in DocBook XML 4.1.2. The *Lire User's Manual* has more information about DocBook.

After editing the *Lire Developer's Manual* or the *Lire User's Manual*, you should run **`make check-xml`** to make sure the document is still a valid DocBook document. You should fix any errors before commiting your changes.

If everything went right, documentation is build in txt, tex, html and pdf format by running **`make dist`**, or just **`make`** in `doc/`. We give some hints which might be helpful in case you'll have to build the documentation manually.

To generate PDF:

```
$ jade -t tex -d /path/to/DSSSL/docbook/print/docbook.dsl roadmap.xml
$ pdfjadetex roadmap.tex
```

The last step is actually done two or three times to resolve page numbers.

To generate HTML:

```
$ jade -t sgml -d html.dsl roadmap.xml
```

And now use the `html.dsl` in the `doc/source` directory. (If needed adjust it to reflect the location of your DSSSL stylesheets). Use lynx to generate TXT output from HTML with:

```
$ lynx -nolist -dump roadmap.html > roadmap.txt
```

UML Diagrams

Unified Modelling Language (UML) is a set of definitions on how design diagrams are composed. These diagrams will help to document and understand the internals of Lire, and are used as such in this manual.

UML Editing

Several UML editors are available, but few are open source. Among these are Dia (general diagram editor for Gnome), ArgoUML (written in Java) and UML Modeler (http://uml.sf.net/) (UML specific editor for KDE). The latter was used to draw the diagrams found in CVS `/service/doc/uml-diagrams`.

Diagram Types

UML supports several diagram types. Two important ones are *class diagrams* and *sequence diagrams*. The former is used to depict the relations and associations between classes. Classes can be programs or modules. The latter is used to show how certain tasks are performed in time, and can be used to model the sequence of events.

IV. Developer’s Reference

Chapter 18. Lire DLF Schema Markup Language

Chapter 19. Lire Report Specification Markup Language

Chapter 20. Lire Report Markup Language

Chapter 21. The Lire::Program API

Chapter 22. The DLF Schema API

Chapter 23. The Report Specification API