Sympa
Mailing Lists Management Software
version 3.3.5

Serge Aumont, Olivier Salaün, Christophe Wolfhugel,

15 May 2002

Table des matières

Chapitre 1

Presentation

Sympa is an electronic mailing list manager. It is used to automate list management functions such as subscription, moderation, archive and shared document management. It also includes management functions which would normally require a substantial amount of work (time-consuming and costly for the list owner). These functions include automatic management of subscription renewals, list maintenance, and many others.

Sympa manages many different kinds of lists. It includes a web interface for all list functions including management. It allows a precise definition of each list feature, such as sender authorization, the moderating process, etc. *Sympa* defines, for each feature of each list, exactly who is authorized to perform the relevant operations, along with the authentication method to be used. Currently, authentication can be based on either an SMTP From header, a password, or an S/MIME signature.
Sympa is also able to extract electronic addresses from an LDAP directory or SQL server, and include them dynamically in a list.

Sympa manages the dispatching of messages, and makes it possible to reduce the load on the computer system where it is installed. In configurations with sufficient memory, *Sympa* is especially well adapted to handling large lists : for a list of 20,000 subscribers, it requires less than 6 minutes to send a message to 95 percent of the subscribers, assuming that the network is available (tested on a 300 MHz, 256 MB i386 server with Linux).

This guide covers the installation, configuration and management of the current release (3.3.5) of sympa.

1.1 License

Sympa is free software ; you may distribute it under the terms of the GNU General Public License Version 2[1]

You may make and give away verbatim copies of the source form of this package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.

1.2 Features

Sympa provides all the basic features that any mailing list management robot should include. While most *Sympa* features have their equivalents in other mailing list applications, *Sympa* is unique in including features in a single software package, including :

- **High speed distribution processing** and **load control**. *Sympa* can be tuned to allow the system administrator to control the amount of computer resources used. Its optimized algorithm allows :
  - the use of your preferred SMTP engine, e.g. `sendmail` `sendmail`, `qmail` `qmail` or `postfix` `postfix`
  - tuning of the maximum number of SMTP child processes
  - grouping of messages according to recipients' domains, and tuning of the grouping factor
  - detailed logging
- **Multilingual** messages. The current version of *Sympa* allows the administrator to choose the language catalog at run time. At the present time the *Sympa* robot is available in Chinese, Czech, English, Finnish, French, German, Hungarian, Italian, Polish, Portuguese, Spanish, Romanian. The web interface is available in English, Spanish, French, Chinese, Czech, Hungarian, Italian.
- **MIME support**. *Sympa* naturally respects MIME in the distribution process, and in addition allows list owners to configure their lists with welcome, goodbye and other predefined messages using complex MIME structures. For example, a welcome message can be **i**n multipart/alternative format, using **text/html**, **audio/x-wav** :-), or whatever (Note that *Sympa* commands in multipart messages are successfully processed, provided that one part is **text/plain**).
- The **sending process is controlled** on a per-list basis. The list definition allows a number of different actions for each incoming message. A `private` list is a list where only subscribers can send messages. A list configured using `privateoreditorkey` mode accepts incoming messages from subscribers, but will forward any other (i.e. non-subscriber) message to the editor with a one-time secret numeric key that will be used by the editor to *reject* or *distribute* it. For details about the different sending modes, refer to the `send` parameter (13.3.8, page 123). The sending process configuration (as well as most other list operations) is defined using a **scenario**. Any listmaster can define new scenarios (scenarii) in order to comple-

[1] http ://www.gnu.org/copyleft/gpl.html

ment the 20 predefined configurations included in the distribution.
Example : forward multipart messages to the list editor, while distributing others without requiring any further authorization.

– Privileged operations can be performed by list editors or list owners (or any other user category), as defined in the list `config` file or by the robot administrator, the listmaster, defined in the `/etc/sympa.conf` global configuration file (listmaster can also be defined for a particular virtual robot). Privileged operations include the usual `ADD`, `DELETE` or `REVIEW` commands, which can be authenticated via a one-time password or an S/MIME signature. Any list owner using the `EXPIRE` command can require the renewal of subscriptions. This is made possible by the presence of a subscription date stored in the *Sympa* database.
– textbf Web interface : *WWSympa* is a global Web interface to all *Sympa* functions (including administration). It provides :
  – classification of lists, along with a search index
  – access control to all functions, including the list of lists (which makes WWSympa particularly well suited to be the main groupware tool within an intranet)
  – management of shared documents (download, upload, specific access control for each document)
  – an HTML document presenting each user with the list of her current subscriptions, including access to archives, and subscription options
  – management tools for list managers (bounce processing, changing of list parameters, moderating incoming messages)
  – tools for the robot administrator (list creation, global robot configuration)

  (See 5.1, page 49)
– **RDBMS** : the internal subscriber data structure can be stored in a database or, for compatibility with versions 1.x, in text files. The introduction of databases came out of the *WWSympa* project. The database ensures a secure access to shared data. The PERL database API `DBI/DBD` enables interoperability with various (, , ,). (See ref sec-rdbms, page 57)
– **Virtual robots** : a single *Sympa* installation can provide multiple virtual robots with both email and web interface customization (See 9, page 78).
– : e-mail addresses can be retrieved dynamically from a database accepting SQL queries, or from an LDAP directory. In the interest of reasonable response times, *Sympa* retains the data source in an internal cache controlled by a TTL (Time To Live) parameter. (See ref include-ldap-query, page 117)
– : via uid and emails stored in LDAP Directories. Alternative email addresses, extracted from LDAP directory, may be used to ”unify” subscriptions. (See ref ldap-auth, page 65)
– **Antivirus scanner** : *Sympa* extracts attachements from incoming messages and run a virus scanner on them. Curently working with McAfee/uvscan, Fsecure/fsav, Sophos, AVP and Trend Micro/VirusWall. (See ref antivirus, page 145)
– Inclusion of the subscribers of one list among the subscribers of another. This is real inclusion, not the dirty, multi-level cascading one might otherwise obtain by simply ”subscribing list B to list A”.

1.3 Project directions

Sympa is a very active project : check the release note release note[2]. So it is no longer possible to maintain multiple document about Sympa project direction. Please refer to in-the-futur document[3] for information about project direction.

1.4 History

Sympa development started from scratch in 1995. The goal was to ensure continuity with the TULP list manager, produced partly by the initial author of *Sympa* : Christophe Wolfhugel.

New features were required, which the TULP code was just not up to handling. The initial version of *Sympa* brought authentication, the flexible management of commands, high performances in internal data access, and object oriented code for easy code maintenance.

It took nearly two years to produce the first market releases.

Other date :

- Mar 1999 Internal use of a database (Mysql), definition of list subscriber with external datasource (RDBMS or LDAP).
- Oct 1999 Stable version of WWsympa, introduction of scenarios.
- Feb 2000 Web bounces management
- Apr 2000 Archives search engine and message removal
- May 2000 List creation feature from the web
- Jan 2001 Support for S/MIME (signing and encryption), list setup through the web interface, Shared document repository for each list. Full rewrite of HTML look and feel
- Jun 2001 Auto-install of aliases at list creation time, antivirus scanner plugging
- Jan 2002 Virtual robot, LDAP authentication

1.5 Authors and credits

Christophe Wolfhugel is the author of the first beta version of *Sympa*. He developed it while working for the Institut Pasteur[4].

[2] http ://listes.cru.fr/sympa/release.shtml
[3] http ://www.sympa.org/sympa/direct/in-the-future.html
[4] http ://www.pasteur.fr

Later developments have mainly been driven by the Comité Réseaux des Universités[5] (Olivier Salaün and Serge Aumont), who look after a large mailing list service.

Our thanks to all contributors, including :

- Pierre David, who in addition to his help and suggestions in developing the code, participated more than actively in producing this manual.
- David Lewis who corrected this documentation
- Philippe Rivière for its persevering in *Sympa* tuning with Postfix.
- Raphaël Hertzog (debian), Jerome Marant (debian) and Stéphane Poirey (redhat) for Linux packages.
- Loic Dachary for guiding us through the *GNU Coding Standards*
- Vincent Mathieu, Lynda amadouche, John Dalbec for there integration of LDAP features in *Sympa*.
- Olivier Lacroix, for all his perseverance in bug fixing.
- Hubert Ulliac for search in archive base on marcsearch.pm
- Florent Guilleux who wrote the Task Manager
- Nadia Euzen for developping the antivirus scanner pluggin.
- Fabien Marquois, who introduced many new features such as the digest.
- Valics Lehel, for his Romanian translations
- Vizi Szilard for his Hungarian translations
- Petr Prazak for his Czech translations
- Rodrigo Filgueira Prates for his Portuguese translations
- Lukasz Zalubski for his Polish translations
- Alex Nappa and Josep Roman for their Spanish translations
- Carsten Clasohm and Jens-Uwe Gaspar for their German translations
- Marco Ferrante for his Italian translations
- Tung Siu Fai, Wang Jian and Autrijus Tang for their Chinese translations
- and also : Manuel Valente, Dominique ROUSSEAU, Laurent Ghys, Francois Petillon, Guy Brand, Jean Brange, Fabrice Gaillard, Hervé Maza, Harald Wilhelmi,
- Anonymous critics who never missed a chance to remind us that *smartlist* already did all that better.
- All contributors and beta-testers cited in the `RELEASE_NOTES` file, who, by serving as guinea pigs and being the first to use it, made it possible to quickly and efficiently debug the *Sympa* software.
- Ollivier Robert, Usenet Canal Historique and the good manners guru in the PERL program.
- Bernard Barbier, without whom *Sympa* would not have a name.

We ask all those we have forgotten to thank to accept our apologies and to let us know, so that we can correct this error in future releases of this documentation.

[5] http ://www.cru.fr

1.6 Mailing lists and support

If you wish to contact the authors of *Sympa*, please use the address `sympa-authors@cru.fr`.

There are also a few mailing-lists about *Sympa* [6] :

– `sympa-users@cru.fr` general info list
– `sympa-fr@cru.fr`, for French-speaking users
– `sympa-announce@cru.fr`, *Sympa* announcements
– `sympa-dev@cru.fr`, *Sympa* developers
– `sympa-translation@cru.fr`, *Sympa* translators

To join, send the following message to `sympa@cru.fr` :

`subscribe` *Listname Firstname Name*

(replace *Listname*, *Firstname* and *Name* by the list name, your first name and your family name).

You may also consult the *Sympa* home page, you will find the latest version, FAQ and so on.

[6] http ://listes.cru.fr/wws/lists/informatique/sympa

Chapitre 2

what does *Sympa* consist of ?

2.1 Organization

Here is a snapshot of what *Sympa* looks like once it has settled down on your system. This also illustrates the *Sympa* philosophy, I guess. Almost all configuration files can be defined for a particular list, for a virtual robot or for the whole site.

– `~sympa/`
  The root directory of *Sympa*. You will find almost everything related to *Sympa* under this directory, except logs and main configuration files.
– `~sympa/bin/`
  This directory contains the binaries, including CGI. It also contains the default scenarios, templates and configuration files as in the distribution. `~sympa/bin/` may be completely overwritten by the `make install` `make install` So you must not customize templates and scenarii under `~sympa/bin/`.
– `~sympa/bin/etc/`
  Here *Sympa* stores the default versions of what it will otherwise find in `~sympa/etc/` (task models, scenarios, templates and configuration files, recognized S/Mime certificates).
– `~sympa/etc/`
  This is your site's configuration directory. Consult `~sympa/bin/etc/` when drawing up your own.
– `~sympa/etc/create_list_templates/`
  List templates (suggested at list creation time).
– `~sympa/etc/scenari/`
  This directory will contain your scenarii (or scenarios, if you prefer). If you don't know what the hell a scenario is, refer to 10.810.8, page 88. Thoses scenarii are default scenarii but you may `~sympa/etc/my.domain.org/scenari/` for default scenarii of my.domain.orgvirtual robot and `~sympa/expl/mylist/scenari` for scenarii specific to a particular list

– `~sympa/etc/list_task_models/`
  This directory will store your own list task models (see 10.10, page 92).
– `~sympa/etc/global_task_models/`
  Contains global task models of yours (see 10.10, page 92).
– `~sympa/etc/wws_templates/`
  The web interface (*WWSympa*) is composed of template HTML files parsed by the CGI program. Templates can also be defined for a particular list in `~sympa/expl/mylist/wws_templates/` or in `~sympa/etc/my.domain.org/wws_templates/`
– `~sympa/etc/templates/`
  Some of the mail robot's replies are defined by templates (`welcome.tpl` for SUBSCRIBE). You can overload these template files in the individual list directories or for each virtual robot, but these are the defaults.
– `~sympa/etc/my.domain.org`
  The directory to define the virtual robot my.domain.orgdedicated to managment of all lists of this domain (list describtion of my.domain.orgare stored in `~sympa/expl/my.domain.org`). Thoses directories for virtual robots has the same structure as `~sympa/etc` which is configuration dir of the default robot.
– `~sympa/expl/`
  Sympa's working directory.
– `~sympa/expl/mylist`
  The list directory (refer to 11.2, page 98). Lists stored in this directory are belong the default robot as defined in sympa.conf file, but a list can be stored in `~sympa/expl/my.domain.org/mylist` directory and it is managed by my.domain.orgvirtual robot.
– `~sympa/expl/X509-user-certs`
  The directory where Sympa store all user's certificat
– `~sympa/nls/`
  Internationalization directory. It contains XPG4-compatible message catalogues. *Sympa* has currently been translated into 8 different languages.
– `~sympa/spool/`
  Sympa uses 7 different spools (see 2.4, page 19).
– `~sympa/src/`
  Sympa sources.

2.2 Binaries

– `sympa.pl`
  The main daemon ; it processes commands and delivers messages. Continuously scans the `msg/` spool.
– `sympa_wizard.pl`
  A wizard to edit `sympa.conf` and `wwsympa.conf`. Maybe it is a good idea to run it at the beginning, but thoses file can also be edited with your favorite text editor.
– `wwsympa.fcgi`
  The CGI program offering a complete web interface to mailing lists. It can work in both classical CGI and FastCGI modes, although we recommend FastCGI mode, being up to 10 times faster.

– `bounced.pl`
  This daemon processes bounces (non-delivered messages), looking for the bad addresses. List owners will later access bounce information via *WWSympa*. Continuously scans the `bounce/` spool.
– `archived.pl`
  This daemon feeds the web archives, converting messages to HTML format and linking them. It uses the amazing `MhOnArc`. Continuously scans the `outgoing/` spool.
– `task_manager.pl`
  The daemon which manages the tasks : creation, checking, execution. It regurlaly scans the `task/` spool.
– `queue`
  This small program gets the incoming messages from the aliases and stores them in `msg/` spool.
– `bouncequeue`
  Same as `queue` for bounces. Stores bounces in `bounce/` spool.

2.3 Configuration files

– `sympa.conf`
  The main configuration file. See 4, page 31.
– `wwsympa.conf`
  WWSympa configuration file. See 1.2, page 13.
– `edit_list.conf`
  Defines which parameters/files are editable by owners. See 12.2, page 109.
– `topics.conf`
  Contains the declarations your site's topics (classification in *WWSympa*), along with their titles. A sample is provided in the `sample/` directory of the sympa distribution. See 10.7, page 88.
– `auth.conf`
  Defines sources for LDAP-based authentication.
– `robot.conf`
  It is a subset of `sympa.conf` defining a Virtual robot (one per Virtual robot).

2.4 Spools

See 4.6, page 38 for spool definition in `sympa.conf`.

– `~sympa/spool/auth/`
  For storing messages until they have been confirmed.
– `~sympa/spool/bounce/`
  For storing incoming bouncing messages.
– `~sympa/spool/digest/`
  For storing lists' digests before they are sent.
– `~sympa/spool/expire/`

Used by the expire process.
– `~sympa/spool/mod/`
For storing unmoderated messages.
– `~sympa/spool/msg/`
For storing incoming messages (including commands).
– `~sympa/spool/msg/bad/`
Sympa stores rejected messages in this directory
– `~sympa/spool/task/`
For storing all created tasks.
– `~sympa/spool/outgoing/`
`sympa.pl` dumps messages in this spool to await archiving by `archived.pl`.

Chapitre 3

Installing *Sympa*

Sympa is a program written in PERL. It also calls a short program written in C for tasks which it would be unreasonable to perform via an interpreted language.

3.1 Obtaining *Sympa*, related links

The *Sympa* distribution is available from `http ://listes.cru.fr/sympa/`. All important resources are referenced there :

– sources
– `RELEASE_NOTES`
– .rpm and .deb packages for Linux
– user mailing list (see 1.6, page 16)
– contributions
– ...

3.2 Prerequisites

Sympa installation and configuration are relatively easy tasks for experienced UNIX users who have already installed PERL packages.

Note that most of the installation time will involve putting in place the prerequisites, if they are not already on the system. No more than a handful of ancillary tools are needed, and on recent UNIX systems their installation is normally very straightforward. We strongly advise you to perform installation steps and checks in the order listed below ; these steps will be explained in detail in later sections.

– identification of host system characteristics
– installation of DB Berkeley module (already installed on most UNIX systems)
– installing a (, , or) and creating *Sympa*'s Database. This is required for using the web interface for *Sympa*. Please refers to ¨*Sympa*and its databaseësection (6, page 57).
– installation of CPAN (Comprehensive PERL Archive Network)[1] modules
– creation of a UNIX user

3.2.1 System requirements

You should have a UNIX system that is more or less recent in order to be able to use *Sympa*. In particular, it is necessary that your system have an ANSI C compiler (in other words, your compiler should support prototypes), as well as XPG4-standard NLS (Native Language Support, for languages other than English) extensions.

Sympa has been installed and tested on the following systems, therefore you should not have any special problems :

– Linux (various distributions)
– FreeBSD 2.2.x and 3.x
– Digital UNIX 4.x
– Solaris 2.5 and 2.6
– AIX 4.x
– HP-UX 10.20

Anyone willing to port it to NT ? ;-)

If your UNIX system has a `gencat` `gencat` command as well as `catgets(3)` `catgets(3)` and `catopen(3)` `catopen(3)` functions, it is likely that it has NLS extensions and that these extensions comply with the XPG4 specifications.

Finally, most UNIX systems are now supplied with an ANSI C compiler ; if this is not the case, you can install the `gcc` `gcc` compiler, which you will find on the nearest GNU site, for example in France[2].

To complete the installation, you should make sure that you have a sufficiently recent release of the `sendmail` `sendmail` MTA, i.e. release 8.9.x[3] or a more recent release. You may also use `postfix` `postfix` or `qmail` `qmail`.

[1] http ://www.perl.com/CPAN
[2] ftp ://ftp.oleane.net/pub/mirrors/gnu/
[3] ftp ://ftp.oleane.net/pub/mirrors/sendmail-ucb/

3.2.2 Install Berkeley DB (NEWDB)

UNIX systems often include a particularly unsophisticated mechanism to manage indexed files. This consists of extensions known as `dbm` and `ndbm`, which are unable to meet the needs of many more recent programs, including *Sympa*, which uses the DB package initially developed at the University of California in Berkeley, and which is now maintained by the company Sleepycat software[4]. Many UNIX systems like Linux, FreeBSD or Digital UNIX 4.x have the DB package in the standard version. If not you should install this tool if you have not already done so.

You can retrieve DB on the Sleepycat site[5], where you will also find clear installation instructions.

3.2.3 Install PERL and CPAN modules

To be able to use *Sympa* you must have release 5.004_03 or later of the PERL language, as well as several CPAN modules.

At `make` time, the `check_perl_modules.pl` `check_perl_modules.pl` script is run to check for installed versions of required PERL and CPAN modules. If a CPAN module is missing or out of date, this script will install it for you.

You can also download and install CPAN modules yourself. You will find a current release of the PERL interpreter in the nearest CPAN archive. If you do not know where to find a nearby site, use the CPAN multiplexor[6] ; it will find one for you.

3.2.4 Required CPAN modules

The following CPAN modules required by *Sympa* are not included in the standard PERL distribution. We try to keep this list up to date ; if you have any doubts run the `check_perl_modules.pl` `check_perl_modules.pl` script.

- `DB_File` (v. 1.50 or later)
- `Msgcat`
- `Digest-MD5`
- `MailTools` (version 1.13 o later)
- `IO-stringy`
- `MIME-tools` (may require IO/Stringy)
- `MIME-Base64`
- `CGI`

[4] http ://www.sleepycat.com
[5] http ://www.sleepycat.com/
[6] http ://www.perl.com/CPAN/src/latest.tar.gz

– `File-Spec`

Since release 2, *Sympa* requires an RDBMS to work properly. It stores users' subscriptions and preferences in a database. *Sympa* is also able to extract user data from within an external database. These features require that you install database-related PERL libraries. This includes the generic Database interface (DBI) and a Database Driver for your RDBMS (DBD) :

– `DBI` (DataBase Interface)
– `DBD` (DataBase Driver) related to your RDBMS (e.g. Msql-Mysql-modules for MySQL)

If you plan to interface *Sympa* with an LDAP directory to build dynamical mailing lists, you need to install PERL LDAP libraries :

– `Net : :LDAP` (perlldap).

Passwords in Sympa database can be crypted ; therefore you need to install the following reversible cryptography library :

– `CipherSaber`

For performence concerns, we recommend using *WWSympa* as a persistent CGI, using FastCGI. Therefore you need to install the following Perl module :

– `FCGI`

3.2.5 Create a UNIX user

The final step prior to installing *Sympa* : create a UNIX user (and if possible a group) specific to the program. Most of the installation will be carried out with this account. We suggest that you use the name `sympa` for both user and group.

