Sympa
Mailing Lists Management Software
version 5.3a.10

Serge Aumont, Olivier Salaün, Christophe Wolfhugel,

21 November 2006

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 (5.3a.10) 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`, `qmail` or `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** user interface. The full user/admin interface (mail and web) is internationalized. Translations are gathered in a standard PO file.
– **Template based** user interface. Every web page and service message can be customized via **TT2** template format.
– **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 in **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 (21.3.8, page 214). The sending process configuration (as well as most other list operations) is defined using an **authorization scenario**. Any listmaster can define new authorization scenarios in order to complement the 20 predefined configurations included in the distribution.

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

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 `/usr/local/sympa-os/etc/sympa.conf` global configuration file (listmaster can also be defined for a particular virtual host). Privileged operations include the usual `ADD`, `DELETE` or `REVIEW` commands, which can be authenticated via a one-time password or an S/MIME signature.
- **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 9.1, page 93)
- **RDBMS** : the internal subscriber and administrative data structure can be stored in a database or, for compatibility with versions 1.x, in text files for subscriber data. 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 RDBMS (MySQL, SQLite, PostgreSQL, Oracle, Sybase). (See ref sec-rdbms, page 79)
- **Virtual hosting** : a single *Sympa* installation can provide multiple virtual robots with both email and web interface customization (See 15, page 153).
- **LDAP-based mailing lists** : 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 21.2.6, page 207)
- **LDAP authentication** : 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 130)
- **Antivirus scanner** : *Sympa* extracts attachements from incoming messages and run a virus scanner on them. Curently working with McAfee/uvscan, Fsecure/fsav, Sophos, AVP, Trend Micro/VirusWall and Clam Antivirus. (See ref antivirus, page 245)
- 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”.
- channel RSS.

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 authorization 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 hosting, LDAP authentication
- Aug 2003 Automatic bounces management
- Sep 2003 CAS-base and Shibboleth-based authentication
- Dec 2003 Sympa SOAP server
- Aug 2004 Changed for TT2 template format and PO catalogue format
- 2005 Changed HTML to XHTML + CSS, RSS, List families, ...

[2]http ://www.sympa.org/release.html
[3]http ://www.sympa.org/sympa/in-the-future.html

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].

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 :

– Virginie Paitrault,Université de Rennes 2, who wrote the excellent online user documentation.
– John-Paul Robinson, University of Alabama at Birmingham, who added to email verification procedure to the Shibboleth support.
– Gwenaelle Bouteille who joined the development team for a few months and produced a great job for various feature introduced in V5 (familly, RSS, shared document moderation, ...).
– 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 his persevering in tuning *Sympa* for 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 their 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

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

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.

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/sympa/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 host or for the whole site.

– `/usr/local/sympa-os`
  The root directory of *Sympa*. You will find almost everything related to *Sympa* under this directory, except logs and main configuration files.
– `/usr/local/sympa-os/bin`
  This directory contains the binaries, including the CGI. It also contains the default authorization scenarios, templates and configuration files as in the distribution. `/usr/local/sympa-os/bin` may be completely overwritten by the `make install` so you must not customize templates and authorization scenarios under `/usr/local/sympa-os/bin`.
– `/usr/local/sympa-os/bin/etc`
  Here *Sympa* stores the default versions of what it will otherwise find in `/usr/local/sympa-os/etc` (task models, authorization scenarios, templates and configuration files, recognized S/Mime certificates, families).
– `/usr/local/sympa-os/etc`
  This is your site's configuration directory. Consult `/usr/local/sympa-os/bin/etc` when drawing up your own.
– `/usr/local/sympa-os/etc/create_list_templates/`
  List templates (suggested at list creation time).
– `/usr/local/sympa-os/etc/scenari/`
  This directory will contain your authorization scenarios. If you don't know what the hell an authorization scenario is, refer to 14,page 143. Those authorization scenarios are default scenarios but you may look at `/usr/local/sympa-os/etc/my.domain.org/scenari/` for default scenarios of

my.domain.orgvirtual host and `/usr/local/sympa-os/expl/mylist/scenari` for scenarios specific to a particular list
- `/usr/local/sympa-os/etc/data_sources/`
  This directory will contain your .incl files (see 18.7, page 172). For the moment it only deals with files requiered by paragraphs `owner_include` and `editor_include` in the config file.
- `/usr/local/sympa-os/etc/list_task_models/`
  This directory will store your own list task models (see 17.8, page 164).
- `/usr/local/sympa-os/etc/global_task_models/`
  Contains global task models of yours (see 17.8, page 164).
- `/usr/local/sympa-os/etc/web_tt2/` (used to be `/usr/local/sympa-os/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 `/usr/local/sympa-os/expl/mylist/web_tt2/` or in `/usr/local/sympa-os/etc/my.domain.org/web_tt2/`
- `/usr/local/sympa-os/etc/mail_tt2/` (used to be `/usr/local/sympa-os/etc/templates/`)
  Some of the mail robot's replies are defined by templates (`welcome.tt2` for SUBSCRIBE). You can overload these template files in the individual list directories or for each virtual host, but these are the defaults.
- `/usr/local/sympa-os/etc/families/`
  Contains family directories of yours (see 19, page 179). Families directories can also be created in `/usr/local/sympa-os/etc/my.domain.org/families/`
- `/usr/local/sympa-os/etc/my.domain.org`
  The directory to define the virtual host my.domain.orgdedicated to managment of all lists of this domain (list description of my.domain.orgare stored in `/usr/local/sympa-os/expl/my.domain.org`). Those directories for virtual hosts have the same structure as `/usr/local/sympa-os/etc` which is the configuration dir of the default robot.
- `/usr/local/sympa-os/expl`
  Sympa's working directory.
- `/usr/local/sympa-os/expl/mylist`
  The list directory (refer to 18, page 169). Lists stored in this directory belong to the default robot as defined in sympa.conf file, but a list can be stored in `/usr/local/sympa-os/expl/my.domain.org/mylist` directory and it is managed by my.domain.orgvirtual host.
- `/usr/local/sympa-os/expl/X509-user-certs`
  The directory where Sympa stores all user's certificates
- `/usr/local/sympa-os/locale`
  Internationalization directory. It contains message catalogues in the GNU .po format.
- `/usr/local/sympa-os/spool`
  Sympa uses 9 different spools (see 2.4, page 24).
- `/usr/local/sympa-os/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 these 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 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 regularly scans the `task/` spool.
- `sympa_soap_server.fcgi`
  The server will process SOAP (web services) request. This server requires FastCGI ; it should be referenced from within your HTTPS config.
- `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

- `/usr/local/sympa-os/etc/sympa.conf`
  The main configuration file. See 7, page 49.
- `/usr/local/sympa-os/etc/wwsympa.conf`
  WWSympa configuration file. See 1.2, page 17.
- `edit_list.conf`
  Defines which parameters/files are editable by owners. See 19.4.4, page 184.
- `topics.conf`
  Contains the declarations of your site's topics (classification in *WWSympa*), along with their titles. A sample is provided in the `sample/` directory of the sympa distribution. See 17.5, page 163.
- `auth.conf`
  Defines authentication backend organisation (LDAP-based authentication, CAS-based authentication and sympa internal)

- `robot.conf`
  It is a subset of `sympa.conf` defining a Virtual host (one per Virtual host).
- `nrcpt_by_domain`
  This file is used to limit the number of recipients per SMTP session. Some ISPs trying to block spams rejects sessions with too many recipients. In such case you can set the 7.4.8 robot.conf parameter to a lower value but this will affect all smtp session with any remote MTA. This file is used to limit the number of receipient for some particular domains. the file must contain a list of domain followed by the maximum number of recipient per SMTP session. Example :
- `data_structure.version`
  This file is automatically created and maintained by Sympa itself. It contains the current version of your Sympa service and is used to detect upgrades and trigger maintenance procedures such as database structure changes.

```
yohaa.com 3
oal.com 5
```

2.4 Spools

See 7.6, page 62 for spool definition in `sympa.conf`.