Numerous files will be located in the *Sympa* user's login directory. Throughout the remainder of this documentation we shall refer to this login directory as `~sympa/`.

3.3 Compilation and installation

Before using *Sympa*, you must customize the sources in order to specify a small number of parameters specific to your installation.

First, extract the sources from the archive file, for example in the `~sympa/src/` direc-

tory : the archive will create a directory named `sympa-3.3.5/` where all the useful files and directories will be located. In particular, you will have a `doc/` directory containing this documentation in various formats ; a `sample/` directory containing a few examples of configuration files ; an `nls/` directory where multi-lingual messages are stored ; and, of course, the `src/` directory for the mail robot and `wwsympa` for the web interface.

Example :

```
# su -
$ gzip -dc sympa-3.3.5.tar.gz | tar xf -
```

Now you can run the installation process :

```
$ ./configure
$ make
$ make install
```

`configure configure` will build the `Makefile` ; it recognizes the following command-line arguments :

– `- -prefix=PREFIX - -prefix=PREFIX`, the *Sympa* homedirectory (default /home/sympa/)
– `--with-bindir=DIR --with-bindir=DIR`, user executables in DIR (default /home/sympa/bin/)
`queue` and `bouncequeue` programs will be installed in this directory. If sendmail is configured to use smrsh (check the mailer prog definition in your sendmail.cf), this should point to `/etc/smrsh`. This is probably the case if you are using Linux RedHat.
– `--with-sbindir=DIR --with-sbindir=DIR`, system admin executables in DIR (default /home/sympa/bin)
– `--with-libexecdir=DIR --with-libexecdir=DIR`, program executables in DIR (default /home/sympa/bin)
– `--with-cgidir=DIR --with-cgidir=DIR`, CGI programs in DIR (default /home/sympa/bin)
– `--with-iconsdir=DIR --with-iconsdir=DIR`, web interface icons in DIR (default /home/httpd/icons)
– `--with-datadir=DIR --with-datadir=DIR`, default configuration data in DIR (default /home/sympa/bin/etc)
– `--with-confdir=DIR --with-confdir=DIR`, Sympa main configuration files in DIR (default /etc)
`sympa.conf` and `wwsympa.conf` will be installed there.
– `--with-exlpdir=DIR --with-exlpdir=DIR`, modifiable data in DIR (default /home/sympa/expl/)
– `--with-libdir=DIR --with-libdir=DIR`, code libraries in DIR (default /home/sympa/bin/)

– `--with-mandir=DIR` `--with-mandir=DIR`, man documentation in DIR (default /usr/local/man/)
– `--with-initdir=DIR` `--with-initdir=DIR`, install System V init script in DIR (default /etc/rc.d/init.d)
– `--with-piddir=DIR` `--with-piddir=DIR`, create .pid files in DIR (default /home/sympa/)
– `--with-perl=FULLPATH` `--with-perl=FULLPATH`, set full path to Perl interpreter (default /usr/bin/perl)
– `--with-openssl=FULLPATH` `--with-openssl=FULLPATH`, set path to OpenSSL (default /usr/local/ssl/bin/openssl)
– `--with-gencat=FULLPATH` `--with-gencat=FULLPATH`, set path to gencat (default /usr/bin/gencat)
– `--with-user=LOGIN` `--with-user=LOGIN`, set sympa user name (default sympa)
Sympa daemons are running under this UID.
– `--with-group=LOGIN` `--with-group=LOGIN`, set sympa group name (default sympa)
Sympa daemons are running under this UID.
– `--with-sendmail_aliases=ALIASFILE` `--with-sendmail_aliases=ALIASFILE`, set aliases file to be used by Sympa (default /etc/mail/sympa_aliases)

– `--with-virtual_aliases=ALIASFILE` `--with-virtual_aliases=ALIASFILE`, set postfix virtual file to be used by Sympa (default /etc/mail/sympa_virtual)

This is used by the `alias_manager.pl` script :
– `--with-newaliases=FULLPATH` `--with-newaliases=FULLPATH`, set path to sendmail newaliases command (default /usr/bin/newaliases)
– `--with-newaliases_arg=ARGS` `--with-newaliases_arg=ARGS`, set arguments to newaliases command (default NONE)
This is used by the `postfix_manager.pl` script :
– `--with-postmap=FULLPATH` `--with-postmap=FULLPATH`, set path to postfix postmap command (default /usr/sbin/postmap)
– `--with-postmap_arg=ARGS` `--with-postmap_arg=ARGS`, set arguments to postfix postmap command (default NONE)

`make` `make` will build a few binaries (`queue`, `bouncequeue` and `aliaswrapper`) and help you install required CPAN modules.

`make install` `make install` does the installation job. It it recognizes the following option :

– DESTDIR, can be set in the main Makefile to install sympa in DESTDIR/DIR (instead of DIR). This is useful for building RPM and DEB packages.

Since version 3.3 of Sympa colors are `sympa.conf` parameters (see 4.1.5, page 32)

If everything goes smoothly, the `~sympa/bin/` directory will contain various PERL programs as well as the `queue` binary. You will remark that this binary has the *set-uid-on-exec* bit (owner is the `sympa` user) : this is deliberate, and indispensable if *Sympa* is to run correctly.

3.3.1 Choosing directory locations

All directories are defined in the `/etc/sympa.conf` file, which is read by *Sympa* at runtime. If no `sympa.conf` file was found during installation, a sample one will be created. For the default organization of directories, please refer to 2.1, page 17.

It would, of course, be possible to disperse files and directories to a number of different locations. However, we recommend storing all the directories and files in the `sympa` user's login directory.

These directories must be created manually now. You can use restrictive authorizations if you like, since only programs running with the `sympa` account will need to access them.

3.4 Robot aliases

An electronic list manager such as *Sympa* is built around two processing steps :

– a message sent to a list or to *Sympa* itself (for subscribe, unsubscribe, help messages, etc.) is received by the SMTP server (`sendmail sendmail` or `qmail qmail`). The SMTP server, on reception of this message, runs the `queue` program (supplied in this package) to store the message in a queue, i.e. in a special directory.
– the `sympa.pl` daemon, set in motion at system startup, scans the queue. As soon as it detects a new message, it processes it and performs the requested action (distribution or processing of an administrative request).

To separate the processing of administrative requests (subscription, unsubscription, help requests, etc.) from the processing of messages destined for mailing lists, a special mail alias is reserved for administrative requests, so that *Sympa* can be permanently accessible to users. The following lines must therefore be added to the `sendmail sendmail` alias file (often `/etc/aliases`) :

```
sympa:              "| /home/sympa/bin/queue sympa"
listmaster:      "| /home/sympa/bin/queue listmaster"
bounce+*:           "| /home/sympa/bin/bouncequeue sympa"
sympa-request:      postmaster
sympa-owner:        postmaster
```

`sympa-request` should be the address of the robot administrator, i.e. a person who looks after *Sympa* (here `postmaster@cru.fr`).

`sympa-owner` is the return address for *Sympa* error messages.

The alias bounce+* is dedicated to collect bounces. It is useful only if at least one list uses `welcome_return_path unique` or `remind_return_path unique`. Don't forget to run `newaliases newaliases` after any change to the `/etc/aliases` file !

Note : aliases based on `listserv` (in addition to those based on `sympa`) can be added for the benefit of users accustomed to the `listserv` and `majordomo` names. For example :

```
listserv:          sympa
listserv-request:  sympa-request
majordomo:         sympa
listserv-owner:    sympa-owner
```

Note : it will also be necessary to add entries in this alias file when lists are created (see list creation section, 11.1, page 97).

3.5 Logs

Sympa keeps a trace of each of its procedures in its log file. However, this requires configuration of the `syslogd` `syslogd` daemon. By default *Sympa* wil use the `local1` facility (`syslog` parameter in `sympa.conf`). WWSympa's logging behaviour is defined by the `log_facility` parameter in `wwsympa.conf` (by default the same facility as *Sympa*).
To this end, a line must be added in the `syslogd` `syslogd` configuration file (`/etc/syslog.conf`). For example :

```
local1.*       /var/log/sympa
```

Then reload `syslogd` `syslogd`.

Depending on your platform, your syslog daemon may use either a UDP or a UNIX socket. *Sympa*'s default is to use a UNIX socket ; you may change this behavior by editing `sympa.conf`'s `log_socket_type` parameter (4.3.3, page 34).

3.6 sympa.pl

Once the files are configured, all that remains is to start *Sympa*. At startup, `sympa.pl` will change its UID to sympa (as defined in `Makefile`). To do this, add the following sequence or its equivalent in your `/etc/rc.local` :

```
~sympa/bin/sympa.pl
~sympa/bin/archived.pl
~sympa/bin/bounced.pl
~sympa/bin/task_manager.pl
```

`sympa.pl` recognizes the following command line arguments :

- `--debug` `--debug` — `-d` `-d`
  Sets *Sympa* in debug mode and keeps it attached to the terminal. Debugging information is output to STDERR, along with standard log information. Each function call is traced. Useful while reporting a bug.
- `--config` *`config_file`* `--config` *`config_file`* — `-f` *`config_file`* `-f` *`config_file`*
  Forces *Sympa* to use an alternative configuration file. Default behavior is to use the configuration file as defined in the Makefile ($CONFIG).
- `--mail` `--mail` — `-m` `-m`
  Sympa will log calls to sendmail, including recipients. Useful for keeping track of each mail sent (log files may grow faster though).
- `--lang` *`catalog`* `--lang` *`catalog`* — `-l` *`catalog`* `-l` *`catalog`*
  Set this option to use a language catalog for *Sympa*. The corresponding catalog file must be located in `~sympa/nls` directory.
  For example, with the `fr.cat` catalog :
- `--keepcopy` *`recipient_directory`* `--keepcopy` *`recipient_directory`* — `-k` *`recipient_directory`* `-k` *`recipient_directory`*
  This option tells Sympa to keep a copy of every incoming message, instead of deleting them. *recipient_directory* is the directory to store messages.

```
/home/sympa/bin/sympa.pl
```

- `--dump` *`listname | ALL`* `--dump` *`listname | ALL`*
  Dumps subscribers of a list or all lists. Subscribers are dumped in `subscribers.db.dump`.
- `--import` *`listname`* `--import` *`listname`*
  Import subscribers in the *listname* list. Data are read from STDIN.
- `--lowercase` `--lowercase`
  Lowercases e-mail addresses in database.
- `--help` `--help` — `-h` `-h`
  Print usage of sympa.pl.
- `--version` `--version` — `-v` `-v`
  Print current version of *Sympa*.

Chapitre 4

sympa.conf parameters

The `/etc/sympa.conf` configuration file contains numerous parameters which are read on start-up of *Sympa*. If you change this file, do not forget that you will need to restart *Sympa* afterwards.

The `/etc/sympa.conf` file contains directives in the following format :

keyword value

Comments start with the `#` character at the beginning of a line. Empty lines are also considered as comments and are ignored. There should only be one directive per line, but their order in the file is of no importance.

4.1 Site customization

4.1.1 `domain`

This keyword is **mandatory**. It is the domain name used in the `From:` header in replies to administrative requests. So the smtp engine (qmail, sendmail, postfix or whatever) must recognize this domain as a local adress. The old keyword `host` is still recognized but should not be used anymore.

Example: `domain cru.fr`

4.1.2 email

(Default value: sympa)

Username (the part of the address preceding the @ sign) used in the From: header in replies to administrative requests.

Example: email listserv

4.1.3 listmaster

The list of e-mail addresses of listmasters (users authorized to perform global server commands). Listmaster can be defined for each virtual robot.

Example: listmaster postmaster@cru.fr,root@cru.fr

4.1.4 wwsympa_url

(Default value: http ://<host>/wws)

This is the root URL of *WWSympa*.

Example: wwsympa_url https ://my.server/wws

4.1.5 dark_color light_color text_color bg_color error_color selected_color shaded_color

They are the color definition for web interface. Default are set in the main Makefile. Thoses parameters can be overwritten in each virtual robot definition.

4.1.6 cookie

This string is used to generate MD5 authentication keys. It allows generated authentication keys to differ from one site to another. It is also used for reversible encryption of user passwords stored in the database. The presence of this string is one reason why access to sympa.conf needs to be restricted to the Sympa user.

Note that changing this parameter will break all http cookies stored in users' browsers,

as well as all user passwords and lists X509 private keys.

Example: `cookie gh869jku5`

4.1.7 create_list

(Default value: `public_listmaster`)

`create_list` parameter is defined by scenario (see 10.8, page 88)

Defines who can create lists (or request list creations). Sympa will use the corresponding scenario.

Example: `create_list intranet`

4.1.8 global_remind

(Default value: `listmaster`)

`global_remind` parameter is defined by scenario (see 10.8, page 88)

Defines who can run a `REMIND *` command.

4.2 Directories

4.2.1 home

(Default value: `~sympa/expl`)

The directory whose subdirectories correspond to the different lists.

Example: `home /home/sympa/expl`

4.2.2 etc

(Default value: `~sympa/etc`)

This is the local directory for configuration files (such as `edit_list.conf`. It contains 5 subdirectories : `scenari` for local scenarii ; `templates` for the site's local templates and default list templates ; `wws_templates` for the site's local html templates ; `global_task_models` for local global task models ; and `list_task_models` for local list task models

Example: `etc /home/sympa/etc`

4.3 System related

4.3.1 `syslog`

(Default value: `LOCAL1`)

Name of the sub-system (facility) for logging messages.

Example: `syslog LOCAL2`

4.3.2 `log_level`

(Default value: `0`)

This parameter sets the verbosity of Sympa processes (including) in log files. With level 0 only main operations are logged, in level 3 almost everything is logged.

Example: `log_level 2`

4.3.3 `log_socket_type`

(Default value: `unix`)

Sympa communicates with `syslogd` `syslogd` using either UDP or UNIX sockets. Set `log_socket_type` to `inet` to use UDP, or `unix` for UNIX sockets.

4.3.4 `pidfile`

(Default value: `~sympa/sympa.pid`)

The file where the `sympa.pl` daemon stores its process number. Warning : the `sympa` user must be able to write to this file, and to create it if it doesn't exist.

Example: `pidfile /var/run/sympa.pid`

4.3.5 `umask`

(Default value: `027`)

Default mask for file creation (see `umask` `umask`(2)).

Example: `umask 007`

4.4 Sending related

4.4.1 `maxsmtp`

(Default value: `20`)

Maximum number of SMTP delivery child processes spawned by *Sympa*. This is the main load control parameter.

Example: `maxsmtp 500`

4.4.2 `log_smtp`

(Default value: `off`)

Set logging of each MTA call. Can be overwritten by -m sympa option.

Example: `log_smtp on`

4.4.3 `max_size`

(Default value: `5 Mb`)

Maximum size allowed for messages distributed by *Sympa*. This may be customized

per virtual robot or per list by setting the `max_size` robot or list parameter.

Example: `max_size 2097152`

4.4.4 misaddressed_commands

(Default value: `reject`)

When a robot command is sent to a list, by default Sympa reject this message. This feature can be turned off setting this parameter to `ignore`.

4.4.5 misaddressed_commands_regexp

(Default value: `(subscribe|unsubscribe|signoff)`)

This is the Perl regular expression applied on messages subject and body to detect misaddressed commands, see `misaddressed_commands` parameter above.

4.4.6 nrcpt

(Default value: 25)

Maximum number of recipients per `sendmail sendmail` call. This grouping factor makes it possible for the (`sendmail sendmail`) MTA to optimize the number of SMTP sessions for message distribution.

4.4.7 avg

(Default value: 10)

Maximum number of different internet domains within addresses per `sendmail sendmail` call.

4.4.8 sendmail

(Default value: `/usr/sbin/sendmail`)

Absolute call path to SMTP message transfer agent (`sendmail sendmail` for example).

Example: `sendmail /usr/sbin/sendmail`

4.4.9 sendmail_args

(Default value: `-oi -odi -oem`)

Arguments passed to SMTP message transfer agent

4.4.10 rfc2369_header_fields

(Default value: `help,subscribe,unsubscribe,post,owner,archive`)

RFC2369 compliant header fields (List-xxx) to be added to distributed messages. These header-fields should be implemented by MUA's, adding menus.

4.4.11 remove_headers

(Default value: `Return-Receipt-To,Precedence,X-Sequence,Disposition-Notification-To`)

This is the list of headers that *Sympa* should remove from outgoing messages. Use it, for example, to ensure some privacy for your users by discarding anonymous options. It is (for the moment) site-wide. It is applied before the *Sympa*, rfc2369_header_fields, and custom_header fields are added.

Example: `remove_headers Resent-Date,Resent-From,Resent-To,Resent-Message-Id,Sender,Delivered-To,`

4.4.12 anonymous_headers_fields

(Default value: `Sender,X-Sender,Received,Message-id,From,X-Envelope-To,Resent-From,Reply-To,Organiz`

This parameter defines the list of SMTP header fields that should be removed when a mailing list is setup in anonymous mode (see 13.4.3, page 127).

4.4.13 list_check_smtp

(Default value: NONE)

If this parameter is set with a SMTP server address, *Sympa* will check if alias with the same name as the list you're gonna create already exists on the SMTP server. It is robot specific, i.e. you can specify a different SMTP server for every virtual robot you are running. This is needed if you are running *Sympa* on somehost.foo.org, but you handle all your mail on a separate mail relay.

4.4.14 list_check_suffixes

(Default value: request,owner,unsubscribe)

This paramater is a comma-separated list of admin suffixes you're using for *Sympa* aliases, i.e. mylist-request, mylist-owner etc... This parameter is used with list_check_smtp parameter.

4.5 Quotas

4.5.1 default_shared_quota

The default disk quota for lists' document repository.

4.5.2 default_archive_quota

The default disk quota for lists' web archives.

4.6 Spool related

4.6.1 spool

(Default value: ~sympa/spool)

The parent directory which contains all the other spools.

4.6.2 queue

The absolute path of the directory which contains the queue, used both by the `queue` program and the `sympa.pl` daemon. This parameter is mandatory.

Example: `queue /home/sympa/queue`

4.6.3 queuemod

(Default value: `~sympa/spool/moderation`)

This parameter is optional and retained solely for backward compatibility.

4.6.4 queuedigest

(Default value: `~digest`)

This parameter is optional and retained solely for backward compatibility.

4.6.5 queueexpire

(Default value: `~sympa/spool/expire`)

This parameter is optional and retained solely for backward compatibility.

4.6.6 queueauth

(Default value: `~sympa/spool/auth`)

This parameter is optional and retained solely for backward compatibility.

4.6.7 queueoutgoing

(Default value: `~sympa/spool/outgoing`)

This parameter is optional and retained solely for backward compatibility.

4.6.8 queuebounce

(Default value: ~sympa/spool/bounce)

Spool to store bounces (non-delivery reports) received by the bouncequeue program via the mylist-owner or bounce+* addresses . This parameter is mandatory and must be an absolute path.

4.6.9 queuetask

(Default value: ~sympa/spool/task)

Spool to store task files created by the task manager. This parameter is mandatory and must be an absolute path.

4.6.10 tmpdir

(Default value: ~sympa/spool/tmpdir)

Temporary directory used by OpenSSL and antiviruses.

4.6.11 sleep

(Default value: 5)

Waiting period (in seconds) between each scan of the main queue. Never set this value to 0 !

4.6.12 clean_delay_queue

(Default value: 1)

Retention period (in days) for “bad” messages in spool (as specified by queue). *Sympa* keeps messages rejected for various reasons (badly formatted, looping, etc.) in this directory, with a name prefixed by BAD. This configuration variable controls the number of days these messages are kept.

Example: clean_delay_queue 3

4.6.13 clean_delay_queuemod

(Default value: 10)

Expiration delay (in days) in the moderation spool (as specified by queuemod). Beyond this deadline, messages that have not been processed are deleted. For moderated lists, the contents of this spool can be consulted using a key along with the MODINDEX command.

4.6.14 clean_delay_queueauth

(Default value: 3)

Expiration delay (in days) in the authentication queue. Beyond this deadline, messages not enabled are deleted.

4.7 Internationalization related

4.7.1 msgcat

(Default value: ~sympa/nls)

The location of multilingual (nls) catalog files. Must correspond to ~src/nls/Makefile.

4.7.2 lang

(Default value: us)

This is the default language for *Sympa*. The message catalog (.cat) located in the corresponding msgcat directory will be used.

4.8 Bounce related

4.8.1 bounce_warn_rate

(Default value: 30)

Site default value for bounce. The list owner receives a warning whenever a message is distributed and the number of bounces exceeds this value.

4.8.2 bounce_halt_rate

(Default value: 50)

FOR FUTURE USE

Site default value for bounce. Messages will cease to be distributed if the number of bounces exceeds this value.

4.8.3 welcome_return_path

(Default value: owner)

If set to string unique, sympa will use a unique e-mail address in the return path, prefixed by bounce+, in order to remove the corresponding subscriber. Requires the bounced daemon to run and bounce+* alias to be installed (plussed aliases as in sendmail 8.7 and later).

4.8.4 remind_return_path

(Default value: owner)

Like welcome_return_path, but relates to the remind message. Also requires the bounce+* alias to be installed.

4.8.5 expire_bounce_task

(Default value: daily)

This parameter tells what task will be used by `task_manager.pl` to perform bounces expiration. This task resets bouncing information for addresses not bouncing in the last 10 days after the latest message distribution.

4.9 Priority related

4.9.1 `sympa_priority`

(Default value: `1`)

Priority applied to *Sympa* commands while running the spool.

Available since release 2.3.1.

4.9.2 `request_priority`

(Default value: `0`)

Priority for processing of messages for mylist-request, i.e. for owners of the list.

Available since release 2.3.3

4.9.3 `owner_priority`

(Default value: `9`)

Priority for processing messages for mylist-owner in the spool. This address will receive non-delivery reports (bounces) and should have a low priority.

Available since release 2.3.3

4.9.4 `default_list_priority`

(Default value: `5`)

Default priority for messages if not defined in the list configuration file.

Available since release 2.3.1.

4.10 Database related

The following parameters are needed when using an RDBMS, but otherwise are not required :

4.10.1 db_type

`Format : db_type mysql | Pg | Oracle | Sybase`

Database management system used (e.g. MySQL, Pg, Oracle)

This corresponds to the PERL DataBase Driver (DBD) name and is therefore case-sensitive.

4.10.2 db_name

(Default value: `sympa`)

Name of the database containing user information. See detailed notes on database structure, **??**, page **??**.

4.10.3 db_host

Database host name.

4.10.4 db_port

Database port.

4.10.5 db_user

User with read access to the database.

4.10.6 db_passwd

Password for db_user.

4.10.7 db_options

If these options are defined, they will be appended to the database connect string.

Example for MySQL :

```
db_options mysql_read_default_file=/home/joe/my.cnf
```

4.10.8 db_env

Sets a list of environment variables to set before database connexion. This is a ';' separated list of variable assignment.

Example for Oracle :

```
db_env ORACLE_TERM=vt100;ORACLE_HOME=/var/hote/oracle/7.3.4
```

4.10.9 db_additional_subscriber_fields

If your **subscriber_table** database table has more fields than required by *Sympa* (because other softwares work this set of data), you can make *Sympa* load these fields. Therefore, you can use them from within mail/web templates and scenario (as [subscriber->field]).

This parameter is a comma-separated list.

Example :

```
db_additional_subscriber_fields  billing_delay,subscription_expiration
```

4.10.10 db_additional_user_fields

If your **user_table** database table has more fields than required by *Sympa* (because other softwares work this set of data), you can make *Sympa* load these fields. Therefore, you can use them from within mail/web templates (as [user->field]).

This parameter is a comma-separated list.

Example :

```
db_additional_user_fields  address,gender
```

4.11 Loop prevention

The following define your loop prevention policy for commands. (see 10.9, page 91)

4.11.1 loop_command_max

(Default value: 200)

The maximum number of command reports sent to an e-mail address. When it is reached, messages are stored with the BAD prefix, and reports are no longer sent.

4.11.2 loop_command_sampling_delay

(Default value: 3600)

This parameter defines the delay in seconds before decrementing the counter of reports sent to an e-mail address.

4.11.3 loop_command_decrease_factor

(Default value: 0.5)

The decrementation factor (from 0 to 1), used to determine the new report counter after expiration of the delay.

4.12 S/MIME configuration

Sympa can optionally verify and use S/MIME signatures for security purposes. In this case, the three first following parameters must be assigned by the listmaster (see 8.4.2, page 75). The two others are optionnals.

4.12.1 openssl

The path for the openSSL binary file.

4.12.2 trusted_ca_options

The option used by OpenSSL for trusted CA certificates. Required if cfkeyword openSSL is defined.

4.12.3 key_passwd

The password for list private key encryption. If not defined, *Sympa* assumes that list private keys are not encrypted.

4.12.4 chk_cert_expiration_task

States the model version used to create the task which regurlaly checks the certificate expiration dates and warns users whose certificate have expired or are going to. To know more about tasks, see 10.10, page 92.

4.12.5 crl_update_task

Specifies the model version used to create the task which regurlaly updates the certificate revocation lists.

4.13 Antivirus plug-in

Sympa can optionally check incoming messages before delivering them, using an external antivirus solution. You must then set two parameters.

4.13.1 antivirus_path

The path to your favorite antivirus binary file (including the binary file).

Example :

```
antivirus_path /usr/local/bin/uvscan
```

4.13.2 antivirus_args

The arguments used by the antivirus software to look for viruses. You must set them so as to get the virus name. You should use, if available, the 'unzip' option and check all extensions.

Example with uvscan :

```
antivirus_args --summary --secure
```

Example with fsav :

```
antivirus_args --dumb --archive
```

Exemple with AVP :

```
antivirus_path  /opt/AVP/AvpLinux
antivirus_args  -Y -O- -MP -I0
```

Exemple with Sophos :

```
antivirus_path  /usr/local/bin/sweep
antivirus_args  -nc -nb -ss -archive
```

Chapitre 5

WWSympa

WWSympa is *Sympa*'s web interface.

5.1 Organization

WWSympa is fully integrated with *Sympa*. It uses `sympa.conf` and *Sympa*'s libraries. The default *Sympa* installation will also install WWSympa.

Every single piece of HTML in *WWSympa* is generated by the CGI code using template files (See 10.1, page 81). This facilitates internationalization of pages, as well as per-site customization.

The code consists of one single PERL CGI script, `WWSympa.fcgi`. To enhance performance you can configure *WWSympa* to use FastCGI ; the CGI will be persistent in memory.
All data will be accessed through the CGI, including web archives. This is required to allow the authentication scheme to be applied systematically.

Authentication is based on passwords stored in the database table user_table ; if the appropriate `Crypt : :CipherSaber` is installed, password are encrypted in the database using reversible encryption based on RC4. Otherwise they are stored in clear text. In both cases reminding of passwords is possible. To keep track of authentication information *WWSympa* uses HTTP cookies stored on the client side. The HTTP cookie only indicates that a specified e-mail address has been authenticated ; permissions are evaluated when an action is requested.

The same web interface is used by the listmaster, list owners, subscribers and others. Depending on permissions, the same URL may generate a different view.

WWSympa's main loop algorithm is roughly the following :

1. Check authentication information returned by the HTTP cookie
2. Evaluate user's permissions for the requested action
3. Process the requested action
4. Set up variables resulting from the action
5. Parse the HTML template files

5.2 HTTPD setup

5.2.1 wwsympa.fcgi access permissions

Because Sympa and WWSympa share a lot of files, `wwsympa.fcgi`, must run with the same uid/gid as `archived.pl`, `bounced.pl` and `sympa.pl`. There are different ways to organize this :

– With some operating systems no special setup is required because wwsympa.fcgi is installed with suid and sgid bits, but this will not work if suid scripts are refused by your system.
– Run a dedicated Apache server with sympa.sympa as uid.gid (The Apache default is nobody.nobody)
– Use a virtual Apache server with sympa.sympa as uid.gid ; Apache needs to be compiled with suexec. Be aware that the Apache suexec usually define a lowest UID/GID allowed to be a target user for suEXEC. For most systems including binaries distribution of Apache, the default value 100 is common. So Sympa UID (and Sympa GID) must be higher then 100 or suexec must be tuned in order to allow lower UID/GID. Check http ://httpd.apache.org/docs/suexec.html#install for details
– Otherwise, you can overcome restrictions on the execution of suid scripts by using a short C program, owned by sympa and with the suid bit set, to start `wwsympa.fcgi`. Here is an example (with no guarantee attached) :

```
#include <unistd.h>

#define WWSYMPA "/home/sympa/bin/wwsympa.fcgi"

int main(int argn, char **argv, char **envp) {
    argv[0] = WWSYMPA;
    execve(WWSYMPA,argv,envp);
}
```

5.2.2 Installing wwsympa.fcgi in your Apache server

If you chose to run `wwsympa.fcgi` as a simple CGI, you simply need to script alias it.

```
Example :
   ScriptAlias /wws /home/sympa/bin/wwsympa.fcgi
```

Running FastCGI will provide much faster responses from your server and reduce load (to understand why, read http ://www.fastcgi.com/fcgi-devkit-2.1/doc/fcgi-perf.htm)

```
     Example :
FastCgiServer /home/sympa/bin/wwsympa.fcgi -processes 2
<Location /wws>
      SetHandler fastcgi-script
</Location>

ScriptAlias /wws /home/sympa/bin/wwsympa.fcgi
```

If you run Virtual robots, then the FastCgiServer(s) can serve multiple robots. Therefore you need to define it in the common section of your Apache configuration file.

5.2.3 Using FastCGI

FastCGI is an extention to CGI that provides persistency for CGI programs. It is extemely usefull with *WWSympa* because it all the intialisations are only performed once, at server startup ; then file wwsympa.fcgi instances are awaiting clients requests.

WWSympa can also work without FastCGI, depending on **use_fast_cgi** parameter (see 5.3.15, page 54).

To run *WWSympa* with FastCGI, you need to install :
– mod_fastcgi : the Apache module that provides FastCGI features
– `FCGI` : the Perl module used by *WWSympa*

5.3 wwsympa.conf parameters

5.3.1 alias_manager

If this parameter is undefined, then you will have to manage your aliases manually. Provide the path to a script that will install aliases for a new list and delete aliases for closed lists. You can use the one of the following scripts distributed with *Sympa* : `~sympa/bin/alias_manager.pl` for sendmail-style aliases with a single aliases file or `~sympa/bin/postfix_manager.pl` for postfix-like aliases using an additional .

Theses expect the following arguments :

1. add — del
2. <list name>
3. <list domain>

Example : ~sympa/bin/alias_manager.pl add mylistcru.fr

~sympa/bin/alias_manager.pl works on the alias file as defined by the variable in the main Makefile (see 3.3, page 25). It runs a newaliases newaliases command (via aliaswrapper), after any changes to aliases file.

~sympa/bin/postfix_manager.pl also requires variable to be defined in the Makefile. It runs a postmap postmap command (via virtualwrapper), after any changes to virtualtable file.

5.3.2 arc_path

(Default value: /home/httpd/html/arc)
Where to store html archives. This parameter is used by the archived.pl daemon. It is a good idea to install the archive outside the web hierarchy to prevent possible back doors in the access control powered by WWSympa. However, if Apache is configured with a chroot, you may have to install the archive in the Apache directory tree.

5.3.3 archive_default_index thrd — mail

(Default value: thrd)
The default index organization when entering web archives : either threaded or chronological order.

5.3.4 archived_pidfile

(Default value: archived.pid)
The file containing the PID of archived.pl.

5.3.5 bounce_path

(Default value: /var/bounce)
Root directory for storing bounces (non-delivery reports). This parameter is used principally by the bounced.pl daemon.

5.3.6 bounced_pidfile

(Default value: `bounced.pid`)
The file containing the PID of `bounced.pl`.

5.3.7 cookie_expire

(Default value: `0`) Lifetime (in minutes) of HTTP cookies. This is the default value when not set explicitly by users.

5.3.8 cookie_domain

(Default value: `localhost`)
Domain for the HTTP cookies. If beginning with a dot ('.'), the cookie is available within the specified internet domain. Otherwise, for the specified host. Example :

```
cookie_domain cru.fr
cookie is available for host 'cru.fr'

cookie_domain .cru.fr
cookie is available for any host within 'cru.fr' domain
```

The only reason for replacing the default value would be where *WWSympa*'s authentication process is shared with an application running on another host.

5.3.9 default_home

(Default value: `home`)
Organization of the WWSympa home page. If you have only a few lists, the default value 'home' (presenting a list of lists organized by topic) should be replaced by 'lists' (a simple alphabetical list of lists).

5.3.10 icons_url

(Default value: `/icons`)
URL of WWSympa's icons directory.

5.3.11 log_facility

WWSympa will log using this facility. Defaults to *Sympa*'s syslog facility. Configure your syslog according to this parameter.

5.3.12 mhonarc

(Default value: `/usr/bin/mhonarc`)
Path to the (superb) MhOnArc program. Required for html archives http ://www.oac.uci.edu/indiv/ehood/mhonarc.html

5.3.13 password_case sensitive — insensitive

(Default value: `insensitive`)
If set to **insensitive**, WWSympa's password check will be insensitive. This only concerns passwords stored in Sympa database, not the ones in LDAP.

Be carefull : in previous 3.xx versions of Sympa, passworsd were lowercased before database insertion. Therefore changing to case-sensitive password checking could bring you some password checking problems.

5.3.14 title

(Default value: `Mailing List Service`)
The name of your mailing list service. It will appear in the Title section of WWSympa.

5.3.15 use_fast_cgi 0 — 1

(Default value: `1`)
Choice of whether or not to use FastCGI. On listes.cru.fr, using FastCGI increases WWSympa performance by as much as a factor of 10. Refer to http ://www.fastcgi.com/ and the Apache config section of this document for details about FastCGI.

5.4 MhOnArc

MhOnArc is a neat little converter from mime messages to html. Refer to http ://www.oac.uci.edu/indiv/ehood/mhonarc.html.

The long mhonarc resource file is used by *WWSympa* in a particular way, as mhonarc is used to produce not a complete html document, but only part (to be included in a complete document starting with <HTML> and terminating with </HTML> ;-)) The best way to start is to use the MhOnArc resource file as provided in the *WWSympa* distribution.

The mhonarc resource file is named `mhonarc-ressources`. You may locate this file either in

1. `~sympa/expl/mylist/mhonarc-ressources` in order to create a specific archive look for a particular list
2. or `~sympa/bin/mhonarc-ressources`

5.5 Archiving daemon

`archived.pl` converts messages from *Sympa*'s spools and calls `mhonarc` to create html versions (whose location is defined by the "arc_path" WWSympa parameter). You should probably install these archives outside the *Sympa* home_dir (*Sympa*'s initial choice for storing mail archives : `~sympa/expl/mylist`). Note that the html archive contains a text version of each message and is totally separate from *Sympa*'s main archive.

1. create a directory according to the WWSympa "arc_path" parameter (must be owned by sympa, does not have to be in Apache space unless your server uses chroot)
2. for each list, if you need a web archive, create a new web archive paragraph in the list configuration. Example :

   ```
   web_archive
   access public|private|owner|listmaster|closed
   ```

   If web_archive is defined for a list, every message distributed by this list is copied to `~sympa/spool/outgoing/`. (No need to create nonexistent subscribers to receive copies of messages)
3. start `archived.pl`. *Sympa* and Apache
4. check *WWSympa* logs, or alternatively, start `archived.pl` in debug mode (-d).
5. If you change mhonarc resources and wish to rebuild the entire archive using the new look defined for mhonarc, simply create an empty file named ".rebuild.mylist@myhost" in `~sympa/spool/outgoing`, and make sure that the owner of this file is *Sympa*.

   ```
   example : su sympa -c "touch ~sympa/spool/outgoing/.rebuild.sympa-fr@cru.fr"
   ```

   You can also rebuild web archives from within the admin page of the list.

5.6 Database configuration

WWSympa needs an RDBMS (Relational Database Management System) in order to run. All database access is performed via the *Sympa* API. *Sympa* currently interfaces with MySQL, PostgreSQL, Oracle and Sybase.

A database is needed to store user passwords and preferences. The database structure is documented in the *Sympa* documentation ; scripts for creating it are also provided with the *Sympa* distribution (in `script`).

User information (password and preferences) are stored in the «User» table. User passwords stored in the database are encrypted using reversible RC4 encryption controlled with the `cookie` parameter, since *WWSympa* might need to remind users of their passwords. The security of *WWSympa* rests on the security of your database.

Chapitre 6

Sympa and its database

Most basic feature of *Sympa* will work without a RDBMS, but WWSympa and bounced require a relational database. Currently you can use one of the following RDBMS : MySQL, PostgreSQL, Oracle, Sybase. Interfacing with other RDBMS requires only a few changes in the code, since the API used, DBI[1] (DataBase Interface), has DBD (DataBase Drivers) for many RDBMS.

6.1 Prerequisites

You need to have a DataBase System installed (not necessarily on the same host as *Sympa*), and the client libraries for that Database installed on the *Sympa* host ; provided, of course, that a PERL DBD (DataBase Driver) is available for your chosen RDBMS ! Check the `DBI` Module Availability[2].

6.2 Installing PERL modules

Sympa will use `DBI` to communicate with the database system and therefore requires the DBD for your database system. DBI and DBD : :YourDB (`Msql-Mysql-modules` for MySQL) are distributed as CPAN modules. Refer to 3.2.3, page 23 for installation details of these modules.

[1] http ://www.symbolstone.org/technology/perl/DBI/
[2] http ://www.symbolstone.org/technology/perl/DBI/

6.3 Creating a sympa DataBase

6.3.1 Database structure

The sympa database structure is slightly different from the structure of a `subscribers` file. A `subscribers` file is a text file based on paragraphs (similar to the `config` file) ; each paragraph completely describes a subscriber. If somebody is subscribed to two lists, he/she will appear in both subscribers files.

The DataBase distinguishes information relative to a person (e-mail, real name, password) and his/her subscription options (list concerned, date of subscription, reception option, visibility option). This results in a separation of the data into two tables : the user_table and the subscriber_table, linked by a user/subscriber e-mail.

6.3.2 Database creation

The `create_db` script below will create the sympa database for you. You can find it in the `script/` directory of the distribution (currently scripts are available for MySQL, PostgreSQL, Oracle and Sybase).

– MySQL database creation script

```
## MySQL Database creation script

CREATE DATABASE sympa;

## Connect to DB
\r sympa

CREATE TABLE user_table (
   email_user           varchar (100) NOT NULL,
   gecos_user           varchar (150),
   password_user varchar (40),
cookie_delay_user int,
lang_user varchar (10),
PRIMARY KEY (email_user)
);

CREATE TABLE subscriber_table (
   list_subscriber        varchar (50) NOT NULL,
user_subscriber varchar (100) NOT NULL,
date_subscriber datetime NOT NULL,
update_subscriber datetime,
visibility_subscriber varchar (20),
reception_subscriber varchar (20),
```

```
bounce_subscriber varchar (35),
comment_subscriber varchar (150),
PRIMARY KEY (list_subscriber, user_subscriber),
INDEX (user_subscriber,list_subscriber)
);
```

– PostgreSQL database creation script

```
-- PostgreSQL Database creation script

CREATE DATABASE sympa;

-- Connect to DB
\connect sympa

DROP TABLE user_table;
CREATE TABLE user_table (
   email_user           varchar (100) NOT NULL,
   gecos_user           varchar (150),
cookie_delay_user       int4,
        password_user varchar (40),
        lang_user               varchar (10),
CONSTRAINT ind_user PRIMARY KEY (email_user)
);

DROP TABLE subscriber_table;
CREATE TABLE subscriber_table (
   list_subscriber        varchar (50) NOT NULL,
user_subscriber varchar (100) NOT NULL,
date_subscriber datetime NOT NULL,
update_subscriber  datetime,
visibility_subscriber varchar (20),
reception_subscriber varchar (20),
bounce_subscriber varchar (35),
comment_subscriber varchar (150),
CONSTRAINT ind_subscriber PRIMARY KEY (list_subscriber, user_subscriber)
);
CREATE INDEX subscriber_idx ON subscriber_table (user_subscriber,list_subscriber);
```

– Sybase database creation script

```
/* Sybase Database creation script 2.5.2 */
/* Thierry Charles <tcharles@electron-libre.com> */
/* 15/06/01 : extend password_user */

/* sympa database must have been created */
/* eg: create database sympa on your_device_data=10 log on your_device_log=4 */
use sympa
go
```

```
create table user_table
(
    email_user             varchar(100)           not null,
    gecos_user             varchar(150)           null    ,
    password_user          varchar(40)            null    ,
    cookie_delay_user      numeric                null    ,
    lang_user              varchar(10)            null    ,
    constraint ind_user primary key (email_user)
)
go

create index email_user_fk on user_table (email_user)
go


create table subscriber_table
(
    list_subscriber        varchar(50)            not null,
    user_subscriber        varchar(100)           not null,
    date_subscriber        datetime               not null,
    update_subscriber      datetime               null,
    visibility_subscriber  varchar(20)            null    ,
    reception_subscriber   varchar(20)            null    ,
    bounce_subscriber      varchar(35)            null    ,
    comment_subscriber      varchar(150)            null    ,
    constraint ind_subscriber primary key (list_subscriber, user_subscriber)
)
go

create index list_subscriber_fk on subscriber_table (list_subscriber)
go

create index user_subscriber_fk on subscriber_table (user_subscriber)
go
```

– Oracle database creation script

```
## Oracle Database creation script
## Fabien Marquois <fmarquoi@univ-lr.fr>

/Bases/oracle/product/7.3.4.1/bin/sqlplus loginsystem/passwdoracle <<-!
 create user SYMPA identified by SYMPA default tablespace TABLESP
temporary tablespace TEMP;
 grant create session to SYMPA;
 grant create table to SYMPA;
 grant create synonym to SYMPA;
 grant create view to SYMPA;
 grant execute any procedure to SYMPA;
```

```
 grant select any table to SYMPA;
 grant select any sequence to SYMPA;
 grant resource to SYMPA;
!

/Bases/oracle/product/7.3.4.1/bin/sqlplus SYMPA/SYMPA <<-!
CREATE TABLE user_table (
        email_user              varchar2(100) NOT NULL,
        gecos_user              varchar2(150),
        password_user           varchar2(40),
        cookie_delay_user       number,
        lang_user               varchar2(10),
        CONSTRAINT ind_user PRIMARY KEY (email_user)
);
CREATE TABLE subscriber_table (
        list_subscriber         varchar2(50) NOT NULL,
        user_subscriber         varchar2(100) NOT NULL,
        date_subscriber         date NOT NULL,
update_subscriber date,
        visibility_subscriber   varchar2(20),
        reception_subscriber    varchar2(20),
  bounce_subscriber       varchar2 (35),
  comment_subscriber       varchar2 (150),
        CONSTRAINT ind_subscriber PRIMARY KEY (list_subscriber,user_subscriber)
);
!
```

You can execute the script using a simple SQL shell such as mysql or psql.