- `/usr/local/sympa-os/spool/auth/`
  For storing messages until they have been confirmed.
- `/usr/local/sympa-os/spool/bounce/`
  For storing incoming bouncing messages.
- `/usr/local/sympa-os/spool/digest/`
  For storing lists' digests before they are sent.
- `/usr/local/sympa-os/spool/mod/`
  For storing unmoderated messages.
- `/usr/local/sympa-os/spool/msg/`
  For storing incoming messages (including commands).
- `/usr/local/sympa-os/spool/msg/bad/`
  Sympa stores rejected messages in this directory
- `/usr/local/sympa-os/spool/distribute/`
  For storing message ready for distribution. This spool is used only if the installation run 2 sympa.pl daemon, one for commands, one for messages.
- `/usr/local/sympa-os/spool/distribute/bad/`
  Sympa stores rejected messages in this directory
- `/usr/local/sympa-os/spool/task/`
  For storing all created tasks.
- `/usr/local/sympa-os/spool/outgoing/`
  `sympa.pl` dumps messages in this spool to await archiving by `archived.pl`.
- `/usr/local/sympa-os/spool/topic/`
  For storing topic information files.

2.5 Roles and privileges

You can assign roles to users (via their email addresses) at different level in Sympa ; privileges are associated (or can be associated) to these roles. We list these roles below (from the most powerful to the less), along with the relevent privileges.

2.5.1 (Super) listmasters

These are the persons administrating the service, defined in the `sympa.conf` file. They inherit the listmaster role in virtual hosts and are the default set of listmasters for virtual hosts.

2.5.2 (Robot) listmasters

You can define a different set of listmasters at a virtual host level (in the `robot.conf` file). They are responsible for moderating mailing lists creation (if list creation is configured this way), editing default templates, providing help to list owners and moderators. Users defined as listmasters get a privileged access to Sympa web interface. Listmasters also inherit the privileges of list owners (for any list defined in the virtual host), but not the moderator privileges.

2.5.3 Privileged list owners

The first defined privileged owner is the person who requested the list creation. Later it can be changed or extended. They inherit (basic) owners privileges and are also responsible for managing the list owners and editors themselves (via the web interface). With Sympa'd default behavior, privileged owners can edit more list parameters than (basic) owners can do ; but this can be customized via the `edit-list.conf` file.

2.5.4 (Basic) list owners

They are responsible for managing the members of the list, editing the list configuration and templates. Owners (and privileged owners) are defined in the list config file.

2.5.5 Moderators (also called Editors)

Moderators are responsible for the messages distributed in the mailing list (as opposed to owners who look after list members). Moderators are active if the list has been setup as a moderated mailing list. If no moderator is defined for the list, then list owners will inherit the moderator role.

2.5.6 Subscribers (or list members)

Subscribers are the persons who are member of a mailing list ; they either subscribed, or got added directly by the listmaster or via a datasource (LDAP, SQL, another list,...). These subscribers receive messages posted in the list (unless they have set the `nomail` option) and have special privileges to post in the mailing list (unless it is a newsletter). Most privileges a subscriber may have is not hardcoded in Sympa but expressed via the so-called authorization scenarios (see 14, page 143).

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 ://www.sympa.org`. All important resources are referenced there :

- sources
- `RELEASE_NOTES`
- .rpm and .deb packages for Linux
- user mailing list (see 1.6, page 20)
- 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 RDBMS (Oracle, MySQL, SQLite, Sybase or PostgreSQL) and creating *Sympa*'s Database. This is required for using the web interface for *Sympa*. Please refers to *Sympa* and its database section (8, page 79).
– installation of libxml 2[1], required by the LibXML perl module.
– installation of CPAN CPAN (Comprehensive PERL Archive Network)[2] 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) ;

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
– NetBSD
– Digital UNIX 4.x
– Solaris 2.5 and 2.6
– AIX 4.x
– HP-UX 10.20

Anyone willing to port it to NT ? ;-)

Finally, most UNIX systems are now supplied with an ANSI C compiler ; if this is not the case, you can install the `gcc` compiler, which you will find on the nearest GNU site, for example in France[3].

To complete the installation, you should make sure that you have a sufficiently recent release of the `sendmail` MTA, i.e. release 8.9.x[4] or a more recent release. You may also use `postfix` or `qmail`.

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

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

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[5]. 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[6], 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` 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[7] ; 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. At `make` time, Sympa will prompt you for missing Perl modules and will attempt to install the missing ones automatically ; this operation requires root privileges.

Because Sympa features evolve from one relaease to another, the following list of modules might not be up to date :

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

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

– `Template-Toolkit`

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`

If you want to Download Zip files of list's Archives, you'll need to install perl Module for Archive Management :

– `Archive : :Zip`

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 `/usr/local/sympa-os`.

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/` directory : the archive will create a directory named `sympa-5.3a.10/` 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 ; a `locale/` 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-5.3a.10.tar.gz | tar xf -
```

Now you can run the installation process :

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

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

- – – `prefix=PREFIX`, the *Sympa* homedirectory (default /home/sympa/)
- – – `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`, system admin executables in DIR (default /home/sympa/bin)
- – – `with-libexecdir=DIR`, program executables in DIR (default /home/sympa/bin)
- – – `with-cgidir=DIR`, CGI programs in DIR (default /home/sympa/bin)
- – – `with-iconsdir=DIR`, web interface icons in DIR (default /home/httpd/icons)
- – – `with-datadir=DIR`, default configuration data in DIR (default /home/sympa/bin/etc)
- – – `with-confdir=DIR`, Sympa main configuration files in DIR (default /etc)
  `sympa.conf` and `wwsympa.conf` will be installed there.
- – – `with-expldir=DIR`, modifiable data in DIR (default /home/sympa/expl/)
- – – `with-libdir=DIR`, code libraries in DIR (default /home/sympa/bin/)

- `--- with-mandir=DIR`, man documentation in DIR (default /usr/local/man/)
- `--- with-docdir=DIR`, man files in DIR (default /home/sympa/doc/)
- `--- with-initdir=DIR`, install System V init script in DIR (default /etc/rc.d/init.d)
- `--- with-lockdir=DIR`, create lock files in DIR (default /var/lock/subsys)
- `--- with-piddir=DIR`, create .pid files in DIR (default /home/sympa/)
- `--- with-etcdir=DIR`, Config directories populated by the user are in DIR (default /home/sympa/etc)
- `--- with-localedir=DIR`, create language files in DIR (default /home/sympa/locale)
- `--- with-scriptdir=DIR`, create script files in DIR (default /home/sympa/script)
- `--- with-sampledir=DIR`, create sample files in DIR (default /home/sympa/sample)
- `--- with-spooldir=DIR`, create directory in DIR (default /home/sympa/spool)
- `--- with-perl=FULLPATH`, set full path to Perl interpreter (default /usr/bin/perl)
- `--- with-openssl=FULLPATH`, set path to OpenSSL (default /usr/local/ssl/bin/openssl)
- `--- with-user=LOGI`, set sympa user name (default sympa)
  Sympa daemons are running under this UID.
- `--- with-group=LOGIN`, set sympa group name (default sympa)
  Sympa daemons are running under this UID.
- `--- with-sendmail_aliases=ALIASFILE`, set aliases file to be used by Sympa (default /etc/mail/sympa_aliases). Set to 'none' to disable alias management (You can overright this value at runtime giving its value in `sympa.conf`)

- `--- 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`, set path to sendmail newaliases command (default /usr/bin/newaliases)
- `--- with-newaliases_arg=ARGS`, set arguments to newaliases command (default NONE)
  This is used by the `postfix_manager.pl` script :
- `--- with-postmap=FULLPATH`, set path to postfix postmap command (default /usr/sbin/postmap)
- `--- with-postmap_arg=ARGS`, set arguments to postfix postmap command (default NONE)
- `--- enable-secure`, install wwsympa to be run in a secure mode, without suidperl (default disabled)

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

`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 7.1.9, page 51)

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 set (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 21.

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

See Robot aliases , 6.1, page 45)

3.5 Logs

Sympa keeps a trace of each of its procedures in its log file. However, this requires configuration of the `syslogd` daemon. By default *Sympa* will 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` configuration file (`/etc/syslog.conf`). For example :

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

Then reload `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 (7.3.3, page 56). You can test log feature by using `testlogs.pl`.

Chapitre 4

Running *Sympa*

4.1 sympa.pl

`sympa.pl` is the main daemon ; it processes mail commands and is in charge of messages distribution.

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

- `- - - debug` — `-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.
- `service` *process_command* — *process_message* — *process_creation*
  Sets *Sympa* daemon in way it process only message distribution (process_message) or in way it process only command (process_command) or to process list creation requests (process_creation)
- `- - - config` *`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` — `-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`* — `-l` *`catalog`*
  Set this option to use a language catalog for *Sympa*. The corresponding catalog file must be located in `~sympa/locale` directory.
- `- - - keepcopy` *`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
```

– `- - create_list - - robot` *`robotname`* `- - input_file` *`/path/to/list_file.xml`*
  Create the list described by the xml file, see 19.3, page 183.
– `- - close_list` *`listname@robot`*
  Close the list (changing its status to closed), remove aliases (if sendmail_aliases parameter was set) and remove subscribers from DB (a dump is created in the list directory to allow restoring the list). See **??**, page **??** when you are in a family context.
– `- - dump` *`listname | ALL`*
  Dumps subscribers of a list or all lists. Subscribers are dumped in `subscribers.db.dump`.
– `- - import` *`listname`*
  Import subscribers in the *listname* list. Data are read from STDIN.
– `- - lowercase`
  Lowercases e-mail addresses in database.
– `- - help` — `-h`
  Print usage of sympa.pl.
– `- - make_alias_file`
  Create an aliases file in /tmp/ with all list aliases (only list which status is 'open'). It uses the list_aliases.tt2 template.
– `- - version` — `-v`
  Print current version of *Sympa*.
– `- - instanciate_family` *`familyname robotname`* `- - input_file` *`/path/to/family_file.xml`*
  Instantiate the family *familyname*. See 20, page 187.
– `- - close_family` *`familyname`* `- - robot` *`robotname`*
  Close the *familyname* family. See 20.2.4, page 193.
– `- - add_list` *`familyname`* `- - robot` *`robotname`* `- - input_file` *`/path/to/list_file.xml`*
  Add the list described in the XML file to the *familyname* family. See 20.2.5, page 193.
– `- - modify_list` *`familyname`* `- - robot` *`robotname`* `- - input_file` *`/path/to/list_file.xml`*
  Modify the existing family list, with description contained in the XML file. See 20.2.7, page 194.
– `- - sync_include` *`listaddress`*
  Trigger an update of list members, usefull if the list uses external data sources.
– `- - upgrade - - from=X - -to=Y`
  Runs Sympa maintenance script to upgrate from version X to version Y
– `- - reload_list_config - -list=mylist@dom`
  Recreates all `configbin` files. You should run this command if you edit authorization scenarios. The list parameter is optional.

4.2 INIT script

The `make install` step should have installed a sysV init script in your `/etc/rc.d/init.d/` directory (you can change this at `configure` time with the `--with-initdir` option). You should edit your runlevels to make sure *Sympa* starts after Apache and MySQL. Note that MySQL should also start before Apache because of `wwsympa.fcgi`.

This script starts these deamons : sympa.pl, task_manager.pl, archived.pl and bounced.pl.

4.3 Stopping *Sympa* and signals

`kill -TERM`

When this signal is sent to sympa.pl (`kill -TERM`), the daemon is stopped ending message distribution in progress and this can be long (for big lists). If `kill -TERM` is used, sympa.pl will stop immediatly whatever a distribution message is in progress. In this case, when sympa.pl restart, message will distributed many times.

`kill -HUP`

When this signal is sent to sympa.pl (`kill -HUP`), it switchs of the `--mail` logging option and continues current task.

Chapitre 5

Upgrading Sympa

Sympa upgrade is a relatively riskless operations, mainly because the install process preserves your customizations (templates, configuration, authorization scenarios,...) and also because Sympa automates a few things (DB update, CPAN modules installation).

5.1 Incompatible changes

New features, changes and bug fixes are summarized in the `NEWS` file, part of the tar.gz (the `Changelog` file is a complete log file of CVS changes).

Sympa is a long-term project, so some major changes may need some extra work. The following list is well kown changes that require some attention :
- version 5.1 (August 2005) use XHTML and CSS in web templates
- version 4.2b3 (August 2004) introduce TT2 template format
- version 4.0a5 (September 2003) change auth.conf (no default anymore so you may have the create this file)
- version 3.3.6b2 (May 2002) the list parameter user_data_source as a new value include2 which is the recommended value for any list.