Example :

```
# mysql  < create_db.mysql
```

6.4 Setting database privileges

We strongly recommend you restrict access to *sympa* database. You will then set `db_user` and `db_passwd` in `sympa.conf`.

With :

```
grant all on sympa.* to sympa@localhost identified by 'your_password';
flush privileges;
```

6.5 Importing subscribers data

6.5.1 Importing data from a text file

You can import subscribers data into the database from a text file having one entry per line : the first field is an e-mail address, the second (optional) field is the free form name. Fields are spaces-separated.

Example :

```
## Data to be imported
## email        gecos
john.steward@some.company.com           John - accountant
mary.blacksmith@another.company.com     Mary - secretary
```

To import data into the database :

```
cat /tmp/my_import_file | sympa.pl --import=my_list
```

(see 3.6, page 28).

6.5.2 Importing data from subscribers files

If a mailing list was previously setup to store subscribers into `subscribers` file (the default mode in versions older then 2.2b) you can load subscribers data into the sympa database. The simple way is to edit the list configuration using *WWSympa* (this requires listmaster privileges) and change the data source from **file** to **database** ; subscribers data will be loaded into the database at the same time.

If the subscribers file is too big, a timeout may occur with the FastCGI (You can set longer timeout with `-idle-timeout` `-idle-timeout` option of `FastCgiServer` Apache configuration directive). Then you should use `load_subscribers.pl` script.

6.6 Extending database table format

You can easily add other fields to **subscriber_table** and **user_table**, they will not disturb *Sympa* because it makes clear what field itexpects in SELECT queries.

Moreover you can access these database fields from within *Sympa* (in templates), as far as you list these additional fields in `sympa.conf` (See 4.10.9, page 45 and 4.10.10, page 45).

6.7 *Sympa* configuration

To store subscriber information in your newly created database, you first need to tell *Sympa* what kind of database to work with, then you must configure your list to access the database.

You define the database source in `sympa.conf` : `db_type`, `db_name`, `db_host`, `db_user`, `db_passwd`.

If you are interfacing *Sympa* with an Oracle database, `db_name` is the SID.

All your lists are now configured to use the database, unless you set list parameter `user_data_source` to **file** or **include**.

Sympa will now extract and store user information for this list using the database instead of the `subscribers` file. Note however that subscriber information is dumped to `subscribers.db.dump` at every shutdown, to allow a manual rescue restart (by renaming subscribers.db.dump to subscribers and changing the user_data_source parameter), if ever the database were to become inaccessible.

Chapitre 7

Using *Sympa* with LDAP

LDAP is a client-server protocol for accessing a directory service. Sympa provide various features based on access to one or more LDAP directories :

– authentication using LDAP directory insteed of sympa internal storage of password

– named filters used in scenario condition

– LDAP extraction of list subscribers (see 13.2.1)

7.1 Authentication via uid or alternate email

Sympa stores the data relative to the subscribers in a DataBase. Among these data : password, email exploited during the Web authentication . The module of LDAP authentication allows to use *Sympa* in intranet without duplicating the user's passwords.

Then, users can indiferently authenticate with their ldap_uid, their alternate_email or their canonic email stored in the LDAP directory.

Sympa gets the canonic email in the LDAP directory with the ldap_uid or the alternate_email. *Sympa* will first intend an anonymous bind to the directory to get the user's DN, and then *Sympa* will bind with the DN and the user's ldap_password in order to realise an efficient authentication. This last bind will work only if the good ldap_password is provided. Indeed the value returned by the bind(DN,ldap_password) is tested.

Example : a person is described by

```
Dn:cn=Fabrice Rafart,
```

```
                    ou=Siege ,
                    o=MaSociete ,
                    c=FR Objectclass:
                    person Cn: Fabrice Rafart
                    Title: Network Responsible
                    O: Siege
                    Or: Data processing
                    Telephonenumber: 01-00-00-00-00
                    Facsimiletelephonenumber:01-00-00-00-00
                    L:Paris
                    Country: France

uid: frafart
  mail: Fabrice.Rafart@MaSociete.fr
                    alternate_email: frafart@MaSociete.fr
                    alternate:rafart@MaSociete.fr
```

So Fabrice Rafart can be authenticated with : frafart, Fabrice.Rafart@MaSociete.fr, frafart@MaSociete.fr,Rafart@MaSociete.fr. After this operation, the address in the field FROM will be the Canonic email, in this case Fabrice.Rafart@MaSociete.fr. That means that *Sympa* will get this email and use it during all the session until you clearly ask to *Sympa* to change your email address via 2 pages : which and pref.

7.1.1 auth.conf

The `~sympa/etc/auth.conf` configuration file contains numerous parameters which are read on start-up of *Sympa*. If you change this file, do not forget that you will need to restart *Sympa* afterwards.

The `sympa/etc/auth.conf` is organised in paragraphs. Each paragraph coincides with the configuration of an ldap directory.

The `sympa/etc/auth.conf` file contains directives in the following format :

paragraphs
keyword value
paragraphs
keyword value

Comments start with the # character at the beginning of a line.

Empty lines are also considered as comments and are ignored at the beginning. After the first paragraph they are consideredas paragrahs separators.

There should only be one directive per line, but their order in the file is of no importance.

Thanks to this type of configuration *Sympa* is able to consult various directories. So, users who come from different directories will be authenticated thanks to their ldap_password. Indeed, *Sympa* will try to bind on the first directory with the given ldap_password, if it does not work, *Sympa* will try to bind on the second with the same ldap_password etc.. This mecanism is useful in the case of homonyms.

Example :

```
#Configuration file auth.conf for the LDAP authentification
#Description of parameters for each directory



ldap
host ldap.univ-rennes1.fr:389
timeout 30
suffix dc=univ-rennes1,dc=fr
get_dn_by_uid_filter (uid=[sender])
get_dn_by_email_filter (|(mail=[sender])(mailalternateaddress=[sender]))
email_attribute mail
alternative_email_attribute mailalternateaddress,ur1mail
scope sub

ldap
host ldap.univ-nancy2.fr:392,ldap1.univ-nancy2.fr:392,ldap2.univ-nancy2.fr:392
timeout 20
suffix dc=univ-nancy2,dc=fr
get_dn_by_uid_filter (uid=[sender])
get_dn_by_email_filter (|(mail=[sender])(n2atraliasmail=[sender]))
alternative_email_attribute n2atrmaildrop
email_attribute mail
scope sub
```

– host

This keyword is **mandatory**. It is the domain name used in order to bind to the directory and then to extract informations. You must mention the port number after the server name. The replication is also taken in charge, then the different servers are comma separated.
Example :

```
host ldap.univ-rennes1.fr:389
host ldap0.university.com:389,ldap1.university.com:389,ldap2.university.com:389
```

– timeout

It corresponds to the timelimit in the Search fonction. A timelimit that restricts the maximum time (in seconds) allowed for a search. A value of 0, and the default, means that no timelimit will be requested.

– suffix

The root of the DIT (Directory Information Tree).The DN that is the base object entry relative to which the search is to be performed.
Example: `dc=university,dc=fr`

– get_dn_by_uid_filter

You define the search filter corresponding to the ldap_uid. (RFC 2254 compliant). If you want to apply the filter on the user, mention him with the variable ' [sender] '. It would work with every type of authentication (uid, alternate_email..).
Example :

```
(Login = [sender])
(|(ID = [sender])(UID = [sender]))
```

– get_dn_by_email_filter

You define the search filter corresponding to the emails (canonic and alternative).(RFC 2254 compliant). If you want to apply the filter on the user, mention him with the variable ' [sender] '. It would work with every type of authentication (uid, alternate_email..).
Example : a person is described by

```
                    Dn:cn=Fabrice Rafart,
                    ou=Siege ,
                    o=MaSociete ,
                    c=FR Objectclass:
                    person Cn: Fabrice Rafart
                    Title: Network Responsible
                    O: Siege
                    Or: Data processing
                    Telephonenumber: 01-00-00-00-00
                    Facsimiletelephonenumber:01-00-00-00-00
                    L:Paris
                    Country: France

uid: frafart
  mail: Fabrice.Rafart@MaSociete.fr
                    alternate_email: frafart@MaSociete.fr
                    alternate:rafart@MaSociete.fr
```

The filters can be :

```
(mail = [sender])
(| (mail = [sender])(alternate_email = [sender]) )
(| (mail = [sender])(alternate_email = [sender])(alternate  = [sender]) )
```

– email_attribute

  The name of the attribute for the canonic email in your directory : for instance mail, canonic_email, canonic_add res ... In the previous example the canonic email is 'mail'.
– alternate_email_attribute

  The name of the attribute for the alternate email in your directory : for instance alternate_email, mailalternatea ddress, ... You make a list of these attributes separated by commas.
  With this list *Sympa* creates a cookie which contains various informations : the user is authenticated via Ldap or not, his alternate email. To store the alternate email is interesting when you want to canonify your preferences an d subscriptions. That is to say you want to use a unique adress in User_table and Subscriber_table which is the canonic email.
– scope

  (Default value: `sub`) By default the search is performed on the whole tree below the specified base object. This may be changed by specifying a scope :
  – base
    Search only the base object.
  – one
    Search the entries immediately below the base object.
  – sub
    Search the whole tree below the base object. This is the default.

7.2 Named Filters

At the moment Named Filters are only used in scenarii. They enable to select a category of people who will be authorized or not to realise some actions.

As a consequence, you can grant privileges in a list to people belonging to an LDAP directory thanks to a scenario.

7.2.1 Definition

People are selected thanks to an LDAP filter defined in a configuration file. This file must have the extension '.ldap'.It is stored in `~sympa/etc/search_filters/`.

You must mention many informations in order to create a Named Filter :

- host
  A list of host :port LDAP directories (replicates) entries.
- suffix
  Defines the naming space covered by the search (optional, depending on the LDAP server).
- filter
  Defines the LDAP search filter (RFC 2254 compliant). But you must absolutely take into account the first part of the filter which is : ('mail_attribute' = [sender]) as shown in the example. you will have to replce 'mail_attribute' by the name of the attribute for the email. So *Sympa* verifies if the user belongs to the category of people defined in the filter.
- scope
  By default the search is performed on the whole tree below the specified base object. This may be chaned by specify ing a scope :
  - base : Search only the base object.
  - one
    Search the entries immediately below the base object.
  - sub
    Search the whole tree below the base object. This is the default.

example.ldap : we want to select the professors of mathematics in the university of Rennes1 in France

```
host ldap.univ-rennes1.fr:389,ldap2.univ-rennes1.fr:390
suffix dc=univ-rennes1.fr,dc=fr
filter (&(canonic_mail = [sender])(EmployeeType = prof)(subject = math))
scope sub
```

7.2.2 Search Condition

The search condition is used in scenarii which are defined and decribed in (see 10.8)

The syntax of this rule is :

```
search(example.ldap,[sender])      smtp,smime,md5    -> do_it
```

The variables used by 'search' are :
- the name of the LDAP Configuration file

– the [sender]
  That is to say the sender email.

The method of authentication does not change.

Chapitre 8

Sympa with S/MIME and HTTPS

S/MIME is a cryptographic method for Mime messages based on X509 certificates. Before installing *Sympa* S/Mime features (which we call S/Sympa), you should be under no illusion about what the S stands for : “S/MIME” means “Secure MIME”. That S certainly does not stand for “Simple”.

The aim of this chapter is simply to describe what security level is provided by *Sympa* while using S/MIME messages, and how to configure *Sympa* for it. It is not intended to teach anyone what S/Mime is and why it is so complex ! RFCs numbers 2311, 2312, 2632, 2633 and 2634, along with a lot of literature about S/MIME, PKCS#7 and PKI is available on the Internet. *Sympa* 2.7 is the first version of *Sympa* to include S/MIME features as beta-testing features.

8.1 Signed message distribution

No action required. You probably imagine that any mailing list manager (or any mail forwarder) is compatible with S/MIME signatures, as long as it respects the MIME structure of incoming messages. You are right. Even Majordomo can distribute a signed message ! As *Sympa* provides MIME compatibility, you don’t need to do anything in order to allow subscribers to verify signed messages distributed through a list. This is not an issue at all, since any processes that distribute messages are compatible with end user signing processes. Sympa simply skips the message footer attachment (ref 11.11, page 104) to prevent any body corruption which would break the signature.

8.2 Use of S/MIME signature by Sympa itself

Sympa is able to verify S/MIME signatures in order to apply S/MIME authentication methods for message handling. Currently, this feature is limited to the distribution process, and to any commands *Sympa* might find in the message body. The reasons for this restriction are related to current S/MIME usage. S/MIME signature structure is based on the encryption of a digest of the message. Most S/MIME agents do not include any part of the message headers in the message digest, so anyone can modify the message header without signature corruption ! This is easy to do : for example, anyone can edit a signed message with their preferred message agent, modify whatever header they want (for example `Subject :` , `Date :` and `To :`, and redistribute the message to a list or to the robot without breaking the signature.

So Sympa cannot apply the S/MIME authentication method to a command parsed in the `Subject :` field of a message or via the `-subscribe` or `-unsubscribe` e-mail address.

8.3 Use of S/MIME encryption

S/Sympa is not an implementation of the “S/MIME Symmetric Key Distribution” internet draft. This sophisticated scheme is required for large lists with encryption. So, there is still some scope for future developments :)

We assume that S/Sympa distributes message as received, i.e. unencrypted when the list receives an unencrypted message, but otherwise encrypted.

In order to be able to send encrypted messages to a list, the sender needs to use the X509 certificate of the list. Sympa will send an encrypted message to each subscriber using the subscriber’s certificate. To provide this feature, *Sympa* needs to manage one certificate for each list and one for each subscriber. This is available in Sympa version 2.8 and above.

8.4 S/Sympa configuration

8.4.1 Installation

The only requirement is OpenSSL (http ://www.openssl.org) version 0.9.5a and above. OpenSSL is used by *Sympa* as an external plugin (like sendmail or postfix), so it must be installed with the appropriate access (x for sympa.sympa).

8.4.2 configuration in sympa.conf

S/Sympa configuration is very simple. If you are used to Apache SSL, you should not feel lost. If you are an OpenSSL guru, you will feel at home, and there may even be changes you will wish to suggest to us.

The basic requirement is to let *Sympa* know where to find the binary file for the OpenSSL program and the certificates of the trusted certificate authority. This is done using the optional parameters `openSSL` `openSSL` and `trusted_ca_options`.

- `openSSL` : the path for the OpenSSL binary file, usually `/usr/local/ssl/bin/openSSL`
- `trusted_ca_options` : the option used by OpenSSL for trusted CA certificates. The file `~sympa/bin/etc/cabundle.crt` is distributed with Sympa and describes a well known set of CA's, such as the default Netscape navigator configuration. You can declare this set of certificates as trusted by setting `trusted_ca_options -CAfile` *`/home/sympa/bin/etc/ca-bundle.crt`* `-CAfile` *`/home/sympa/bin/etc/ca-bundle.crt`*. You can also use the `-CApath -CApath openssl openssl` option, or both `-CApath` and `-CAfile`. Example : `trusted_ca_options -CApath ~sympa/etc/ssl.crt -CAfile ~sympa/bin/etc/ca-bundle.crt`.
  Both the `-CAfile -CAfile` file and the `-CApath -CApath` directory should be shared with your Apache+mod_ssl configuration. This is useful for the S/Sympa web interface. Please refer to the OpenSSL documentation for details.
- `key_password` : the password used to protect all list private keys. xxxxxxx

8.4.3 configuration to recognize S/MIME signatures

Once `OpenSSL` has been installed, and `sympa.conf` configured, your S/Sympa is ready to use S/Mime signatures for any authentication operation. You simply need to use the appropriate scenario for the operation you want to secure. (see 10.8, page 88).

When receiving a message, *Sympa* applies the scenario with the appropriate authentication method parameter. In most cases the authentication method is "`smtp`", but in cases where the message is signed and the signature has been checked and matches the sender e-mail, *Sympa* applies the "`smime`" authentication method.

It is vital to ensure that if the scenario does not recognize this authentication method, the operation requested will be rejected. Consequently, scenarii distributed prior to version 2.7 are not compatible with the OpenSSL configuration of Sympa. All standard scenarii (those distributed with sympa) now include the `smime` method. The following example is named `send.private_smime`, and restricts sends to subscribers using an S/mime signature :

```
title.us restricted to subscribers check smime signature
title.fr limité aux abonnés, vérif de la signature smime
```

```
is_subscriber([listname],[sender])              smime  -> do_it
is_editor([listname],[sender])                  smime  -> do_it
is_owner([listname],[sender])                   smime  -> do_it
```

It as also possible to mix various authentication methods in a single scenario. The following example, `send.private_key`, requires either an md5 return key or an S/Mime signature :

```
title.us restricted to subscribers with previous md5 authentication
title.fr réservé aux abonnés avec authentification MD5 préalable

is_subscriber([listname],[sender]) smtp          -> request_auth
true()                             md5,smime     -> do_it
```

8.4.4 distributing encrypted messages

In this section we describe S/Sympa encryption features. The goal is to use S/MIME encryption for distribution of a message to subscribers whenever the message has been received encrypted from the sender.

Why is S/Sympa concerned by the S/MIME encryption distribution process ? It is because encryption is performed using the **recipient** X509 certificate, whereas the signature requires the sender's private key. Thus, an encrypted message can be read by the recipient only if he or she is the owner of the private key associated with the certificate. Consequently, the only way to encrypt a message for a list of recipients is to encrypt and send the message for each recipient. This is what S/Sympa does when distributing a encrypted message.

The S/Sympa encryption feature in the distribution process supposes that sympa has received an encrypted message for some list. To be able to encrypt a message for a list, the sender must have some access to an X509 certificate for the list. So the first requirement is to install a certificate and a private key for the list. The mechanism whereby certificates are obtained and managed is complex. Current versions of S/Sympa assume that list certificates and private keys are installed by the listmaster. It is a good idea to have a look at the OpenCA (http ://www.openca.org) documentation and/or PKI providers' web documentation. You can use commercial certificates or home-made ones. Of course, the certificate must be approved for e-mail applications, and issued by one of the trusted CA's described in the `-CAfile -CAfile` file or the `-CApath -CApath` OpenSSL option.

The list private key must be installed in a file named `~sympa/expl/mylist/private_key`. All the list private keys must be encrypted using a single password defined by the `password` parameter in `sympa.conf`.

Use of Netscape navigator to obtain X509 list certificates

In many cases e-mail X509 certificates are distributed via a web server and loaded into the browser using your mouse :) Netscape allows certificates to be exported to a file. So one way to get a list certificate is to obtain an e-mail certificate for the canonical list address in your browser, and then to export and install it for Sympa :

1. browse the net and load a certificate for the list address on some PKI provider (your own OpenCa pki server , thawte, verisign, ...). Be careful : the e-mail certificate must be correspond exactly to the canonical address of your list, otherwise, the signature will be incorrect (sender e-mail will not match signer e-mail).
2. in the security menu, select the intended certificate and export it. Netscape will prompt you for a password and a filename to encrypt the output file. The format used by Netscape is "pkcs#12". Copy this file to the list home directory.
3. convert the pkcs#12 file into a pair of pem files : `cert.pem` and `private_key` using the `sympa/bin/p12topem.pl sympa/bin/p12topem.pl` script. Use `p12topem.pl -help` `p12topem.pl -help` for details.
4. be sure that `cert.pem` and `private_key` are owned by sympa with "r" access.
5. As soon as a certificate is installed for a list, the list home page includes a new link to load the certificate to the user's browser, and the welcome message is signed by the list.

8.5 Managing certificates with tasks

You may automate the management of certificates with two global task models provided with *Sympa*. See 10.10, page 92 to know more about tasks. Report to 4.12.3, page 47 to configure your *Sympa* to use these facilities.

8.5.1 chk_cert_expiration.daily.task model

A task created with the model `chk_cert_expiration.daily.task` checks every day the expiration date of certificates stored in the `~sympa/expl/X509-user-certs/` directory. The user is warned with the `daily_cert_expiration` template when his certificate has expired or is going to expire within three days.

8.5.2 crl_update.daily.task model

You may use the model `crl_update.daily.task` to create a task which daily updates the certificate revocation lists when needed.

Chapitre 9

Virtual robot

Sympa is designed to manage multiple distinct mailing list servers on a single host with a single Sympa installation. Sympa virtual robots are like Apache virtual hosting. Sympa virtual robot definition includes a specific email address for the robot itself and its lists and also a virtual http server. Each robot provides access to a set of lists, each list is related to only one robot.

Most configuration parameters can be define for each robot except general Sympa installation parameters (binary and spool location, smtp engine, antivirus plugging,...).

The Virtual robot name as defined in *Sympa* documentation and configuration file refers to the Internet domaine of the Virtual robot.