The file `NEWS` list all changes and of course, all changes that may require some attention from the installer. As mentionned at the beginning of this file, incompatible changes are preceded by `'*****'`. While running the `make install` Sympa will detect the previously installed version and will prompt you with incompatible changes between both versions of the software. You can interrupt the install process at that stage if you are too frightened. Output of the `make install` :

```
You are upgrading from Sympa 4.2
You should read CAREFULLY the changes listed below ; they might be incompatible changes :
<RETURN>
```

```
*****   require new perlmodule XML-LibXML

*****   You should update your DB structure (automatically performed by Sympa wi
*****   CREATE TABLE admin_table (
*****   list_admin              varchar(50) NOT NULL,
*****   user_admin              varchar(100) NOT NULL,
*****   role_admin              enum('listmaster','owner','editor') NOT NULL,
*****   date_admin              datetime NOT NULL,
*****   update_admin            datetime,
*****   reception_admin         varchar(20),
*****   comment_admin           varchar(150),
*****   subscribed_admin        enum('0','1'),
*****   included_admin          enum('0','1'),
*****   include_sources_admin   varchar(50),
*****   info_admin              varchar(150),
*****   profile_admin           enum('privileged','normal'),
*****   PRIMARY KEY (list_admin, user_admin,role_admin),
*****   INDEX (list_admin, user_admin,role_admin)
*****   );

*****   Extend the generic_sso feature ; Sympa is now able to retrieve the user e
<RETURN>
```

5.2 CPAN modules update

Required and optional perl modules (CPAN) installation is automatically handled at the `make` time. You are asked before each module is installed. For optional modules, associated features are listed.

Output of the `make` command :

```
Checking for REQUIRED modules:
------------------------------------------
perl module          from CPAN        STATUS
-----------          ---------        ------
Archive::Zip         Archive-Zip     OK (1.09   >= 1.05)
CGI                  CGI             OK (2.89   >= 2.52)
DB_File              DB_FILE         OK (1.806  >= 1.75)
Digest::MD5          Digest-MD5      OK (2.20   >= 2.00)
FCGI                 FCGI            OK (0.67   >= 0.67)
File::Spec           File-Spec       OK (0.83   >= 0.8)
IO::Scalar           IO-stringy      OK (2.104  >= 1.0)
LWP                  libwww-perl     OK (5.65   >= 1.0)
Locale::TextDomain   libintl-perl    OK (1.10   >= 1.0)
MHonArc::UTF8        MHonArc         version is too old ( < 2.4.6).
>>>>>>> You must update "MHonArc" to version "" <<<<<<.
Setting FTP Passive mode
```

```
Description:
Install module MHonArc::UTF8 ? n
MIME::Base64          MIME-Base64    OK (3.05   >= 3.03)
MIME::Tools           MIME-tools     OK (5.411  >= 5.209)
Mail::Internet        MailTools      OK (1.60   >= 1.51)
Regexp::Common        Regexp-Common  OK (2.113  >= 1.0)
Template              Template-ToolkitOK (2.13   >= 1.0)
XML::LibXML           XML-LibXML     OK (1.58   >= 1.0)

Checking for OPTIONAL modules:
------------------------------------------
perl module           from CPAN       STATUS
-----------           ---------       ------
Bundle::LWP           LWP            OK (1.09   >= 1.09)
Constant subroutine CGI::XHTML_DTD redefined at /usr/lib/perl5/5.8.0/constant.pm line 108,
CGI::Fast             CGI            CGI::Fast doesn't return 1 (check it).
Crypt::CipherSaber    CipherSaber    OK (0.61   >= 0.50)
DBD::Oracle           DBD-Oracle     was not found on this system.
Description: Oracle database driver, required if you connect to a Oracle database.
Install module DBD::Oracle ?
```

5.3 Database structure update

Whatever RDBMS you are using (mysql, SQLite, Pg, Sybase or Oracle) Sympa will check every database tables and fields. If one is missing `sympa.pl` will not start. If you are using mysql Sympa will also check field types and will try to change them (or create them) automatically ; assuming that the DB user configured has sufficient privileges. If You are not using Mysql or if the DB user configured in `sympa.conf` does have sufficient privileges, then you should change the database structure yourself, as mentionned in the `NEWS` file.

Output of Sympa logs :

```
Table admin_table created in database sympa
Field 'comment_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting t
Field comment_admin added to table admin_table
Field 'date_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to a
Field date_admin added to table admin_table
Field 'include_sources_admin' (table 'admin_table' ; database 'sympa') was NOT found. Atte
Field include_sources_admin added to table admin_table
Field 'included_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting
Field included_admin added to table admin_table
Field 'info_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to a
Field info_admin added to table admin_table
Field 'list_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to a
Field list_admin added to table admin_table
Field 'profile_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting t
Field profile_admin added to table admin_table
Field 'reception_admin' (table 'admin_table' ; database 'sympa') was NOT found.
Field reception_admin added to table admin_table
Field 'role_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attem
Field role_admin added to table admin_table
Field 'subscribed_admin' (table 'admin_table' ; database 'sympa') was NOT found.
Field subscribed_admin added to table admin_table
Field 'update_admin' (table 'admin_table' ; database 'sympa') was NOT found. Att
Field update_admin added to table admin_table
Field 'user_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attem
Setting list_admin,user_admin,role_admin fields as PRIMARY
Field user_admin added to table admin_table
```

You might need, for some reason, to make Sympa run the migration procedure from version *X* to version *Y*. This procedure is run automatically by `sympa.pl --upgrade` when it detects that `/data_structure.version` is older than the current version, but you can also run trigger this procedure yourself :

```
sympa.pl --upgrade --from=4.1 --to=5.2
```

5.4 Preserving your customizations

Sympa comes with default configuration files (templates, scenarios,...) that will be installed in the `/usr/local/sympa-os/bin` directory. If you need to customize some of them, you should copy the file first in a safe place, ie in the `/usr/local/sympa-os/etc` directory. If you do so, the Sympa upgrade process will preserve your site customizations.

5.5 Running 2 Sympa versions on a single server

This can be very convenient to have a stable version of Sympa and a fresh version for test purpose, both running on the same server.

Both sympa instances must be completely partitioned, unless you want the make production mailing lists visible through the test service.

The biggest part of the partitioning is achieved while running the `./configure`. Here is a sample call to `./configure` on the test server side :

```
./configure --prefix=/home/sympa-dev \
            --with-confdir=/home/sympa-dev/etc \
            --with-mandir=/home/sympa-dev/man \
            --with-initdir=/home/sympa-dev/init \
    --with-piddir=/home/sympa-dev/pid
```

```
                  --with-lockdir=/home/sympa-dev/lock \
                  --with-sendmail_aliases=/home/sympa-dev/etc/sympa_aliases
```

You can also customize more parameters via the `/home/sympa-dev/etc/sympa.conf` file.

If you wish to share the same lists in both Sympa instances, then some parameters should have the same value : `home`, `db_name`, `arc_path`

5.6 Moving to another server

If you're upgrading and moving to another server at the same time, we recommend you first to stop the operational service, move your data and then upgrade Sympa on the new server. This will guarantee that Sympa upgrade procedures have been applied on the data.

The migration process requires that you move the following data from the old server to the new one :
- the user database. If using mysql you can probably just stop `mysqld` and copy the `/var/lib/mysql/sympa/` directory to the new server.
- the `/usr/local/sympa-os/expl` directory that contains list config
- the directory that contains the spools
- the directory and `/usr/local/sympa-os/etc/sympa.conf` and `wwsympa.conf`. Sympa new installation create a file `/usr/local/sympa-os/etc/sympa.conf` (see 7) and initialize randomly the cookie parameter. Changing this parameter will break all passwords. When upgrading Sympa on a new server take care that you start with the same value of this parameter, otherwise you will have troubles !
- the web archives

In some case, you may want to install the new version and run it for a few days before switching the existing service to the new Sympa server. In this case perform a new installation with an empty database and play with it. When you decide to move the existing service to the new server :

1. stop all sympa processus on both servers,
2. transfert the database
3. edit the `/data_structure.version` on the new server ; change the version value to reflect the old number
4. start `sympa.pl --upgrade`, it will upgrade the database structure according the hop you do.

Chapitre 6

Mail aliases

Mail aliases are required in Sympa for `sympa.pl` to receive mail commands and list messages. Management of these aliases management will depend on the MTA (`sendmail`, `qmail`, `postfix`, `exim`) you're using, where you store aliases and whether you are managing virtual domains or not.

6.1 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 (commands such as subscribe or unsubscribe) is received by the SMTP server. The SMTP server, on reception of this message, runs the `queue` program (supplied in this package) to store the message in a spool.
– the `sympa.pl` daemon, set in motion at system startup, scans this spool. As soon as it detects a new message, it processes it and performs the requested action (distribution or processing of a command).

To separate the processing of commands (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` alias file (often `/etc/aliases`) :

sympa : "— /usr/local/sympa-os/bin/queue sympa@my.domain.org"
listmaster : "— /usr/local/sympa-os/bin/queue listmaster@my.domain.org"
bounce+* : "— /usr/local/sympa-os/bin/bouncequeue

sympa@my.domain.org"
abuse-feedback-report : "— /usr/local/sympa-os/bin/bouncequeue
sympa@my.domain.org"
sympa-request : postmaster
sympa-owner : postmaster

Note : if you run *Sympa* virtual hosts, you will need one `sympa` alias entry per virtual host (see virtual hosts section, 15, page 153).

`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 where VERP (variable envelope return path) was actived. It is useful if `welcome_return_path unique` or `remind_return_path unique` or the `verp_rate` parameter is no null for at least one list.

The alias abuse-feedback-report is used for processing automatically feedback that respect ARF format (Abuse Report Feedback) which is a draft to specify how end user can complain about spam. It is mainly used by AOL.

Don't forget to run `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
```

6.2 List aliases

For each new list, it is necessary to create up to six mail aliases (at least three). If you managed to setup the alias manager (see next section) then *Sympa* will install automatically the following aliases for you.

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

```
mylist :               "|/usr/local/sympa-os/bin/queue mylist@my.domain.org"
mylist-request :       "|/usr/local/sympa-os/bin/queue mylist-request@my.domain.org"
mylist-editor :        "|/usr/local/sympa-os/bin/queue mylist-editor@my.domain.org"
mylist-owner :         "|/usr/local/sympa-os/bin/bouncequeue mylist@my.domain.org
mylist-subscribe :     "|/usr/local/sympa-os/bin/queue mylist-subscribe@my.domain.org"
mylist-unsubscribe :   "|/usr/local/sympa-os/bin/queue mylist-unsubscribe@my.domain.org"
```

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 `/usr/local/sympa-os/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 `/usr/local/sympa-os/expl/mylist/config`. This address definition is not compulsory.

The address `mylist-owner` is the address receiving non-delivery reports (note that the -owner suffix can be customized, see 7.8.4, page 67). The `bouncequeue` program stores these messages in the `queuebounce` directory. *WWSympa* ((see 1.2, page 17) 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 !

6.3 Alias manager

The `alias_manager.pl` script does aliases management. It is run by *WWSympa* and will install aliases for a new list and delete aliases for closed lists.

The script expects the following arguments :

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

Example : `/usr/local/sympa-os/bin/alias_manager.pl add mylistcru.fr`

`/usr/local/sympa-os/bin/alias_manager.pl` works on the alias file as defined in `sympa.conf`) by the `sendmail_aliases` variable (default is `/etc/mail/sympa_aliases`). You must refer to this aliases file in your `sendmail.mc` (if using sendmail) :