9.1 How to create a virtual robot

You don't need to install several Sympa server. A single `sympa.pl` deamon and one or more fastcgi servers can serve all virtual robot. Just configure the server environment in order to accept the new domain definition.

– **The DNS** must be configured to define a new mail exchanger record (MX) to route message to your server. A new host (A record) or alias (CNAME) are mandatory to define the new web server.

– Configure your **MTA (sendmail, postfix, exim, ...)** to accept incoming messages for the new robot domain. Add mail aliases for the robot :
Examples (with sendmail) :

```
sympa@your.virtual.domain:      "| /home/sympa/bin/queue sympa@your.virtual.dom
listmaster@your.virtual.domain: "| /home/sympa/bin/queue listmaster@your.virtua
```

– Define a **virtual host in your HTTPD server**. The fastcgi servers defined in the common section of you httpd server can be used by each virtual server. You don't

need to run dedicated fascgi server for each virtual robot.
Examples :

```
FastCgiServer /home/sympa/bin/wwsympa.fcgi -processes 3 -idle-timeout 120
.....
<VirtualHost 195.215.92.16>
  ServerAdmin webmaster@your.virtual.domain
  DocumentRoot /var/www/your.virtual.domain
  ServerName your.virtual.domain

  <Location /wws>
     SetHandler fastcgi-script
  </Location>

  ScriptAlias /wws /home/sympa/bin/wwsympa.fcgi

</VirtualHost>
```

– Create a `robot.conf` for the virtual robot (current web interface does not provide Virtual robot creation yet).

9.2 robot.conf

A robot is named by its domain, let's say my.domain.organd defined by a directory `~sympa/etc/my.domain.org`. This directory must contain at least a `robot.conf` file. This files has the same format as `/etc/sympa.conf` (have a look at robot.conf in the sample dir). Only the following parameters can be redefined for a particular robot :

– `http_host`
  This hostname will be compared with 'SERVER_NAME' ENV var in wwsympa.fcgi to deduce the current Virtual Robot.
– `wwsympa_url`
  The base URL of WWSympa
– `cookie_domain`
– `email`
– `title`
– `default_home`
– `create_list`
– `lang`
– `log_smtp`
– `listmaster`
– `max_size`
– `dark_color`, `light_color`, `text_color`, `bg_color`, `error_color`, `selected_color`, `shaded_color`

These settings overwrite the equivalent global parameter as defined in `/etc/sympa.conf` for my.domain.orgrobot; the main `listmaster` still has privileges on Virtual Robots though. The http_host parameter is compared by wwsympa

with the SERVER_NAME environment variable to recognize which robot is in used.

9.2.1 Robot customization

If needed, you can customize each virtual robot using its set of templates and scenario.

`~sympa/etc/my.domain.org/wws_templates/`, `~sympa/etc/my.domain.org/templates/`, `~sympa/etc/my.domain.org/scenari/` directories are searched when loading templates or scenari before searching into `~sympa/etc` and `~sympa/bin/etc`. This allows to define different privileges and a different GUI for a Virtual Robot.

Chapitre 10

Customizing *Sympa*/*WWSympa*

10.1 Template file format

Template files within *Sympa* and *WWSympa* are text files containing programming elements (variables, conditions, loops, file inclusions) that will be parsed in order to adapt to the runtime context. These templates are an extension of programs and therefore give access to a limited list of variables (those defined in the '*hash*' parameter given to the parser).

Review the Site template files (10.2, page 84) and List template files (11.8, page 101).

The following describes the syntactical elements of templates.

10.1.1 Variables

Variables are enclosed between brackets '*[]*'. The variable name is composed of alphanumerics (0-1a-zA-Z) or underscores (_). The syntax for accessing an element in a '*hash*' is [hash->elt].

Examples :

```
[url]
[is_owner]
[list->name]
[user->lang]
```

For each template you wish to customize, check the available variables in the documentation.

10.1.2 Conditions

Conditions include variable comparisons (= and <>), or existence. Syntactical elements for conditions are [IF xxx], [ELSE], [ELSIF xxx] and [ENDIF].

Examples :

```
[IF  user->lang=fr]
Bienvenue dans la liste [list->name]
[ELSIF user->lang=es]
Bienvenida en la lista [list->name]
[ELSE]
Welcome in list [list->name]
[ENDIF]

[IF is_owner]
The following commands are available only
for lists owners or moderators:
....
[ENDIF]
```

10.1.3 Loops

Loops make it possible to traverse a list of elements (internally represented by a '*hash*' or an '*array*').

Example :

```
A review of public lists

[FOREACH l IN lists]
   [l->NAME]
   [l->subject]
[END]
```

`[elt->NAME]` is a special element of the current entry providing the key in the '*hash*' (in this example the name of the list). When traversing an '*array*', `[elt->INDEX]` is the index of the current entry.

10.1.4 File inclusions

You can include another file within a template . The specified file can be included as is, or itself parsed (there is no loop detection). The file path is either specified in the directive or accessed in a variable.

Inclusion of a text file :

```
[INCLUDE 'archives/last_message']
[INCLUDE file_path]
```

The first example includes a file whose relative path is `archives/last_message`. The second example includes a file whose path is in file_path variable.

Inclusion and parsing of a template file :

```
[PARSE 'welcome.tpl']
[PARSE file_path]
```

The first example includes the template file `welcome.tpl`. The second example includes a template file whose path is in file_path variable.

10.1.5 Stop parsing

You may need to exclude certain lines in a template from the parsing process. You can perform this by stopping and restarting the parsing.

Escaping sensitive JavaScript functions :

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- for other browsers
  function toggle_selection(myfield) {
    for (i = 0; i < myfield.length; i++) {
    [STOPPARSE]
       if (myfield[i].checked) {
            myfield[i].checked = false;
       }else {
    myfield[i].checked = true;
       }
    [escaped_start]
    }
  }
// end browsers -->
</SCRIPT>
</HEAD>
```

10.2 Site template files

These files are used by Sympa as service messages for the HELP, LISTS and REMIND * commands. These files are interpreted (parsed) by *Sympa* and respect the template format ; every file has a .tpl extension. See 10.1, page 81.

Sympa looks for these files in the following order (where <list> is the listname if defined, <action> is the name of the command, and <lang> is the preferred language of the user) :

1. ~sympa/expl/<list>/<action>.<lang>.tpl.
2. ~sympa/expl/<list>/<action>.tpl.
3. ~sympa/etc/templates/<action>.<lang>.tpl.
4. ~sympa/etc/templates/<action>.tpl.
5. ~sympa/bin/etc/templates/<action>.<lang>.tpl.
6. ~sympa/bin/etc/templates/<action>.tpl.

If the file starts with a From : line, it is considered as a full message and will be sent (after parsing) without adding SMTP headers. Otherwise the file is treated as a text/plain message body.

The following variables may be used in these template files :

- [conf->email] : sympa e-mail address local part
- [conf->domain] : sympa robot domain name
- [conf->sympa] : sympa's complete e-mail address
- [conf->wwsympa_url] : *WWSympa* root URL
- [conf->listmaster] : listmaster e-mail addresses
- [user->email] : user e-mail address
- [user->gecos] : user gecos field (usually his/her name)
- [user->password] : user password
- [user->lang] : user language

10.2.1 helpfile.tpl

This file is sent in response to a HELP command. You may use additional variables
- [is_owner] : TRUE if the user is list owner
- [is_editor] : TRUE if the user is list editor

10.2.2 lists.tpl

File returned by LISTS command. An additional variable is available :
- [lists] : this is a hash table indexed by list names and containing lists' subjects. Only lists visible to this user (according to the visibility list parameter) are listed.

Example :

```
These are the public lists for [conf->email]@[conf->domain]

[FOREACH l IN lists]

 [l->NAME]: [l->subject]

[END]
```

10.2.3 global_remind.tpl

This file is sent in response to a `REMIND *` command. (see 17.2, page 150) You may use additional variables
- [lists] : this is an array containing the list names the user is subscribed to.
Example :

```
This is a subscription reminder.

You are subscribed to the following lists :
[FOREACH l IN lists

 [l] : [conf->wwsympa\_url]/info/[l]

[END]

Your subscriber e-mail : [user->email]
Your password : [user->password]
```

10.2.4 your_infected_msg.tpl

This message is sent to warn the sender of a virus infected mail, indicating the name of the virus found (see **??**, page **??**).

10.3 Web template files

You may define your own web template files, different from the standard ones. *WWSympa* first looks for list specific web templates, then for site web templates, before

falling back on its defaults.

Your list web template files should be placed in the `~sympa/expl/mylist/wws_templates` directory ; your site web templates in `~sympa/expl/wws_templates` directory.

Note that web colors are defined in *Sympa*'s main Makefile (see 3.3, page 25).

10.4 Sharing data with other applications

You may extract subscribers for a list from any of :
– a text file
– a Relational database
– a LDAP directory
See lparam user_data_source liste parameter 13.2.1, page 115.

The **subscriber_table** and **user_table** can have more fields than the one used by *Sympa*. by defining these additional fields, they will be available from within *Sympa*'s scenario and templates (see 4.10.9, page 45 and 4.10.10, page 45).

10.5 Sharing *WWSympa* authentication with other applications

You might want to make other web applications collaborate with *Sympa*, and share the same authentication system. *Sympa* uses HTTP cookies to carry users' auth information from page to page. This cookie carries no information concerning privileges. To make your application work with *Sympa*, you have two possibilities :

– Delegating authentication operations to *WWSympa*
  If you want to avoid spending a lot of time programming a CGI to do Login, Logout and Remindpassword, you can copy *WWSympa*'s login page to your application, and then make use of the cookie information within your application. The cookie format is :
  `user=<user_email>:<md5>`
  where <user_email> is the user's complete e-mail address, and <md5> is a MD5 checksum of the <user_email>+*Sympa* `cookie` configuration parameter. Your application needs to know what the `cookie` parameter is, so it can check the HTTP cookie validity ; this is a secret shared between *WWSympa* and your application. *WWSympa*'s *loginrequest* page can be called to return to the referrer URL when an action is performed. Here is a sample HTML anchor :
  `<A HREF="/wws/loginrequest/referrer">Login page</A>`
– Using *WWSympa*'s HTTP cookie format within your auth module
  To cooperate with *WWSympa*, you simply need to adopt its HTTP cookie format

and share the secret it uses to generate MD5 checksums, i.e. the `cookie` configuration parameter. In this way, *WWSympa* will accept users authenticated through your application without further authentication.

10.6 Internationalization

Sympa was originally designed as a multilingual Mailing List Manager. Even in its earliest versions, *Sympa* separated messages from the code itself, messages being stored in NLS catalogues (according to the XPG4 standard). Later a `lang` list parameter was introduced. Nowadays *Sympa* is able to keep track of individual users' language preferences.

10.6.1 *Sympa* internationalization

Every message sent by *Sympa* to users, owners and editors is outside the code, in a message catalog. These catalogs are located in the `~sympa/nls/` directory. Messages have currently been translated into 10 different languages :

- cn-big5 : BIG5 Chinese (Hong Kong, Taiwan)
- cn-gb : GB Chinese (Mainland China)
- cz : Czech
- de : German
- es : Spanish
- fi : Finnish
- fr : French
- hu : Hungarian
- it : Italian
- pl : Polish
- us : US English

To tell *Sympa* to use a particular message catalog, you can either set the `lang` parameter in `sympa.conf`, or set the `sympa.pl -l` option on the command line.

10.6.2 List internationalization

The `lang` list parameter defines the language for a list. It is currently used by *WWSympa* and to initialize users' language preferences at subscription time.

In future versions, all messages returned by *Sympa* concerning a list should be in the list's language.

10.6.3 User internationalization

The user language preference is currently used by *WWSympa* only. There is no e-mail-based command for a user to set his/her language. The language preference is initialized when the user subscribes to his/her first list. *WWSympa* allows the user to change it.

10.7 Topics

WWSympa's homepage shows a list of topics for classifying mailing lists. This is dynamically generated using the different lists' `topics` configuration parameters. A list may appear in multiple categories.

The list of topics is defined in the `topics.conf` configuration file, located in the `~sympa/etc` directory. The format of this file is as follows :

```
<topic1_name>
title <topic1 title>
visibility <topic1 visibility>
....
<topicn_name/subtopic_name>
title <topicn title>
```

You will notice that subtopics can be used, the separator being /. The topic name is composed of alphanumerics (0-1a-zA-Z) or underscores (_). The order in which the topics are listed is respected in *WWSympa*'s homepage. The **visibility** line defines who can view the topic (now available for subtopics). It refers to the associated topics_visibility scenario. You will find a sample `topics.conf` in the `sample` directory ; NONE is installed as the default.

A default topic is hard-coded in *Sympa* : *default*. This default topic contains all lists for which a topic has not been specified.

10.8 Scenarii

List parameters controlling the behavior of commands are linked to different scenarii. For example : the `send private` parameter is related to the send.private scenario. There are three possible locations for a scenario. When *Sympa* seeks to apply a scenario, it looks first in the related list directory `~sympa/expl/<list>/scenari`. If it does not find the file there, it scans `~sympa/etc/scenari`, and finally `~sympa/bin/etc/scenari`, which is the directory installed by the Makefile.

A scenario is a small configuration language to describe who can perform an operation and which authentication method is requested for it. A scenario is an ordered set of

rules. The goal is to provide a simple and flexible way to configure authorization and authentication for each operation.

Each scenario rule contains :
– a condition : the condition is evaluated by *Sympa*. It can use variables such as [sender] for the sender e-mail, [list] for the listname etc.
– an authentication method. The authentication method can be `smtp`, `md5` or `smime`. The rule is applied by *Sympa* if both condition and authentication method match the runtime context. `smtp` is used if *Sympa* use the SMTP `from :` header , `md5` is used if a unique md5 key as been returned by the requestor to validate her message, `smime` is used for signed messages (see 8.4.3, page 75).
– a returned atomic action that will be executed by *Sympa* if the rule matches

Example

del.auth

```
title.us deletion performed only by list owners, need authentication
title.fr suppression réservée au propriétaire avec authentification
title.es eliminación reservada sólo para el propietario, necesita autentificación
```

```
is_owner([listname],[sender])  smtp       -> request_auth
is_listmaster([sender])        smtp       -> request_auth
true()                         md5,smime  -> do_it
```

10.8.1 rules specifications

A scenario consists of rules, evaluated in order beginning with the first. Rules are defined as follows :

```
<rule> ::= <condition> <auth_list> -> <action>

<condition> ::= [!] <condition
                | true ()
                | all ()
                | equal (<var>, <var>)
                | match (<var>, /perl_regexp/)
                | is_subscriber (<listname>, <var>)
                | is_owner (<listname>, <var>)
                | is_editor (<listname>, <var>)
                | is_listmaster (<var>)
                | older (<date>, <date>)    # true if first date is anterior to the second
                | newer (<date>, <date>)    # true if first date is posterior to the secon
<var> ::= [email] | [sender] | [user-><user_key_word>]
    | [subscriber-><subscriber_key_word>] | [list-><list_key_word>]
  | [conf-><conf_key_word>] | [msg_header-><smtp_key_word>] | [msg_body]
    | [msg_part->type] | [msg_part->body] | [is_bcc] | <string>
```

```
[is_bcc] ::= set to 1 if the list is neither in To: nor Cc:

<date> ::= <epoch date> | <date format>

<listname> ::= [listname] | <listname_string>

<auth_list> ::= <auth>,<auth_list> | <auth>

<auth> ::= smtp|md5|smime

<action> ::=   do_it [,notify]
             | do_it [,quiet]
             | reject(<tpl_name>)
             | request_auth
             | owner
     | editor
     | editorkey

<tpl_name> ::= corresponding template (<tpl_name>.tpl) is send to the sender

<user_key_word> ::= email | gecos | lang | password | cookie_delay_user
            | <additional_user_fields>

<subscriber_key_word> ::= email | gecos | bounce | reception
                  | visibility | date | update_date
  | <additional_subscriber_fields>

<list_key_word> ::= name | host | lang | max_size | priority | reply_to |
    status | subject | account |

<conf_key_word> ::= domain | email | listmaster | default_list_priority |
      sympa_priority | request_priority | lang | max_size
```

(Refer to 10.10, page 92 for date format definition)

perl_regexp can contain the string [host] (interpreted at run time as the list or robot domain). The variable notation [msg_header-><smtp_key_word>] is interpreted as the SMTP header value only when performing the sending message scenario. It can be used, for example, to require editor validation for multipart messages. [msg_part->type] and [msg_part->body] are the MIME parts content-types and bodies ; the body is available for MIME parts in text/xxx format only.

A bunch of scenarii is provided with the *Sympa* distribution ; they provide a large set of configuration that allow to create lists for most usage. But you will probably create scenarii for your own need. In this case, don't forget to restart *Sympa* and wwsympa because scenarii are not reloaded dynamicaly.

These standard scenarii are located in the `~sympa/bin/scenari/` directory. Default scenarii are named <command>.default.

You may also define and name your own scenarii. Store them in the ~sympa/etc/scenari directory. They will not be overwritten by Sympa release. Scenarii can also be defined for a particular virtual robot (using directory ~sympa/etc/<robot>/scenari) or for a list (~sympa/expl/<robot>/<list>/scenari).

Example :

Copy the previous scenario to scenari/subscribe.rennes1 :

```
equal([sender], 'userxxx@univ-rennes1.fr') smtp,smime -> reject
match([sender], /univ-rennes1\.fr\$/) smtp,smime -> do_it
true()                                smtp,smime -> owner
```

You may now refer to this scenario in any list configuration file, for example :

```
subscribe rennes1
```

10.8.2 scenario inclusion

Scenarii can also contain includes :

```
    subscribe
        include commonreject
        match([sender], /cru\.fr$/)          smtp,smime -> do_it
true()                                smtp,smime -> owner
```

In this case sympa applies recursively the scenario named include.commonreject before introducing the other rules. This possibility was introduced in order to facilitate the administration of common rules.

You can define a set of common scenario rules, used by all lists. include.<action>.header is automatically added to evaluated scenarios.

10.9 Loop detection

Sympa uses multiple tools to avoid loops in Mailing lists

First, it rejects messages coming from a robot (as indicated by the From : and other header fields), and messages containing commands.

Secondly, every message sent by *Sympa* includes an X-Loop header field set to the listname. If the message comes back, *Sympa* will detect that it has already been sent (unless X-Loop header fields have been erased).

Thirdly, *Sympa* keeps track of Message IDs and will refuse to send multiple messages with the same message ID to the same mailing list.

Finally, *Sympa* detect loops arising from command reports (i.e. sympa-generated replies to commands). This sort of loop might occur as follows :

```
1 - X sends a command to Sympa
2 - Sympa sends a command report to X
3 - X has installed a home-made vacation program replying to programs
4 - Sympa processes the reply and sends a report
5 - Looping to step 3
```

Sympa keeps track (via an internal counter) of reports sent to any particular address. The loop detection algorithm is :

- Increment the counter
- If we are within the sampling period (as defined by the `loop_command_sampling_delay` parameter)
  - If the counter exceeds the `loop_command_max` parameter, then do not send the report, and notify the listmaster
  - Else, start a new sampling period and reinitialize the counter, i.e. multiply it by the `loop_command_decrease_factor` parameter

10.10 Tasks

A task is a sequence of simple actions which realize a complex routine. It is executed in background by the task manager daemon and allow the list master to automate the processing of recurrent tasks. For example a task sends every year the subscribers of a list a message to remind their subscription.

A task is created with a task model. It is a text file which describes a sequence of simple actions. It may have different versions (for instance reminding subscribers every year or semester). A task model file name has the following format : `<model name>.<model version>.task`. For instance `remind.annual.task` or `remind.semestrial.task`.

Sympa provides several task models stored in `~sympa/bin/etc/global_task_models` and `~sympa/bin/etc/list_task_models` directories. Others can be designed by the list master.

A task is global or related to a list.

10.10.1 List task creation

You define in the list config file which model with which version you want to use (see 13.3.5, page 122). Then the task manager daemon will automatically create the task by looking for the appropriate model file in differents directories in the following order :

1. `~sympa/expl/<list name>/`
2. `~sympa/etc/list_task_models/`
3. `~sympa/bin/etc/list_task_models/`

See also 11.10, page 104, to know more about standard list models provided with *Sympa*.

10.10.2 Global task creation

The task manager daemon checks if a version of a global task model is specified in `sympa.conf` and then creates a task as soon as it finds the model file by looking in different directories in the following order :

1. `~sympa/etc/global_task_models/`
2. `~sympa/bin/etc/global_task_models/`

10.10.3 Model file format

Model files are composed of comments, labels, references, variables, date values and commands. All those syntactical elements are composed of alphanumerics (0-9a-zA-Z) and underscores (_).

– Comment lines begin by '#' and are not interpreted by the task manager.
– Label lines begin by '/' and are used by the next command (see below).
– References are enclosed between brackets '[]'. They refer to a value depending on the object of the task (for instance [list->name]). Those variables are instantiated when a task file is created from a model file. The list of available variables is the same as for templates (see 11.8, see page 101) plus [creation_date] (see below).
– Variables store results of some commands and are paramaters for others. Their name begins with '@'.
– A date value may be written in two ways :
  – absolute dates follow the format : xxxxYxxMxxDxxHxxMin. Y is the year, M the month (1-12), D the day (1-28|30|31, leap-years are not managed), H the hour (0-23), Min the minute (0-59). H and Min are optionnals. For instance, 2001y12m4d44min is the 4th of December 2001 at 00h44.

  - relative dates use the [creation_date] or [execution_date] references. [creation_date] is the date when the task file is created, [execution_date] when the command line is executed. A duration may follow with '+' or '-' operators. The duration is expressed like an absolute date whose all parameters are optionnals. Examples : [creation_date], [execution_date]+1y, [execution_date]-6m4d
- Command arguments are separated by commas and enclosed between parenthesis '()'.

Here is the list of current avalable commands :
- stop ()
  Stops the execution of the task and delete the task file
- next (<date value>, <label>)
  Stop the execution. The task will go on at the date value and begin at the label line.
- <@deleted_users> = delete_subs (<@user_selection>)
  Delete @user_selection email list and stores user emails successfully deleted in @deleted_users.
- send_msg (<@user_selection>, <template>)
  Send the template message to emails stored in @user_selection.
- @user_selection = select_subs (<condition>)
  Store emails which match the condition in @user_selection. See 8.6 Scenarii section to know how to write conditions. Only available for list models.
- create (global — list (<list name>), <model type>, <model>)
  Create a task for object with model file `~model type.model.task`.
- chk_cert_expiration (<template>, <date value>)
  Send the template message to emails whose certificate has expired or will expire before the date value.
- update_crl (<file name>, <date value>)
  Update certificate revocation lists (CRL) which are expired or will expire before the date value. The file stores the CRL's URLs.

Model files may have a scenario-like title line at the beginning.

When you change a configuration file by hand, and a task parameter is created or modified, it is up to you to remove existing task files in the `task/` spool if needed. Task file names have the following format :

`<date>.<label>.<model name>.<list name | global>` where :

- date is when the task is executed, it is an epoch date
- label states where in the task file the execution begins. If empty, starts at the beginning

10.10.4 Model file examples

- remind.annual.task

- expire.annual.task

– crl_update.daily.task