```
define(`ALIAS_FILE', `/etc/aliases,/etc/mail/sympa_aliases')dnl
```

Note that `sendmail` has requirements regarding the ownership and rights on both `sympa_aliases` and `sympa_aliases.db` files (the later being created by sendmail via the `newaliases` command). Anyhow these two files should be located in a directory, every path component of which being owned by and writable only by the root user.

`/usr/local/sympa-os/bin/alias_manager.pl` runs a `newaliases` command (via `aliaswrapper`), after any changes to aliases file.

If you manage virtual domains with your mail server, then you might want to change the form of aliases used by the alias_manager. You can customize the `list_aliases` template that is parsed to generate list aliases (see18.8.8, page 175).

Note that you don't need alias management if you use MTA functionalities such as Postfix' `virtual_transport`. You can then disable alias management in *Sympa* by positioning the `sendmail_aliases` parameter to `none`.

A L. Marcotte has written a version of `ldap_alias_manager.pl` that is LDAP enabled. This script is distributed with Sympa distribution ; it needs to be customized with your LDAP parameters.

6.4 Virtual domains

When using virtual domains with `sendmail` or `postfix`, you can't refer to `mylist@my.domain.org` on the right-hand side of an `/etc/aliases` entry. You need to define an additional entry in a virtual table. You can also add a unique entry, with a regular expression, for your domain.

With Postfix, you should edit the `/etc/postfix/virtual.regexp` file as follows :

/^(.*)@my.domain.org$/ my.domain.org-$1

Entries in the 'aliases' file will look like this :

my.domain.org-sympa : "—/usr/local/sympa-os/bin/queue sympa@my.domain.org" ..... my.domain.org-listA : "—/usr/local/sympa-os/bin/queue listA@my.domain.org"

With Sendmail, add the following entry to `/etc/mail/virtusertable` file :

@my.domain.org my.domain.org-%1%3

Chapitre 7

sympa.conf parameters

The `/usr/local/sympa-os/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 `/usr/local/sympa-os/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.

7.1 Site customization

7.1.1 `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 address. The old keyword `host` is still recognized but should not be used anymore.

Example: `domain cru.fr`

7.1.2 `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`

7.1.3 `listmaster`

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

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

7.1.4 `listmaster_email`

(Default value: `listmaster`)

Username (the part of the address preceding the `@` sign) used in the listmaster email. This parameter is useful if you want to run more than one sympa on the same host (a sympa test for example).

If you change the default value, you must modify the sympa aliases too.

For example, if you put :

> listmaster listmaster-test

you must modify the sympa aliases like this :

> listmaster-test : ”— /home/sympa/bin/queue listmaster@my.domain.org”

See 6.1,page 45 for all aliases.

7.1.5 `wwsympa_url`

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

This is the root URL of *WWSympa*.

Example: `wwsympa_url https ://my.server/sympa`

7.1.6 `soap_url`

This is the root URL of Sympa's SOAP server. Sympa's WSDL document refer to this URL in its `service` section.

Example: `soap_url http ://my.server/sympasoap`

7.1.7 `spam_protection`

spam_protection (Default value: `javascript`)

There is a need to protection Sympa web site against spambot which collect email adresse in public web site. Various method are availble into Sympa and you can choose it with `spam_protection` and `web_archive_spam_protection` parameters. Possible value are :

- javascript : the adresse is hidden using a javascript. User who enable javascript can see a nice mailto adresses where others have nothing.
- at : the @ char is replaced by the string " AT ".
- none : no protection against spammer.

7.1.8 `web_archive_spam_protection`

(Default value: `cookie`)

Idem `spam_protection` but restricted to web archive. A additional value is available : cookie which mean that users must submit a small form in order to receive a cookie before browsing archives. This block all robot, even google and co.

7.1.9 `color_0`, `color_1` .. `color_15`

They are the color definition for web interface. These parameters can be overwritten in each virtual host definition. The color are used in the CSS file and unfortunitly they are also in use in some web templates. The sympa admin interface show every colors in use.

7.1.10 `dark_color`, `light_color`, `text_color`, `bg_color`, `error_color`, `selected_color`, `shaded_color`

Deprecated. They are the color definition for previous web interface. These parameters are unused in 5.1 and higher version but still available.style.css, print.css, print-preview.css and fullPage.css

7.1.11 `logo_html_definition`

This parameter allow you to insert in the left top page corner oa piece of html code, usually to insert la logo in the page. This is a very basic but easy customization. Example: `logo_html_definition <a href="http ://www.mycompagnie.com"><img style="float : left ; margin-top : 7px ; margin-left : 37px ;" src="http :/logos/mylogo.jpg" alt="my compagnie" /></a>`

7.1.12 `css_path`

Pre-parsed CSS files (let's say static css files) can be installed using Sympa server skins module. These CSS files are installed in a part of the web server that can be reached without using sympa web engine. In order to do this edit the robot.conf file and set the css_path parameter. Then retart the server and use skins module from the "admin sympa" page to install preparsed CSS file. The in order to replace dynamic CSS by these static files set the `css_url` parameter.

After an upgrade, `sympa.pl` automatically updates the static CSS files with the new installed css.tt2. Therefore it's not a good place to store customized CSS files.

7.1.13 `css_url`

By default, CSS files style.css, print.css, print-preview.css and fullPage.css are delivred by Sympa web interface itself using a sympa action named css. URL look like http ://foo.org/sympa/css/style.css . CSS file are made parsing a web_tt2 file named css.tt2. This allow dynamique definition of colors and in a near futur a complete definition of the skin, user preference skins etc.

In order to make sympa web interface faster, it is strongly recommended to install static css file somewhere in your web site. This way sympa will deliver only one page insteed of one page and four css page at each clic. This can be done using css_url parameter. The parameter must contain the URL of the directory where style.css, print.css, print-preview.css and fullPage.css are installed. You can make your own a sophisticated new skin editing these files. The server admin module include a CSS administration page that can help you to install static CSS.

7.1.14 `static_content_path`

Some content may be delivred by http server (apache) without any need to be controled or parserd by Sympa. They are stored in directory choosen with parameter static_content_dir. Current Sympa version store in this directory subscribers pictures. Later update will add style sheet, icons, ... The directory is created by Sympa.pl when started. This parameter can be defined also in robot.conf

7.1.15 `static_content_url`

Content stored in directory specified by parameter `static_content_url` must be served by http server under the URL specified by `static_content_url`. Check apache configuration in order to make this directory available. This parameter can be defined in robot.conf.

7.1.16 `pictures_feature`

(Default value: `off`) Example: `pictures_feature on`

Subscribers can upload their picture (from the subscriber option page) so reviewing subsribers shows a gallery. This parameter defines the default for corresponding list parameter but it does NOT allow to disable the feature globaly. If you want to disable the feature for your whole site, you need to customize the `edit-list.conf` file to disallow edition of the corresponding list parameter.

Pictures are stored in a directory specified by `static_content_path` parameter.

7.1.17 `pictures_max_size`

The maximum size of the uploaded picture file (bytes)

7.1.18 `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. To prevent a catastroph, sympa.pl refuse to start if the cookie parameter was changed.

Example: `cookie gh869jku5`

7.1.19 `create_list`

(Default value: `public_listmaster`)

`create_list` parameter is defined by an authorization scenario (see 14, page 143)

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

Example: `create_list intranet`

7.1.20 `automatic_list_feature`

(Default value: `off`) Example: `automatic_list_feature on`

If set to `on`, Sympa will enable automatic list creation through family instantiation (see 20.3, page 194.

7.1.21 `automatic_list_creation`

(Default value: `none`)

`automatic_list_creation` parameter is defined by an authorization scenario (see 14, page 143)

If `automatic_list_feature` is activated, this parameter (corresponding to an authorization scenario) defines who is allowed to use the automatic list creation feature.

7.1.22 `automatic_list_removal`

(Default value:) Example: `automatic_list_feature if_empty`

If set to `if_empty`, then Sympa will remove automatically created mailing lists just after their creartion, if they contain no list membe (see 20.3, page 194.

7.1.23 `global_remind`

(Default value: `listmaster`)

`global_remind` parameter is defined by an authorization scenario (see 14, page 143)

Defines who can run a `REMIND *` command.

7.2 Directories

7.2.1 `home`

(Default value: `/usr/local/sympa-os/expl`)

The directory whose subdirectories correspond to the different lists.

Example: `home /home/sympa/expl`

7.2.2 `etc`

(Default value: `/usr/local/sympa-os/etc`)

This is the local directory for configuration files (such as `edit_list.conf`. It contains 5 subdirectories : `scenari` for local authorization scenarios ; `mail_tt2` for the site's local mail templates and default list templates ; `web_tt2` 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`

7.3 System related

7.3.1 `syslog`

(Default value: `LOCAL1`)

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

Example: `syslog LOCAL2`

7.3.2 `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`

7.3.3 `log_socket_type`

(Default value: `unix`)

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

7.3.4 `pidfile`

(Default value: `/usr/local/sympa-os/etc/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`

7.3.5 `pidfile_creation`

(Default value: `/usr/local/sympa-os/etc/sympa-creation.pid`)

The file where the automatic list creation dedicated `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_creation /var/run/sympa-creation.pid`

7.3.6 `umask`

(Default value: `027`)

Default mask for file creation (see `umask`(2)). Note that it will be interpreted as an octual value.

Example: `umask 007`

7.4 Sending related

7.4.1 `distribution_mode`

(Default value: `single`) Use this parameter to determine if your installation nrun only one sympa.pl daemon that process both messages to distribute and commands (single) or if sympa.pl will fork to run two separate processus one dedicated to message distribution and one dedicated to commands and message pre-processing (fork). The second choice make a better priority processing for message distribution and faster command response, but it require a bit more computer ressources.

Example: `distribution_mode fork`

7.4.2 `maxsmtp`

(Default value: `20`)

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

Example: `maxsmtp 500`

7.4.3 `log_smtp`

(Default value: `off`)

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

Example: `log_smtp on`

7.4.4 `use_blacklist`

(Default value: `send,create_list`) Sympa provide a blacklist feature available for list editor and list owner. The `use_blacklist` parameter define which operation use the blacklist. Search in black list is mainly usefull for the `send` service (distribution of a message to the subscribers). You may use blacklist for more operation such as review,archive etc but be aware that thoses web services needs fast response and blacklist may require some ressources.

If you don't want blacklist at all, define `use_blacklist none` so the user interface to manage blacklist will disappear from the web interface.

7.4.5 `max_size`

(Default value: `5 Mb`)

Maximum size allowed for messages distributed by *Sympa*. This may be customized per virtual host or per list by setting the `max_size` robot or list parameter.

Example: `max_size 2097152`

7.4.6 `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`.

7.4.7 `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.

7.4.8 `nrcpt`

(Default value: `25`)

Maximum number of recipients per `sendmail` call. This grouping factor makes it possible for the (`sendmail`) MTA to optimize the number of SMTP sessions for message distribution. If needed, you can limit the number of receipient for a particular domain. Check nrcpt_by_domain configuration file. (see 2.3, page 24)

7.4.9 `avg`

(Default value: `10`)

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

7.4.10 `sendmail`

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

Absolute path to SMTP message transfer agent binary. Sympa expects this binary to be sendmail compatible (postfix, Qmail and Exim binaries all provide sendmail compatibility).

Example: `sendmail /usr/sbin/sendmail`

7.4.11 `sendmail_args`

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

Arguments passed to SMTP message transfer agent

7.4.12 `sendmail_aliases`

(Default value: `defined by makefile, sendmail_aliases | none`)

Path of the alias file that contain all lists related aliases. It is recommended to create a specific alias file so Sympa never overright the standard alias file but only a dedicated file.You must refer to this aliases file in your `sendmail.mc` :

Set this parameter to 'none' if you want to disable alias management in sympa (e.g. if you use `virtual_transport` with Postfix).

7.4.13 `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.

7.4.14 `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,Deli`

7.4.15 `anonymous_headers_fields`

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

This parameter defines the list of SMTP header fields that should be removed when a mailing list is setup in anonymous mode (see 21.4.3, page 218).

7.4.16 `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 host 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.

7.4.17 `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. It is also used to check list names at list creation time.

7.4.18 `urlize_min_size`

(Default value: `10240`)

This parameter is related to the `URLIZE` subscriber reception mode ; it defines the minimum size (in bytes) for MIME attachments to be urlized.

7.5 Quotas

7.5.1 `default_shared_quota`

The default disk quota (the unit is Kbytes) for lists' document repository.

7.5.2 `default_archive_quota`

The default disk quota (the unit is Kbytes) for lists' web archives.

7.6 Spool related

7.6.1 `spool`

(Default value: `/usr/local/sympa-os/spool`)

The parent directory which contains all the other spools.

7.6.2 `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: `/usr/local/sympa-os/spool/msg`

7.6.3 `queuedistribute`

(Default value: `/usr/local/sympa-os/spool/distribute`)

This parameter is optional and retained solely for backward compatibility.

7.6.4 `queuemod`

(Default value: `/usr/local/sympa-os/spool/moderation`)

This parameter is optional and retained solely for backward compatibility.

7.6.5 `queuedigest`

This parameter is optional and retained solely for backward compatibility.

7.6.6 `queueauth`

(Default value: `/usr/local/sympa-os/spool/auth`)

This parameter is optional and retained solely for backward compatibility.

7.6.7 `queueoutgoing`

(Default value: `/usr/local/sympa-os/spool/outgoing`)

This parameter is optional and retained solely for backward compatibility.

7.6.8 `queuetopic`

(Default value: `/usr/local/sympa-os/spool/topic`)

This parameter is optional and retained solely for backward compatibility.

7.6.9 `queuebounce`

(Default value: `/usr/local/sympa-os/spool/bounce`)

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

7.6.10 `queuetask`

(Default value: `/usr/local/sympa-os/spool/task`)

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

7.6.11 `queueautomatic`

(Default value: `none`)

The absolute path of the directory which contains the queue for automatic list creation, used both by the `familyqueue` program and the `sympa.pl` daemon. This parameter is mandatory when enabling automatic_list_creation.

Example: `/usr/local/sympa-os/spool/msg`

7.6.12 `tmpdir`

(Default value: `/usr/local/sympa-os/spool/tmp`)

Temporary directory used by OpenSSL and antiviruses.

7.6.13 `sleep`

(Default value: `5`)

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

7.6.14 `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`

7.6.15 `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.

7.6.16 `clean_delay_queueauth`

(Default value: `3`)

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

7.6.17 `clean_delay_queuesubscribe`

(Default value: `10`)

Expiration delay (in days) in the subscription requests queue. Beyond this deadline, requests not validated are deleted.

7.6.18 `clean_delay_queuetopic`

(Default value: `7`)

Delay for keeping message topic files (in days) in the topic queue. Beyond this deadline, files are deleted.

7.6.19 `clean_delay_queueautomatic`

(Default value: `10`)

Retention period (in days) for "bad" messages in automatic spool (as specified by `queueautomatic`). *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.

7.7 Internationalization related

7.7.1 `localedir`

(Default value: `/usr/local/sympa-os/locale`)

The location of multilingual catalog files. Must correspond to `~src/locale/Makefile`.

7.7.2 `supported_lang`

Example: `supported_lang fr,en_US,de,es`

This parameter lists all supported languages (comma separated) for the user interface. The default value will include all message catalogues but it can be narrowed by the listmaster.

7.7.3 `lang`

(Default value: `en_US`)

This is the default language for *Sympa*. The message catalog (.po, compiled as a .mo file) located in the corresponding `locale` directory will be used.

7.7.4 `web_recode_to`

(OBSOLETE)

All web pages are now encoded in utf-8.

Note : if you recode web pages to utf-8, you should also add the following tag to your `mhonarc-ressources.tt2` file :