```
title.fr mise a jour quotidienne des listes de révocation
title.us daily certificate revocation list update
title.et sertifikaatide kehtivuse nimekirja uuendatakse iga päev
title.hu napi frissítése a hitelesítési bizonylatok visszavonásának

/ACTION
update_crl (CA_list, [execution_date]+1d)
next ([execution_date] + 1d, ACTION)
```

Chapitre 11

Mailing list definition

The mailing list creation tool is Sympa's web interface. However, this web feature has only been available from version 2.7 onwards. Users of previous versions will need to create new lists using their favorite text file editor.

This chapter describe how to create a mailing list without using web tools. See 12, page 107 for instructions on the use of WWSympa, which is no doubt the easier method.

The only part of list creation requiring system privileges is the declaration of new system-wide mail aliases. All the other steps should be performed by the `sympa` user, which will ensure that the files created have the correct access permissions.

- add aliases in the alias file
- create the list directory `~sympa/expl/mylist`
- create the configuration file in the `~sympa/expl/mylist` directory
- create customized message files (welcome, bye, removed remind, message.header, message.footer) if needed ; in most cases you will probably need at least to create the welcome message.

11.1 Mail aliases

For each new list, it is necessary to create three mail aliases (the location of the `sendmail` `sendmail` alias file varies from one system to another).

For example, to create the `mylist` list, the following aliases must be added :

```
mylist :              "|/home/sympa/bin/queue mylist"
mylist-request :      "|/home/sympa/bin/queue mylist-request"
mylist-editor :       "|/home/sympa/bin/queue mylist-editor"
mylist-owner :        "|/home/sympa/bin/bouncequeue mylist
mylist-subscribe :    "|/home/sympa/bin/queue mylist-subscribe"
mylist-unsubscribe :  "|/home/sympa/bin/queue mylist-unsubscribe"
```

The address `mylist-request` should correspond to the person responsible for managing `mylist` (the owner). *Sympa* will forward messages for `mylist-request` to the owner of `mylist`, as defined in the `~sympa/expl/mylist/config` file. Using this feature means you would not need to modify the alias file if the owner of the list were to change.

Similarly, the address `mylist-editor` can be used to contact the list editors if any are defined in `~sympa/expl/mylist/config`. This address definition is not compulsory.

The address `mylist-owner` is the address receiving non-delivery reports. The `bouncequeue` program stores these messages in the `queuebounce` directory. *WWSympa* ((see 1.2, page 13) may then analyze them and provide a web access to them.

The address `mylist-subscribe` is an address enabling users to subscribe in a manner which can easily be explained to them. Beware : subscribing this way is so straightforward that you may find spammers subscribing to your list by accident.

The address `mylist-unsubscribe` is the equivalent for unsubscribing. By the way, the easier it is for users to unsubscribe, the easier it will be for you to manage your list !

11.2 List directory

Each list has its own directory whose name defines the list name. We recommend creating it with the same name as the alias. This directory is located in `~sympa/expl` (or any other `home` which you might have defined in the`/etc/sympa.conf` file).

Here is a list of files/directories you may find in the list directory :

```
archives/
bye.tpl
config
info
invite.tpl
homepage
message.header
message.footer
reject.tpl
remind.tpl
```

```
removed.tpl
stats
subscribers
welcome.tpl
```

11.3 List configuration file

The configuration file for the `mylist` list is named `~sympa/expl/mylist/config`. *Sympa* reads it into memory the first time the list is referred to. This file is not rewritten by *Sympa*, so you may put comment lines in it. It is possible to change this file when the program is running. Changes are taken into account the next time the list is accessed. Be careful to provide read access for *Sympa* to this file !

You will find a few configuration files in the `sample` directory. Copy one of them to `~sympa/expl/mylist/config` and customize it.

List configuration parameters are described in the list creation section, 13, page 111.

11.4 Examples of configuration files

This first example is for a list open to everyone :

```
subject First example (an open list)

visibility noconceal

owner
email Pierre.David@prism.uvsq.fr

send public

review public
```

The second example is for a moderated list with authenticated subscription :

```
subject Second example (a moderated list)

visibility noconceal

owner
email moi@ici.fr
```

```
editor
email big.prof@ailleurs.edu

send editor

subscribe auth

review owner

reply_to_header
value list

cookie 142cleliste
```

The third example is for a moderated list, with subscription controlled by the owner, and running in digest mode. Subscribers who are in digest mode receive messages on Mondays and Thursdays.

```
owner
email moi@ici.fr

editor
email prof@ailleurs.edu

send editor

subscribe owner

review owner

reply_to_header
value list

digest 1,4 12:00
```

11.5 Subscribers file

WARNING : *Sympa* will not use this file if the list is configured with `include` or `database user_data_source`.

The `~sympa/expl/mylist/subscribers` file is automatically created and populated. It contains information about list subscribers. It is not advisable to edit this file. Main parameters are :

– `email` *address*
  E-mail address of subscriber.
– `gecos` *data*
  Information about subscriber (last name, first name, etc.) This parameter is optional at subscription time.
– `reception nomail | digest | summary | notice | txt | html | urlize | not_me |`
  Special receive modes which the subscriber may select. Special modes can be either *nomail*, *digest*, *summary*, *notice*, *txt*, *html*, *urlize*, *not_me* . In normal receive mode, the receive attribute for a subscriber is not displayed. See the `SET LISTNAME SUMMARY` (17.1, page 149), the `SET LISTNAME NOMAIL` command (17.1, page 149), and the `digest` parameter (13.4.7, page 128).
– `visibility conceal`
  Special mode which allows the subscriber to remain invisible when a `REVIEW` command is issued for the list. If this parameter is not declared, the subscriber will be visible for `REVIEW`. Note : this option does not affect the results of a `REVIEW` command issued by an owner. See the `SET LISTNAME MAIL` command (17.1, page 149) for details.

11.6 Info file

`~sympa/expl/mylist/info` should contain a detailed text description of the list, to be displayed by the `INFO` command. It can also be referenced from template files for service messages.

11.7 Homepage file

`~sympa/expl/mylist/homepage` is the HTML text on the *WWSympa* info page for the list.

11.8 List template files

These files are used by Sympa as service messages for commands such as `SUB`, `ADD`, `SIG`, `DEL`, `REJECT`. These files are interpreted (parsed) by *Sympa* and respect the template format ; every file has the .tpl extension. See 10.1, page 81.

Sympa looks for these files in the following order :

1. `~sympa/expl/mylist/<file>.tpl`
2. `~sympa/etc/templates/<file>.tpl`.
3. `~sympa/bin/etc/templates/<file>.tpl`.

If the file starts with a From : line, it is taken to be a full message and will be sent (after parsing) without the addition of SMTP headers. Otherwise the file is treated as a text/plain message body.

The following variables may be used in list template files :

- [conf->email] : sympa e-mail address local part
- [conf->domain] : sympa robot domain name
- [conf->sympa] : sympa's complete e-mail address
- [conf->wwsympa_url] : *WWSympa* root URL
- [conf->listmaster] : listmaster e-mail addresses
- [list->name] : list name
- [list->host] : list hostname (default is sympa robot domain name)
- [list->lang] : list language
- [list->subject] : list subject
- [list->owner] : list owners table hash
- [user->email] : user e-mail address
- [user->gecos] : user gecos field (usually his/her name)
- [user->password] : user password
- [user->lang] : user language
- [execution_date] : the date when the scenario is executed

You may also dynamically include a file from a template using the [INCLUDE] directive.

Example :

```
Dear [user->email],

Welcome to list [list->name]@[list->host].

Presentation of the list :
[INCLUDE 'info']

The owners of [list->name] are :
[FOREACH ow IN list->owner]
   [ow->gecos] <[ow->email]>
[END]
```

11.8.1 welcome.tpl

Sympa will send a welcome message for every subscription. The welcome message can be customized for each list.

11.8.2 bye.tpl

Sympa will send a farewell message for each SIGNOFF mail command received.

11.8.3 removed.tpl

This message is sent to users who have been deleted (using the `DELETE` command) from the list by the list owner.

11.8.4 reject.tpl

Sympa will send a reject message to the senders of messages rejected by the list editor. If the editor prefixes her `REJECT` with the keyword QUIET, the reject message will not be sent.

11.8.5 invite.tpl

This message is sent to users who have been invited (using the `INVITE` command) to subscribe to a list.

You may use additional variables
- [requested_by] : e-mail of the person who sent the `INVITE` command
- [url] : the mailto : URL to subscribe to the list

11.8.6 remind.tpl

This file contains a message sent to each subscriber when one of the list owners sends the `REMIND` command (see 17.2, page 150).

11.8.7 summary.tpl

Template for summaries (reception mode close to digest), see 17.1, page 149.

11.9 Stats file

`~sympa/expl/mylist/stats` is a text file containing statistics about the list. Data are numerics separated by white space within a single line :

– Number of messages sent, used to generate X-sequence headers
– Number of messages X number of recipients
– Number of bytes X number of messages
– Number of bytes X number of messages X number of recipients
– Number of subscribers

11.10 List model files

These files are used by *Sympa* to create task files. They are interpreted (parsed) by the task manager and respect the task format. See 10.10, page 92.

11.10.1 remind.annual.task

Every year *Sympa* will send a message (the template `remind.tpl`) to all subscribers of the list to remind them their subscription.

11.10.2 expire.annual.task

Every month *Sympa* will delete subscribers older than one year who haven't answered two warning messages.

11.11 Message header and footer

You may create `~sympa/expl/mylist/message.header` and `~sympa/expl/mylist/message.footer` files. Their content is added, either at the beginning or at the end of each message before the distribution process.

The `footer_type` list parameter defines whether to attach the header/footer content as a MIME part (except for multipart/alternative messages), or to append them to the message body (for text/plain messages).

11.11.1 Archive directory

The `~sympa/expl/mylist/archives/` directory contains the archived messages for lists which are archived ; see 13.6.1, page 131. The files are named in accordance with the archiving frequency defined by the `archive` parameter.

Chapitre 12

Creating and editing mailing using the web

The management of mailing lists by list owners will usually be done via the web interface. This is based on a strict definition of privileges which pertain respectively to the listmaster, to the main list owner, and to basic list owners. The goal is to allow each listmaster to define who can create lists, and which parameters may be set by owners. Therefore, a complete installation requires some careful planning, although default values should be acceptable for most sites.

Some features are already available, others will be shortly, as specified in the documentation.

12.1 List creation

Listmasters have all privileges. Currently the listmaster is defined in `sympa.conf` but in the future, it might be possible to define one listmaster per virtual robot. By default, newly created lists must be activated by the listmaster. List creation is possible for all intranet users (i.e. : users with an e-mail address within the same domain as Sympa). This is controlled by the `create_list` scenario.

List creation request message and list creation notification message are both templates that you can customize (`create_list_request.tpl` and `list_created.tpl`).

12.1.1 Who can create lists

It is defined by `create_list` sympa.conf parameter (see 4.1.7, page 33). This parameter refers to a **create_list** scenario. It will determine if the *create list* button is displayed, if it requires a listmaster confirmation.

The scenario can accepts any condition concerning the [sender] (ie WWSympa user), and it returns `reject`, `do_it` or `listmaster` as an action.

Only in cases where a user is authorized by the create_list scenario will the ”create” button be available in the main menu. If the scenario returns `do_it`, the list will be created and installed. If the scenario returns ”listmaster”, the user is allowed to create a list, but the list is created with the `pending` status, which means that only the list owner may view or use it. The listmaster will need to open the list of pending lists using the ”pending list” button in the ”server admin” menu in order to install or refuse a pending list.

12.1.2 typical list profile

Mailing lists can have many different uses. *Sympa* offers a wide choice of parameters to adapt a list’s behavior to different situations. Users might have difficulty selecting all the correct parameters, so instead the create list form asks the list creator simply to choose a profile for the list, and to fill in the owner’s e-mail and the list subject together with a short description.

List profiles can be stored in `~sympa/etc/create_list_templates` or `~sympa/bin/etc/create_list_templates`, which are part of the Sympa distribution and should not be modified. `~sympa/etc/create_list_templates`, which will not be overwritten by make install, is intended to contain site customizations.

A list profile is an almost complete list configuration, but with a number of missing fields (such as owner e-mail) to be replaced by WWSympa at installation time. It is easy to create new list templates by modifying existing ones. Contributions to the distribution are welcome.

You might want to hide or modify profiles (not useful, or dangerous for your site). If a profile exists both in the local site directory `~sympa/etc/create_list_templates` and `~sympa/bin/etc/create_list_templates` directory, then the local profile will be used by WWSympa.

Another way to control publicly available profiles is to edit the `create_list.conf` file (the default for this file is in the `~sympa/bin/etc/` directory, and you may create your own customized version in `~sympa/etc/`). This file controls which of the available list templates are to be displayed. Example :

```
# Do not allow the public_anonymous profile
```

```
public_anonymous hidden
* read
```

When a list is created, whatever its status (`pending` or `open`), the owner can use WWSympa admin features to modify list parameters, or to edit the welcome message, and so on.

WWSympa logs the creation and all modifications to a list as part of the list's `config` file (and old configuration files are saved).

12.1.3 creating list alias

If you defined an alias_manager in `wwsympa.conf` (see 5.3.1, page 51), *WWSympa* will run this script for installing aliases. You can write your own alias_manager script, adapted to your MTA or mail configuration, provided that it recognizes the same set of parameters.

12.2 List edition

For each parameter, you may specify (via the `~sympa/etc/edit_list.conf` configuration file) who has the right to edit the parameter concerned; the default `~sympa/bin/etc/edit_list.conf` is reasonably safe.

```
Each line is a set of 3 field
<Parameter> <Population> <Privilege>
<Population> : <listmaster|privileged_owner|owner>
<Privilege> : <write|read|hidden>
parameter named "default" means any other parameter
```

There is no hierarchical relation between populations in this configuration file. You need to explicitely list populations.

Eg : listmaster will not match rules refering to owner or privileged_owner

```
     examples :

# only listmaster can edit user_data_source, priority, ...
user_data_source listmaster write

priority  owner,privileged_owner  read
priority  listmaster  write

# only privileged owner can modify  editor parameter, send, ...
```

```
editor privileged_owner write

send  owner  read
send  privileged_owner,listmaster  write

# other parameters can be changed by simple owners
default  owner  write
```

Privileged owners are defined in the list's `config` file as follows :

```
owner
email owners.email@foo.bar
profile privileged
```

The following rules are hard coded in WWSympa :
– only listmaster can edit the ”profile privileged” owner attribute
– owners can edit their own attributes (except profile and e-mail)
– the requestor creating a new list becomes privileged owner
– privileged owners can edit any gecos/reception/info attribute of any owner
– privileged owners can edit owners' e-mail addresses (but not privileged owners' e-mail addresses)

Sympa aims to define two levels of trust for owners (some being entitled simply to edit secondary parameters such as ”custom_subject”, others having the right to manage more important parameters), while leaving control of crucial parameters (such as the list of privileged owners and user_data_sources) in the hands of the listmaster. Consequently, privileged owners can change owners' e-mails, but they cannot grant the responsibility of list management to others without referring to the listmaster.

Chapitre 13

List configuration parameters

The configuration file is composed of paragraphs separated by blank lines and introduced by a keyword.

Even though there are a very large number of possible parameters, the minimal list definition is very short. The only required parameters are `owner` and `subject`. All other parameters have a default value.

keyword value

WARNING : configuration parameters must be separated by blank lines and BLANK LINES ONLY !

13.1 List description

13.1.1 editor

The `config` file contains one `editor` paragraph per moderator (or editor).

Example :

```
editor
email Pierre.David@prism.uvsq.fr
gecos Pierre (Universite de Versailles St Quentin)
```

Only the editor of a list is authorized to send messages to the list when the `send` para-

meter (see 13.3.8, page 123) is set to either `editor`, `editorkey`, or `editorkeyonly`. The `editor` parameter is also consulted in certain other cases (`privateoreditorkey`).

The syntax of this directive is the same as that of the `owner` parameter (see 13.1.4, page 112), even when several moderators are defined.

13.1.2 host

(Default value: `domain robot parameter`)

`host` *fully-qualified-domain-name*

Domain name of the list, default is the robot domain name set in the related `robot.conf` file or in file `/etc/sympa.conf`.

13.1.3 lang

(Default value: `lang robot parameter`)

Example :

```
lang cn-big5
```

This parameter defines the language used for the list. It is used to initialize a user's lang preference ; *Sympa* command reports are extracted from the associated message catalog.

See 10.6, page 87 for available languages.

13.1.4 owner

The `config` file contains one `owner` paragraph per owner.

Example :

```
owner
email serge.aumont@cru.fr
gecos C.R.U.
info Tel: 02 99 76 45 34
```

```
reception nomail
```

The list owner is usually the person who has the authorization to send ADD (see 17.2, page 150) and DELETE (see 17.2, page 150) commands on behalf of other users.

When the subscribe parameter (see 13.3.1, page 120) specifies a restricted list, it is the owner who has the exclusive right to subscribe users, and it is therefore to the owner that SUBSCRIBE requests will be forwarded.

There may be several owners of a single list ; in this case, each owner is declared in a paragraph starting with the owner keyword.

The owner directive is followed by one or several lines giving details regarding the owner's characteristics :

– email *address*
  Owner's e-mail address
– reception nomail
  Optional attribute for an owner who does not wish to receive mails. Useful to define an owner with multiple e-mail addresses : they are all recognized when *Sympa* receives mail, but thanks to reception nomail, not all of these addresses need receive administrative mail from *Sympa*.
– gecos *data*
  Public information on the owner
– info *data*
  Available since release 2.3
  Private information on the owner
– profile privileged | normal
  Available since release 2.3.5
  Profile of the owner. This is currently used to restrict access to some features of WWSympa, such as adding new owners to a list.

13.1.5 subject

subject *subject-of-the-list*

This parameter indicates the subject of the list, which is sent in response to the LISTS mail command. The subject is a free form text limited to one line. This parameter is *not* used by *Sympa* if the ~sympa/expl/lists file (a static list of lists) exists.

13.1.6 topics

topics computing/internet,education/university

This parameter allows the classification of lists. You may define multiple topics as well as hierarchical ones. *WWSympa*'s list of public lists uses this parameter.

13.1.7 visibility

(Default value: `conceal`)

`visibility` parameter is defined by scenario (see 10.8, page 88)

This parameter indicates whether the list should feature in the output generated in response to a `LISTS` command. This parameter is *not* used by *Sympa* if the `~sympa/expl/lists` file (a static list of lists) exists.

- `visibility conceal`
  conceal unless for subscribers
- `visibility intranet`
  intranet access
- `visibility noconceal`
  no conceal
- `visibility private`
  subscribers
- `visibility secret`
  conceal even for subscribers

13.2 Data source related

13.2.1 user_data_source

(Default value: `file|database, if using an RDBMS`)

`user_data_source file | database | include`

Sympa allows the mailing list manager to choose how *Sympa* loads subscriber data. Subscriber information can be stored in a text file or relational database, or included from various external sources (list, flat file, result of LDAP or SQL query).

- `user_data_source file`
  When this value is used, subscriber data are stored in a file whose name is defined by the `subscribers` parameter in `sympa.conf`. This is maintained for backward compatibility.
- `user_data_source database`
  This mode was been introduced to enable data to be stored in a relational database, in

order, for example, for subscriber data to be shared with an HTTP interface, or simply to facilitate the administration of very large mailing lists. It has been tested with MySQL, using a list of 200 000 subscribers. We strongly recommend the use of database in place of text files. It will improve performance, and solve possible conflicts between *Sympa* and *WWSympa*. Please refer to the ¨*Sympa*and its databaseëection (6, page 57).

– `user_data_source include`
  Here, subscribers are not defined *extensively* (enumeration of their e-mail addresses) but *intensively* (definition of criteria subscribers must satisfy). Includes can be performed by extracting e-mail addresses using an SQL or LDAP query, or by including other mailing lists. At least one include paragraph, defining a data source, is needed. Valid include paragraphs (see below) are `include_file`, `include_list`, `include_sql_query` and `include_ldap_query`.

13.2.2 ttl

(Default value: `3600`)

`ttl delay_in_seconds`

Sympa caches user data extracted using the include parameter. Their TTL (time-to-live) within *Sympa* can be controlled using this parameter. The default value is 3600.

13.2.3 include_list

`include_list listname`

This parameter will be interpreted only if `user_data_source` is set to `include`. All subscribers of list `listname` become subscribers of the current list. You may include as many lists as required, using one `include_list listname` line for each included list. Any list at all may be included ; the `user_data_source` definition of the included list is irrelevant, and you may therefore include lists which are also defined by the inclusion of other lists. Be careful, however, not to include list `A` in list `B` and then list `B` in list `A`, since this will give rise an infinite loop.

13.2.4 include_sql_query

`include_sql_query`

This parameter will be interpreted only if the `user_data_source` value is set to `include`, and is used to begin a paragraph defining the SQL query parameters :

– `db_type` *dbd_name*
  The type of database (mysql, Pg, Oracle, Sybase, CSV ...). This value identifies the PERL DataBase Driver (DBD) to be used, and is therefore case-sensitive.
– `host` *hostname*
  The Database Server *Sympa* will try to connect to.
– `db_name` *sympa_db_name*
  The hostname of the database system.
– `connect_options` *option1=x ;option2=y*
  These options ar appended to the connect string. This parameter is optional.
  Example :