```
<TextEncode>
utf-8; MHonArc::UTF8::to_utf8; MHonArc/UTF8.pm
</TextEncode>
```

7.7.5 `filesystem_encoding`

(Default value: `utf-8`)

Example: `filesystem_encoding iso-8859-1`

Sympa (and Perl) use utf-8 as the its internal encoding and also for the encoding of web pages. Because you might use a different character encoding on your filesystem, you need to declare it, so that Sympa is able to properly decode strings.

7.8 Bounce related

7.8.1 `verp_rate`

(Default value: `0%`)

See 24.1,page 242 for more information on VERP in Sympa.

When `verp_rate` is null VERP is not used ; if `verp_rate` is 100% VERP is alway in use.

VERP requires plussed aliases to be supported and the bounce+* alias to be installed.

7.8.2 `welcome_return_path`

(Default value: `owner`)

If set to string `unique`, Sympa enable VERP for welcome message and bounce processing will remove the subscription if a bounce is received for the welcome message. This prevent to add bad address in subscriber list.

7.8.3 `remind_return_path`

(Default value: `owner`)

Like `welcome_return_path`, but relates to the remind message.

7.8.4 `return_path_suffix`

(Default value: `-owner`)

This defines the suffix that is appended to the list name to build the return-path of messages sent to the lists. This is the address that will receive all non delivery reports (also called bounces).

7.8.5 `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.

7.8.6 `purge_orphan_bounces_task`

(Default value: `Monthly`)

This parameter tells what task will be used by `task_manager.pl` to perform bounces cleaning. This task delete bounces archives for unsubscribed users.

7.8.7 `eval_bouncers_task`

(Default value: `daily`)

The task eval_bouncers evaluate all bouncing users for all lists, and fill the field `bounce_score_suscriber` in table `suscriber_table` with a score. This score allow the auto-management of bouncing-users.

7.8.8 `process_bouncers_task`

(Default value: `monthly`)

The task process_bouncers execute configured actions on bouncing users, according to their Score. The association between score and actions has to be done in List configuration, This parameter define the frequency of execution for this task.

7.8.9 `minimum_bouncing_count`

(Default value: `10`)

This parameter is for the bounce-score evaluation : the bounce-score is a note that allows the auto-management of bouncing users. This score is evaluated with,in particular, the number of messages bounces received for the user. This parameter sets the minimum number of these messages to allow the bounce-score evaluation for a user.

7.8.10 `minimum_bouncing_period`

(Default value: `10`)

Determine the minimum bouncing period for a user to allow his bounce-score evaluation. Like previous parameter, if this value is too low, bounce-score will be 0.

7.8.11 `bounce_delay`

(Default value: `0`) Days

Another parameter for the bounce-score evaluation : This one represent the average time (days) for a bounce to come back to sympa-server after a post was send to a list. Usually bounces are arriving same day as the original message.

7.8.12 `default_bounce_level1_rate`

(Default value: `45`)

This is the default value for `bouncerslevel1 rate` entry (**??**, page **??**)

7.8.13 `default_bounce_level2_rate`

(Default value: `75`)

This is the default value for `bouncerslevel2 rate` entry (21.5.3, page 224)

7.8.14 `bounce_email_prefix`

(Default value: `bounce`)

The prefix string used to build variable envelope return path (VERP). In the context of VERP enabled, the local part of the address start with a constant string specified by this parameter. The email is used to collect bounce. Plussed aliases are used in order to introduce the variable part of the email that encode the subscriber address. This parameter is useful if you want to run more than one sympa on the same host (a sympa test for example).

If you change the default value, you must modify the sympa aliases too.

For example, if you set it as :

> bounce_email_prefix bounce-test

you must modify the sympa aliases like this :

> bounce-test+* : ”— /home/sympa/bin/queuebounce sympa@my.domain.org”

See 6.1,page 45 for all aliases.

7.8.15 `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.

7.8.16 `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.

7.8.17 `default_remind_task`

(Default value: `2month`)

This parameter defines the default `remind_task` list parameter.

7.9 Tuning

7.9.1 `cache_list_config`

`Format : none | binary_file` (Default value: `none`)

If this parameter is set to binary_file, then Sympa processes will maintain a binary version of the list config structure on disk (`config.bin` file). This file is bypassed whenever the `config` file changes on disk. Thanks to this method, the startup of Sympa processes is much faster because it saves the time for parse all config files. The drawback of this method is that the list config cache could live for a long time (not recreated when Sympa process restart) ; Sympa processes could still use authorization scenario rules that have changed on disk in the meanwhile.

You should use list config cache if you are managing a big amount of lists (1000+).

7.9.2 `sympa_priority`

(Default value: `1`)

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

Available since release 2.3.1.

7.9.3 `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

7.9.4 `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

7.9.5 `default_list_priority`

(Default value: `5`)

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

Available since release 2.3.1.

7.10 Database related

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

7.10.1 `update_db_field_types`