```
user_data_source include

include_sql_query
      db_type mysql
      host sqlserv.admin.univ-x.fr
      user stduser
      passwd mysecret
      db_name studentbody
      sql_query SELECT DISTINCT email FROM student
      connect_options mysql_connect_timeout=5
```

  Connexion timeout is set to 5 seconds.
– `db_env` *list_of_var_def*
  Sets a list of environment variables to set before database connexion. This is a ';' separated list of variable assignment.
  Example for Oracle :

```
db_env ORACLE_TERM=vt100;ORACLE_HOME=/var/hote/oracle/7.3.4
```

– `user` *user_id*
  The user id to be used when connecting to the database.
– `passwd` *some secret*
  The user passwd for `user`.
– `sql_query` *a query string* The SQL query string. No fields other than e-mail addresses should be returned by this query !

Example :

```
user_data_source include

include_sql_query
      db_type oracle
      host sqlserv.admin.univ-x.fr
      user stduser
      passwd mysecret
      db_name studentbody
```

```
sql_query SELECT DISTINCT email FROM student
```

13.2.5 include_ldap_query

`include_ldap_query`

This paragraph defines parameters for a LDAP query returning a list of subscribers. This paragraph is used only if `user_data_source` is set to `include`. This feature requires the `Net : :LDAP` (perlldap) PERL module.

- `host` *ldap_directory_hostname*
  Name of the LDAP directory host.
- `port` *ldap_directory_port* (Default 389)
  Port on which the Directory accepts connections.
- `user` *ldap_user_name*
  Username with read access to the LDAP directory.
- `passwd` *LDAP_user_password*
  Password for `user`.
- `suffix` *directory name*
  Defines the naming space covered by the search (optional, depending on the LDAP server).
- `timeout` *delay_in_seconds*
  Timeout when connecting the remote server.
- `filter` *search_filter*
  Defines the LDAP search filter (RFC 2254 compliant).
- `attrs` *mail_attribute* (Default value: `mail`)
  The attribute containing the e-mail address(es) in the returned object.
- `select` *first* / *all* (Default value: `first`)
  Defines whether to use only the first address, or all the addresses, in cases where multiple values are returned.
- `scope` *base* / *one* / *sub* (Default value: `sub`)
  By default the search is performed on the whole tree below the specified base object. This may be chaned by specifying a scope parameter with one of the following values.
  - **base** : Search only the base object.
  - **one** : Search the entries immediately below the base object.
  - **sub** : Search the whole tree below the base object.

Example :

```
include_ldap_query
host ldap.cru.fr
suffix dc=cru, dc=fr
timeout 10
```

```
filter (&(cn=aumont) (c=fr))
attrs mail
select first
scope one
```

13.2.6 include_ldap_2level_query

`include_ldap_2level_query`

This paragraph defines parameters for a two-level LDAP query returning a list of subscribers. Usually the first-level query returns a list of DNs and the second-level queries convert the DNs into e-mail addresses. This paragraph is used only if `user_data_source` is set to `include`. This feature requires the `Net : :LDAP` (perlldap) PERL module.

- `host` *ldap_directory_hostname*
  Name of the LDAP directory host.
- `port` *ldap_directory_port* (Default 389)
  Port on which the Directory accepts connections.
- `user` *ldap_user_name*
  Username with read access to the LDAP directory.
- `passwd` *LDAP_user_password*
  Password for `user`.
- `suffix1` *directory name*
  Defines the naming space covered by the first-level search (optional, depending on the LDAP server).
- `timeout1` *delay_in_seconds*
  Timeout for the first-level query when connecting to the remote server.
- `filter1` *search_filter*
  Defines the LDAP search filter for the first-level query (RFC 2254 compliant).
- `attrs1` *attribute*
  The attribute containing the data in the returned object that will be used for the second-level query. This data is referenced using the syntax “[attrs1]”.
- `select1` *first* / *all* / *regex* (Default value: `first`)
  Defines whether to use only the first attribute value, all the values, or only those values matching a regular expression.
- `regex1` *regular_expression* (Default value:)
  The Perl regular expression to use if “select1” is set to “regex”.
- `scope1` *base* / *one* / *sub* (Default value: `sub`)
  By default the first-level search is performed on the whole tree below the specified base object. This may be changed by specifying a scope parameter with one of the following values.
  - **base** : Search only the base object.
  - **one** : Search the entries immediately below the base object.
  - **sub** : Search the whole tree below the base object.
- `suffix2` *directory name*

Defines the naming space covered by the second-level search (optional, depending on the LDAP server). The "[attrs1]" syntax may be used to substitute data from the first-level query into this parameter.

– `timeout2` *delay_in_seconds*
  Timeout for the second-level queries when connecting to the remote server.
– `filter2` *search_filter*
  Defines the LDAP search filter for the second-level queries (RFC 2254 compliant). The "[attrs1]" syntax may be used to substitute data from the first-level query into this parameter.
– `attrs2` *mail_attribute* (Default value: `mail`)
  The attribute containing the e-mail address(es) in the returned objects from the second-level queries.
– `select2` *first* | *all* | *regex* (Default value: `first`)
  Defines whether to use only the first address, all the addresses, or only those addresses matching a regular expression in the second-level queries.
– `regex2` *regular_expression* (Default value:)
  The Perl regular expression to use if "select2" is set to "regex".
– `scope2` *base* | *one* | *sub* (Default value: `sub`)
  By default the second-level search is performed on the whole tree below the specified base object. This may be changed by specifying a scope2 parameter with one of the following values.
  – **base** : Search only the base object.
  – **one** : Search the entries immediately below the base object.
  – **sub** : Search the whole tree below the base object.

Example : (cn=testgroup,dc=cru,dc=fr should be a groupOfUniqueNames here)

```
include_ldap_2level_query
host ldap.cru.fr
suffix1 cn=testgroup, dc=cru, dc=fr
timeout1 10
filter1 (objectClass=*)
attrs1 uniqueMember
select1 all
scope1 base
suffix2 dc=cru, dc=fr
timeout2 10
filter2 (&(dn=[attrs1]) (c=fr))
attrs2 mail
select2 regex
regex2 ^*@cru.fr$
scope2 one
```

13.2.7 include_file

`include_file` `path to file`

This parameter will be interpreted only if the `user_data_source` value is set to `include`. The file should contain one e-mail address per line (lines beginning with a "#" are ignored).

13.3 Command related

13.3.1 subscribe

(Default value: `open`)

`subscribe` parameter is defined by scenario (see 10.8, page 88)

The `subscribe` parameter defines the rules for subscribing to the list. Predefined scenarii are :

- `subscribe auth`
  subscription request confirmed
- `subscribe auth_notify`
  need authentication (notification is sent to owners)
- `subscribe auth_owner`
  requires authentication then owner approval
- `subscribe closed`
  subscribe is impossible
- `subscribe intranet`
  restricted to local domain users
- `subscribe intranetorowner`
  local domain users or owner approval
- `subscribe open`
  for anyone without authentication
- `subscribe open_notify`
  anyone, notification is sent to list owner
- `subscribe open_quiet`
  anyone, no welcome message
- `subscribe owner`
  owners approval
- `subscribe smime`
  requires S/MIME signed
- `subscribe smimeorowner`
  requires S/MIME signed or owner approval

13.3.2 unsubscribe

(Default value: `open`)

`unsubscribe` parameter is defined by scenario (see 10.8, page 88)

This parameter specifies the unsubscription method for the list. Use `open_notify` or `auth_notify` to allow owner notification of each unsubscribe command. Predefined scenarii are :

- `unsubscribe auth`
  need authentication
- `unsubscribe auth_notify`
  authentication requested, notification sent to owner
- `unsubscribe closed`
  impossible
- `unsubscribe open`
  anyone without authentication
- `unsubscribe open_notify`
  without authentication, notification is sent to owners
- `unsubscribe owner`
  owner approval

13.3.3 add

(Default value: `owner`)

`add` parameter is defined by scenario (see 10.8, page 88)

This parameter specifies who is authorized to use the `ADD` command. Predefined scenarii are :

- `add auth`
  add need authentication
- `add closed`
  add impossible
- `add owner`
  add perform by list owner do not need authentication
- `add owner_notify`
  add performed by owner do not need authentication (notification)

13.3.4 del

(Default value: `owner`)

`del` parameter is defined by scenario (see 10.8, page 88)

This parameter specifies who is authorized to use the `DEL` command. Predefined scenarii are :

- `del auth`
  deletion performed only by list owners, need authentication
- `del closed`
  remove subscriber impossible
- `del owner`
  by owner without authentication
- `del owner_notify`
  list owners, authentication not needed (notification)

13.3.5 remind

(Default value: `owner`)

`remind` parameter is defined by scenario (see 10.8, page 88)

This parameter specifies who is authorized to use the `remind` command. Predefined scenarii are :

- `remind listmaster`
  listmaster only
- `remind owner`
  restricted to list owner

13.3.6 remind_task

(Default value: `no default value`)

This parameter states which model is used to create a `remind` task. A `remind` task regurlaly sends to the subscribers a message which reminds them their subscription to list.

example :

remind annual

13.3.7 expire_task

(Default value: `no default value`)

This parameter states which model is used to create a `remind` task. A `expire` task regurlaly checks the inscription or reinscription date of subscribers and asks them to renew their subscription. If they don't they are deleted.

example :

expire annual

13.3.8 send

(Default value: `private`)

`send` parameter is defined by scenario (see 10.8, page 88)

This parameter specifies who can send messages to the list. Valid values for this parameter are pointers to *scenarii*.

- `send closed`
  closed
- `send editor`
  Moderated, old style
- `send editorkey`
  Moderated
- `send editorkeyonly`
  Moderated, even for moderators
- `send editorkeyonlyauth`
  Moderated, with editor confirmation
- `send intranet`
  restricted to local domain
- `send intranetorprivate`
  restricted to local domain and subscribers
- `send newsletter`
  Newsletter, restricted to moderators
- `send newsletterkeyonly`
  Newsletter, restricted to moderators after confirmation
- `send private`
  restricted to subscribers
- `send private_smime`
  restricted to subscribers check smime signature
- `send privateandeditorkey`
  Moderated, restricted to subscribers

- `send privateandnomultipartoreditorkey`
  Moderated, for non subscribers sending multipart messages
- `send privatekey`
  restricted to subscribers with previous md5 authentication
- `send privatekeyandeditorkeyonly`
  Moderated, for subscribers and moderators
- `send privateoreditorkey`
  Private, moderated for non subscribers
- `send privateorpublickey`
  Private, confirmation for non subscribers
- `send public`
  public list
- `send public_nobcc`
  public list Bcc rejected (anti-spam)
- `send publickey`
  anyone with previous md5 authentication
- `send publicnoattachment`
  public list multipart/mixed messages are forwarded to moderator
- `send publicnomultipart`
  public list multipart messages are rejected

13.3.9 review

(Default value: `owner`)

`review` parameter is defined by scenario (see 10.8, page 88)

This parameter specifies who can use `REVIEW` (see 17.1, page 148), administrative requests.

Predefined scenarii are :

- `review closed`
  nobody can review
- `review intranet`
  restricted to local domain users
- `review listmaster`
  listmaster only
- `review owner`
  only owner (and listmaster)
- `review private`
  restricted to subscribers
- `review public`
  anyone can do it !

13.3.10 shared_doc

This paragraph defines read and edit access to the shared document repository.

d_read

(Default value: `private`)

`d_read` parameter is defined by scenario (see 10.8, page 88)

This parameter specifies who can read shared documents (access the contents of a list's `shared` directory).

Predefined scenarii are :

– `d_read owner`
  restricted to list owners and document author
– `d_read private`
  restricted to subscribers
– `d_read public`
  public documents

d_edit

(Default value: `owner`)

`d_edit` parameter is defined by scenario (see 10.8, page 88)

This parameter specifies who can perform changes within a list's `shared` directory (i.e. upload files and create subdirectories).

Predefined scenarii are :

– `d_edit owner`
  restricted to list owners and document author
– `d_edit private`
  restricted to subscribers
– `d_edit public`
  public documents

Example :

```
shared_doc
d_read public
```

```
d_edit private
```

quota

`quota` *number-of-Kbytes*

This parameter specifies the disk quota for the document repository, in kilobytes. If quota is exceeded, file uploads fail.

13.4 List tuning

13.4.1 reply_to_header

The `reply_to_header` parameter starts a paragraph defining what *Sympa* will place in the `Reply-To:` SMTP header field of the messages it distributes.

- `value sender | list | all | other_email` (Default value: `sender`)
  This parameter indicates whether the `Reply-To:` field should indicate the sender of the message (`sender`), the list itself (`list`), both list and sender (`all`) or an arbitrary e-mail address (defined by the `other_email` parameter).
  Note : it is inadvisable to change this parameter, and particularly inadvisable to set it to `list`. Experience has shown it to be almost inevitable that users, mistakenly believing that they are replying only to the sender, will send private messages to a list. This can lead, at the very least, to embarrassment, and sometimes to more serious consequences.
- `other_email` *an_email_address*
  If `value` was set to `other_email`, this parameter defines the e-mail address used.
- `apply respect | forced` (Default value: `respect`)
  The default is to respect (preserve) the existing `Reply-To:` SMTP header field in incoming messages. If set to `forced`, `Reply-To:` SMTP header field will be overwritten.

Example :

```
reply_to_header
value other_email
other_email listowner@my.domain
apply forced
```

13.4.2 max_size

(Default value: `max_size robot parameter`)

`max_size` *number-of-bytes*

Maximum size of a message in 8-bit bytes. The default value is set in the `/etc/sympa.conf` file.

13.4.3 anonymous_sender

`anonymous_sender` *value*

If this parameter is set for a list, all messages distributed via the list are rendered anonymous. SMTP `From :` headers in distributed messages are altered to contain the value of the `anonymous_sender` parameter. Various other fields are removed (`Received :, Reply-To :, Sender :, X-Sender :, Message-id :, Resent-From :`

13.4.4 custom_header

`custom_header` *header-field* `:` *value*

This parameter is optional. The headers specified will be added to the headers of messages distributed via the list. As of release 1.2.2 of *Sympa*, it is possible to put several custom header lines in the configuration file at the same time.

Example: `custom_header X-url : http ://www.cru.fr/listes/apropos/sedesabonner.faq.html`.

13.4.5 custom_subject

`custom_subject` *value*

This parameter is optional. It specifies a string which is added to the subject of distributed messages (intended to help users who do not use automatic tools to sort incoming messages).

Example: `custom_subject [sympa-users]`.

13.4.6 footer_type

(Default value: `mime`)

`footer_type (optional, default value is mime) mime | append`

List owners may decide to add message headers or footers to messages sent via the list. This parameter defines the way a footer/header is added to a message.

– `footer_type mime`
  The default value. Sympa will add the footer/header as a new MIME part. If the message is in multipart/alternative format, no action is taken (since this would require another level of MIME encapsulation).
– `footer_type append`
  Sympa will not create new MIME parts, but will try to append the header/footer to the body of the message. `~sympa/expl/mylist/message.footer.mime` will be ignored. Headers/footers may be appended to text/plain messages only.

13.4.7 digest

`digest` *daylist hour* :*minutes*

Definition of `digest` mode. If this parameter is present, subscribers can select the option of receiving messages in multipart/digest MIME format. Messages are then grouped together, and compilations of messages are sent to subscribers in accordance with the rythm selected with this parameter.

Daylist designates a list of days in the week in number format (from 0 for Sunday to 6 for Saturday), separated by commas.

Example: `digest 1,2,3,4,5 15 :30`

In this example, *Sympa* sends digests at 3 :30 PM from Monday to Friday.

WARNING : if the sending time is too late, *Sympa* may not be able to process it. It is essential that *Sympa* should scan the digest queue at least once between the time laid down for sending the digest and 12 :00 AM (midnight). As a rule of thumb, do not use a digest time later than 11 :00 PM.

13.4.8 available_user_options

The `available_user_options` parameter starts a paragraph to define available options for the subscribers of the list.

– `reception` *modelist*
  (Default value: `reception mail,notice,digest,summary,nomail`)
  modelist is a list of modes (mail, notice, digest, summary, nomail), separated by commas. Only these modes will be allowed for the subscribers of this list. If a subscriber has a reception mode not in the list, sympa uses the mode specified in the *default_user_options* paragraph.

Example :

```
## Nomail reception mode is not available
available_user_options
reception   digest,mail
```

13.4.9 default_user_options

The `default_user_options` parameter starts a paragraph to define a default profile for the subscribers of the list.

– `reception` `notice | digest | summary | nomail | mail`
  Mail reception mode.
– `visibility` `conceal | noconceal`
  Visibility of the subscriber with the `REVIEW` command.

Example :

```
default_user_options
reception   digest
visibility noconceal
```

13.4.10 cookie

(Default value: `cookie robot parameter`)

`cookie` *random-numbers-or-letters*

This parameter is a confidential item for generating authentication keys for administrative commands (`ADD`, `DELETE`, etc.). This parameter should remain concealed, even for owners. The cookie is applied to all list owners, and is only taken into account when the owner has the `auth` parameter (`owner` parameter, see 13.1.4, page 112).

Example: `cookie secret22`

13.4.11 priority

(Default value: `default_list_priority robot parameter`)

`priority` *0-9*

The priority with which *Sympa* will process messages for this list. This level of priority is applied while the message is going through the spool.

0 is the highest priority. The following priorities can be used : `0...9 z`. `z` is a special priority causing messages to remain spooled indefinitely (useful to hang up a list).

Available since release 2.3.1.

13.5 Bounce related

13.5.1 bounce

This paragraph defines bounce management parameters :

- `warn_rate`
  (Default value: `bounce_warn_rate robot parameter`)
  The list owner receives a warning whenever a message is distributed and the number (percentage) of bounces exceeds this value.
- `halt_rate`
  (Default value: `bounce_halt_rate robot parameter`)
  `NOT USED YET`
  If bounce rate reaches the `halt_rate`, messages for the list will be halted, i.e. they are retained for subsequent moderation. Once the number of bounces exceeds this value, messages for the list are no longer distributed.
- `expire_bounce_task`
  (Default value: `d`)aily
  Name of the task template use to remove old bounces. Usefull to remove bounces for a subscriber email if some message are distributed without receiving new bounce. In this case, the subscriber email seems to be OK again. Active if task_manager.pl is running.

Example :

```
## Owners are warned with 10% bouncing addresses
## message distribution is halted with 20% bouncing rate
bounce
warn_rate 10
halt_rate 20
```

13.5.2 welcome_return_path

(Default value: `welcome_return_path robot parameter`) `welcome_return_path` unique | owner

If set to `unique`, the welcome message is sent using a unique return path in order to re-

move the subscriber immediately in the case of a bounce. See welcome_return_path sympa.conf parameter (4.8.3, page 42).

13.5.3 remind_return_path

(Default value: remind_return_path robot parameter) remind_return_path unique | owner

Same as welcome_return_path, but applied to remind messages. See remind_return_path sympa.conf parameter (4.8.4, page 42).

13.6 Archive related

Sympa maintains 2 kinds of archives : mail archives and web archives.

Mail archives can be retreived via a mail command send to the robot, they are stored in ~sympa/expl/mylist/archives/ directory.

Web archives are accessed via the web interface (with access control), they are stored in a directory defined in wwsympa.conf.

13.6.1 archive

If the config file contains an archive paragraph *Sympa* will manage an archive for this list.

Example :

```
archive
period week
access private
```

If the archive parameter is specified, archives are accessible to users through the GET command, and the index of the list archives is provided in reply to the INDEX command (the last message of a list can be consulted using the LAST command).

period day | week | month | quarter | year

This parameter specifies how archiving is organized : by day, week, month, quarter, or year. Generation of automatic list archives requires the creation of an archive di-

rectory at the root of the list directory (~sympa/expl/mylist/archives/), used to store these documents.

access private | public | owner | closed |

This parameter specifies who is authorized to use the GET, LAST and INDEX commands.

13.6.2 web_archive

If the config file contains a web_archive paragraph *Sympa* will copy all messages distributed via the list to the "queueoutgoing" spool. It is intended to be used with WWSympa html archive tools. This paragraph must contain at least the access parameter to control who can browse the web archive.

Example :

```
web_archive
access private
quota 1000
```

access

access_web_archive parameter is defined by scenario (see 10.8, page 88)

Predefined scenarii are :

- access closed
  closed
- access intranet
  restricted to local domain users
- access listmaster
  listmaster
- access owner
  by owner
- access private
  subscribers only
- access public
  public

quota

quota *number-of-Kbytes*

This parameter specifies the disk quota for the list's web archives, in kilobytes. This parameter's default is `default_archive_quota` `sympa.conf` parameter. If quota is exceeded, messages are no more archived, list owner is notified. When archives are 95% full, the list owner is warned.

Chapitre 14

Shared documents

Shared documents are documents that different users can manipulate on-line via the web interface of *Sympa*, provided that the are authorized to do so. A shared space is associated with a list, and users of the list can upload, download, delete, etc, documents in the shared space.

WWSympa shared web features are fairly rudimentary. It is not our aim to provide a sophisticated tool for web publishing, such as are provided by products like *Rearsite*. It is nevertheless very useful to be able to define privilege on web documents in relation to list attributes such as *subscribers*, *list owners*, or *list editors*.

All file and directory names are lowercased by Sympa. It is consequently impossible to create two different documents whose names differ only in their case. The reason Sympa does this is to allow correct URL links even when using an HTML document generator (typically Powerpoint) which uses random case for file names !

In order to have better control over the documents in the shared space, each document is linked to a set of specific control information : its access rights. Security is thus ensured.

A list's shared documents are stored in the `~sympa/expl/mylist/shared` directory.

This chapter describes how the shared documents are managed, especially as regards their access rights. We shall see :

– the kind of operations which can be performed on shared documents
– access rights management
– access rights control specifications
– actions on shared documents
– template files

14.1 The three kind of operations on a document

Where shared documents are concerned, there are three kinds of operation which have the same constraints relating to access control :

– The read operation :

  – If a directory, open it and list its contents (only those sub-documents the user is authorized to "see").
  – If a file, download it, and if a viewable file (*text/plain*, *text/html*, or image), display it.

– The edit operation :

  – Subdirectory creation
  – File uploading
  – Description of a document (title and basic information)
  – On-line editing of a text file
  – Document (file or directory) removal. If a directory, it must be empty.

  These different edit actions are equivalent as regards access rights. Users who are authorized to edit a directory can create a subdirectory or upload a file to it, as well as describe or delete it. Users authorized to edit a file can edit it on-line, describe it, replace or remove it.

– The control operation :
  The control operation is directly linked to the notion of access rights. If we wish shared documents to be secure, we have to control the access to them. Not everybody must be authorized to do everything to them. Consequently, each document has specific access rights for reading and editing. Performing a control action on a document involves changing its Read/Edit rights.
  The control operation has more restrictive access rights than the other two operations. Only the owner of a document, the privileged owner of the list and the listmaster have control rights on a document. Another possible control action on a document is therefore specifying who owns it.

14.2 The description file

The information (title, owner, access rights...) relative to each document must be stored, and so each shared document is linked to a special file called a description file, whose name includes the `.desc` prefix.

The description file of a directory having the path `mydirectory/mysubdirectory` has the path `mydirectory/mysubdirectory/.desc` . The description file of a file having the path `mydirectory/mysubdirectory/myfile.myextension` has the path `mydirectory/mysubdirectory/.desc.myfile.myextension` .

14.2.1 Structure of description files

The structure of a document (file or directory) description file is given below. You should *never* have to edit a description file.

```
title
  <description of the file in a few words>

creation
  email        <e-mail of the owner of the document>
  date_epoch   <date_epoch of the creation of the document>

access
 read <access rights for read>
 edit <access rights for edit>
```

The following example is for a document that subscribers can read, but which only the owner of the document and the owner of the list can edit.

```
title
  module C++ which uses the class List

creation
  email foo@some.domain.com
  date_epoch 998698638

access
 read  private
 edit  owner
```

14.3 The predefined scenarii

14.3.1 The public scenario

The **public** scenario is the most permissive scenario. It enables anyone (including unknown users) to perform the corresponding action.

14.3.2 The private scenario

The **private** scenario is the basic scenario for a shared space. Every subscriber of the list is authorized to perform the corresponding action. The **private** scenario is the default read scenario for `shared` when this shared space is created. This can be modified by editing the list configuration file.

14.3.3 The scenario owner

The scenario **owner** is the most restrictive scenario for a shared space. Only the listmaster, list owners and the owner of the document (or those of a parent document) are allowed to perform the corresponding action. The **owner** scenario is the default scenario for editing.

14.4 Access control

Access control is an important operation performed every time a document within the shared space is accessed.

The access control relative to a document in the hierarchy involves an iterative operation on all its parent directories.

14.4.1 Listmaster and privileged owners

The listmaster and privileged list owners are special users in the shared web. They are allowed to perform every action on every document in the shared space. This privilege enables control over the shared space to be maintained. It is impossible to prevent the listmaster and privileged owners from performing whatever action they please on any document in the shared space.

14.4.2 Special case of the `shared` directory

In order to allow access to a root directory to be more restrictive than that of its subdirectories, the `shared` directory (root directory) is a special case as regards access control. The access rights for read and edit are those specified in the list configuration file. Control of the root directory is specific. Only those users authorized to edit a list's configuration may change access rights on its `shared` directory.

14.4.3 General case

`mydirectory/mysubdirectory/myfile` is an arbitrary document in the shared space, but not in the *root* directory. A user **X** wishes to perform one of the three operations (read, edit, control) on this document. The access control will proceed as follows :

– Read operation

  To be authorized to perform a read action on `mydirectory/mysubdirectory/myfile`, **X** must be authorized to read every

document making up the path ; in other words, she must be allowed to read `myfile` (the scenario of the description file of `myfile` must return *do_it* for user **X**), and the same goes for `mysubdirectory` and `mydirectory`).
In addition, given that the owner of a document or one of its parent directories is allowed to perform **all actions on that document**, `mydirectory/mysubdirectory/myfile` may also have read operations performed on it by the owners of `myfile`, `mysubdirectory`, and `mydirectory`.
This can be schematized as follows :

```
X can read <a/b/c>

if

(X can read <c>
AND X can read <b>
AND X can read <a>)

OR

(X owner of <c>
OR X owner of <b>
OR X owner of <a>)
```

– Edit operation
The access algorithm for edit is identical to the algorithm for read :

```
X can edit <a/b/c>

if

(X can edit <c>
AND X can edit <b>
AND X can edit <a>)

OR

(X owner of <c>
OR X owner of <b>
OR X owner of <a>)
```

– Control operation
The access control which precedes a control action (change rights or set the owner of a document) is much more restrictive. Only the owner of a document or the owners of a parent document may perform a control action :

```
X can control <a/b/c>

if

(X owner of <c>
OR X owner of <b>
OR X owner of <a>)
```

14.5 Shared document actions

The shared web feature has called for some new actions.

– action D_ADMIN
  Create the shared web, close it or restore it. The d_admin action is accessible from a list's **admin** page.
– action D_READ
  Reads the document after read access control. If a folder, lists all the subdocuments that can be read. If a file, displays it if it is viewable, else downloads it to disk. If the document to be read contains a file named `index.html` or `index.htm`, and if the user has no permissions other than read on all contained subdocuments, the read action will consist in displaying the index. The d_read action is accessible from a list's **info** page.
– action D_CREATE_DIR
  Creates a new subdirectory in a directory that can be edited. The creator is the owner of the directory. The access rights are those of the parent directory.
– action D_DESCRIBE
  Describes a document that can be edited.
– action D_DELETE
  Deletes a document after edit access control. If a folder, it has to be empty.
– action D_UPLOAD
  Uploads a file into a directory that can be edited.
– action D_OVERWRITE
  Overwrites a file if it can be edited. The new owner of the file is the one who has done the overwriting operation.
– actions D_EDIT_FILE and D_SAVE_FILE
  Edits a file and saves it after edit access control. The new owner of the file is the one who has done the saving operation.
– action D_CHANGE_ACCESS
  Changes the access rights of a document (read or edit), provided that control of this document is authorized.
– action D_SET_OWNER
  Changes the owner of a directory, provided that control of this document is authorized. The directory must be empty. The new owner can be anyone, but authentication is necessary before any action may be performed on the document.

14.6 Template files

The following template files have been created for the shared web :

14.6.1 d_read.tpl

The default page for reading a document. If a file, displays it (if viewable) or downloads it. If a directory, displays all readable subdocuments, each of which will feature buttons corresponding to the different actions this sub document allows. If the directory is editable, displays buttons to describe it, upload a file to it and, create a new subdirectory. If access to the document is editable, displays a button to edit the access to it.

14.6.2 d_editfile.tpl

The page used to edit a file. If a text file, allows it to be edited on-line. This page also enables the description of the file to be edited, or another file to be substituted in its place.

14.6.3 d_control.tpl

The page to edit the access rights and the owner of a document.

Chapitre 15

Bounce management

Sympa allows bounce (non-delivery report) management. This prevents list owners from receiving each bounce (1 per message sent to a bouncing subscriber) in their own mailbox. Without automatic processing of bounces, list owners either go mad, or just delete them without further attention.

Bounces are received at mylist-owner address, which should be sent to the `bouncequeue` program via aliases :

```
\samplelist-owner: "|/home/sympa/bin/bouncequeue \samplelist"
```

`bouncequeue` (see 2.2, page 18) stores bounces in a `~sympa/spool/bounce/` spool.

Bounces are then processed by the `bounced.pl` daemon. This daemon analyses bounces to find out which e-mail addresses are concerned and what kind of error was generated. If bouncing addresses match a subscriber's address, information is stored in the *Sympa* database (in subscriber_table). Moreover, the most recent bounce itself is archived in `bounce_path/mylist/email` (where bounce_path is defined in a `wwsympa.conf` parameter and email is the user e-mail address).

When reviewing a list, bouncing addresses are tagged as bouncing. You may access further information such as dates of first and last bounces, number of received bounces for the address, the last bounce itself.

Future development of *Sympa* should include the automatic deletion of bouncing addresses.

Chapitre 16

Antivirus

Sympa lets you use an external antivirus solution to check incoming mails. In this case you must set the `antivirus_path` and `antivirus_args` configuration parameters (see 4.13, page 47. *Sympa* is already compatible with McAfee/uvscan, Fsecure/fsav, Sophos, AVP and Trend Micro/VirusWall. For each mail received, *Sympa* extracts its MIME parts in the `~sympa/spool/tmp/antivirus` directory and then calls the antivirus software to check them. When a virus is detected, *Sympa* looks for the virus name in the virus scanner STDOUT and sends a `your_infected_msg.tpl` warning to the sender of the mail. The mail is saved as 'bad' and the working directory is deleted (except if *Sympa* is running in debug mode).

Chapitre 17

Using *Sympa* commands

Users interact with *Sympa*, of course, when they send messages to one of the lists, but also indirectly through administrative requests (subscription, list of users, etc.).

This section describes administrative requests, as well as interaction modes in the case of private and moderated lists. Administrative requests are messages whose body contains commands understood by *Sympa*, one per line. These commands can be indiscriminately placed in the `Subject:` or in the body of the message. The `To:` address is generally the `sympa@domain` alias, although it is also advisable to recognize the `listserv@domain` address.

Example :

```
From: pda@prism.uvsq.fr
To: sympa@cru.fr

LISTS
INFO sympa-users
REVIEW sympa-users
QUIT
```

Most user commands can have three-letter abbreviations (e.g. `REV` instead of `REVIEW`).

17.1 User commands

– `HELP`
  Provides instructions for the use of *Sympa* commands. The result is the content of the `helpfile.tpl` template file.
– `INFO` *listname*

Provides the welcome message for the specified list. The result is the content of `~welcome[.mime]`.

– `LISTS`
Provides the names of lists managed by *Sympa*. This list can either be generated dynamically, using the `visibility` (see 13.1.7, page 114) and `subject` list parameters (13.1.5, page 113), as well as `~sympa/expl/lists.header` and `~sympa/expl/lists.footer`). It can also be generated statically by including the contents of the `~sympa/expl/lists` file, which must be updated manually by the robot administrator.

– `REVIEW` *listname*
Provides the parameters of the specified list (owner, subscription mode, etc.), as well as the addresses of subscribers if the run mode authorizes it. See the `review` parameter (13.3.9, page 124) for the configuration file of each list, which controls consultation authorizations for the subscriber list. Since subscriber addresses can be abused by spammers, it is strongly recommended that you **only authorize owners to access the subscriber list**.

– `WHICH`
Returns the list of lists to which one is subscribed, as well as the configuration of his or her subscription to each of the lists (DIGEST, NOMAIL, SUMMARY, CONCEAL).

– `STATS` *listname*
Provides statistics for the specified list : number of messages received, number of messages sent, megabytes received, megabytes sent. This is the contents of the `~sympa/expl/stats` file.
Access to this command is controlled by `review` parameter.

– `INDEX` *listname*
Provides index of archives for specified list. Access rights to this function are the same as for the `GET` command.

– `GET` *listname archive*
To retrieve archives for list (see above). Access rights are the same as for the `REVIEW` command. See `review` parameter (13.3.9, page 124).

– `LAST` *listname*
To receive the last message distributed in a list (see above). Access rights are the same as for the `GET` command.

– `SUBSCRIBE` *listname firstname name*
Requests sign-up to the specified list. The *firstname* and *name* are optional. If the list is parameterized with a restricted subscription (see `subscribe` parameter, 13.3.1, page 120), this command is sent to the list owner for approval.

– `INVITE` *listname user@host name*
Invite someone to subscribe to the specified list. The *name* is optional. The command is similar to the `ADD` but the specified person is not added to the list but invited to subscribe to it in accordance with the `subscribe` parameter, 13.3.1, page 120).

– `SIGNOFF` *listname* [*user@host*]
Requests unsubscription from the specified list. `SIGNOFF *` means unsubscription from all lists.

– `SET` *listname* `DIGEST`
Puts the subscriber in *digest* mode for the *listname* list. Instead of receiving mail from the list in a normal manner, the subscriber will periodically receive it in a DIGEST. This DIGEST compiles a group of messages from the list, using multipart/digest mime format.

The sending period for these DIGESTS is regulated by the list owner using the `digest` parameter (see 13.4.7, page 128). See the `SET LISTNAME MAIL` command (17.1, page 149) and the `reception` parameter (11.5, page 101).

– `SET` *listname* `SUMMARY`
Puts the subscriber in *summary* mode for the *listname* list. Instead of receiving mail from the list in a normal manner, the subscriber will periodically receive the list of messages. This mode is very close to the DIGEST reception mode but the subscriber receives only the list of messages.
This option is available only if the digest mode is set.

– `SET` *listname* `NOMAIL`
Puts subscriber in *nomail* mode for the *listname* list. This mode is used when a subscriber no longer wishes to receive mail from the list, but nevertheless wishes to retain the possibility of posting to the list. This mode therefore prevents the subscriber from unsubscribing and subscribing later on. See the `SET LISTNAME MAIL` command (17.1, page 149) and the `reception` (11.5, page 101).

– `SET` *listname* `TXT`
Puts subscriber in *txt* mode for the *listname* list. This mode is used when a subscriber wishes to receive mails sent in both format txt/html and txt/plain only in txt/plain format. See the `reception` (11.5, page 101).

– `SET` *listname* `HTML`
Puts subscriber in *html* mode for the *listname* list. This mode is used when a subscriber wishes to receive mails sent in both format txt/html and txt/plain only in txt/html format. See the `reception` (11.5, page 101).

– `SET` *listname* `URLIZE`
Puts subscriber in *urlize* mode for the *listname* list. This mode is used when a subscriber wishes not to receive attached files. The attached files are replaced by an URL leading to the file stored on the list site.
See the `reception` (11.5, page 101).

– `SET` *listname* `NOT_ME`
Puts subscriber in *not_me* mode for the *listname* list. This mode is used when a subscriber wishes not to receive back the message that he has sent to the list.
See the `reception` (11.5, page 101).

– `SET` *listname* `MAIL`
Puts the subscriber in normal mode (default) for the *listname* list. This option is mainly used to cancel the *nomail*, *summary* or *digest* modes. If the subscriber was in *nomail* mode, he or she will again receive mail from the list in a normal manner. See the `SET LISTNAME NOMAIL` command (17.1, page 149) and the `reception` parameter (11.5, page 101).

– `SET` *listname* `CONCEAL`
Puts the subscriber in *conceal* mode for the *listname* list. The subscriber will then become invisible during `REVIEW` on this list. Only owners will see the whole subscriber list.
See the `SET LISTNAME NOCONCEAL` command (17.1, page 149) and the `visibility` parameter (13.1.7, page 114).

– `SET` *listname* `NOCONCEAL`
Puts the subscriber in *noconceal* mode (default) for *listname* list. The subscriber will then become visible during `REVIEW` of this list. The *conceal* mode is then cancelled. See `SET LISTNAME CONCEAL` command (17.1, page 149) and `visibility` parameter (13.1.7, page 114).

– `QUIT`

Ends acceptance of commands. This can prove useful when the message contains additional lines, as for example in the case where a signature is automatically added by the user's mail program (MUA).

– CONFIRM *key*
If the send parameter of a list is set to privatekey, publickey or privateorpublickey, messages are only distributed in the list after an authentication phase by return mail, using a one-time password (numeric key). For this authentication, the sender of the message is requested to post the "CONFIRM *key*" command to *Sympa*.

– QUIET
This command is used for silent (mute) processing : no performance report is returned for commands prefixed with QUIET).

17.2 Owner commands

Some administrative requests are only available to list owner(s). They are indispensable for all procedures in limited access mode, and to perform requests in place of users. These requests are :

– ADD *listname user@host firstname name*
Add command similar to SUBSCRIBE

– DELETE *listname user@host*
Delete command similar to SIGNOFF

– REMIND *listname* or REMIND *
REMIND is used usually by list owner in order to send an individual service message to each subscriber. This message is make by parsing the remind.tpl file.
REMIND is used to send to each subscriber of any list a single message with a summary of his/her subscriptions. In this case the message sent is constructed by parsing the global_remind.tpl file. For each list, *Sympa* tests whether the list is configured as hidden to each subscriber (parameter lparam visibility). By default the use of this command is restricted to listmasters. Processing may take a lot of time !

– EXPIRE
listname age (in days) deadline (in days) (listname) (age (in days)) (deadline (in days)) *explanatory text to be sent to the subscribers concerned*
This command activates an expiration process for former subscribers of the designated list. Subscribers for which no procedures have been enabled for more than *age* days receive the explanatory text appended to the EXPIRE command. This text, which must be adapted by the list owner for each subscriber population, should explain to the people receiving this message that they can update their subscription date so as to not be deleted from the subscriber list, within a deadline of *deadline* days.
Past this deadline, the initiator of the EXPIRE command receives the list of persons who have not confirmed their subscription. It is up to the initiator to send *Sympa* the corresponding DELETE commands.
Any operation updating the subscription date of an address serves as confirmation of subscription. This is also the case for SET option selecting commands and for the

SUBSCRIBE subscription command itself. The fact of sending a message to the list also updates the subscription date.
The explanatory message should contain at least 20 words ; it is possible to delimit it by the word QUIT, in particular in order not to include a signature, which would systematically end the command message.
A single expiration process can be activated at any given time for a given list. The EXPIRE command systematically gives rise to authentication by return mail. The EXPIRE command has **no effect on the subscriber list**.

– EXPIREINDEX *listname*
Makes it possible, at any time, for an expiration process activated using an EXPIRE command to receive the list of addresses for which no enabling has been received.
– EXPIREDEL *listname*
Deletion of a process activated using the EXPIRE command. The EXPIREDEL command has no effect on subscribers, but it possible to activate a new expiration process with new deadlines.

As above, these commands can be prefixed with QUIET to indicate processing without acknowledgment of receipt.

17.3 Moderator commands

If a list is moderated, *Sympa* only distributes messages enabled by one of its moderators (editors). Moderators have several methods for enabling message distribution, depending on the send list parameter (13.3.8, page 123).

– DISTRIBUTE *listname key*
If the send parameter of a list is set to editorkey or editorkeyonly, each message queued for moderation is stored in a spool (see 4.6.3, page 39), and linked to a key.
The moderator must use this command to enable message distribution.
– REJECT *listname key*
The message with the *key* key is deleted from the moderation spool of the *listname* list.
– MODINDEX *listname*
This command returns the list of messages queued for moderation for the *listname* list.
The result is presented in the form of an index, which supplies, for each message, its sending date, its sender, its size, and its associated key, as well as all messages in the form of a digest.

See also the recommendations for moderators[1].

[1] http ://listes.cru.fr/admin/moderation.html