```
Format : update_db_field_types auto | disabled
```

(Default value: `auto`)

This parameter defines if Sympa may automatically update database structure to match the expected datafield types. This feature is only available with mysql.

7.10.2 `db_type`

```
Format : db_type mysql | SQLite | 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.

7.10.3 `db_name`

(Default value: `sympa`)

Name of the database containing user information. See detailed notes on database structure, **??**, page **??**. If you are using SQLite, then this parameter is the DB file name.

7.10.4 `db_host`

Database host name.

7.10.5 `db_port`

Database port.

7.10.6 `db_user`

User with read access to the database.

7.10.7 `db_passwd`

Password for `db_user`.

7.10.8 `db_timeout`

This parameter is used for SQLite only.

7.10.9 `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;mysql_socket=tmp/mysql.sock-test
```

Check the related DBD documentation to learn about the available options.

7.10.10 `db_env`

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

Example for Oracle :

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

7.10.11 `db_additional_subscriber_fields`

If your **subscriber_table** database table has more fields than required by *Sympa* (because other programs access this table), you can make *Sympa* recognize these fields. You will then be able to use them from within mail/web templates and authorization scenarios (as [subscriber->field]). These fields will also appear in the list members review page and will be editable by the list owner. This parameter is a comma-separated list.

Example :

```
db_additional_subscriber_fields  billing_delay,subscription_expiration
```

7.10.12 `db_additional_user_fields`

If your **user_table** database table has more fields than required by *Sympa* (because other programs access this table), you can make *Sympa* recognize these fields. You will then be able to use them from within mail/web templates (as [user->field]).

This parameter is a comma-separated list.

Example :

```
db_additional_user_fields  address,gender
```

7.10.13 `purge_user_table_task`

This parameter refers to the name of the task (Example: `monthly`) that will be regularly run by the `task_manager.pl` to remove entries in the user_table table that have no corresponding entries in the subscriber_table table.

7.11 Loop prevention

The following define your loop prevention policy for commands. (see 17.7, page 164)

7.11.1 `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.

7.11.2 `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.

7.11.3 `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.

7.11.4 `loop_prevention_regex`

(Default value: `mailer-daemon|sympa|listserv|majordomo|smartlist|mailman`)

This regular expression is applied to messages sender address. If the sender address matches the regular expression, then the message is rejected. The goal of this parameter is to prevent loops between Sympa and other robots.

7.12 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 set by the listmaster (see 27.4.3, page 251). The two others are optionnal.

7.12.1 `openssl`

The path for the openSSL binary file.

7.12.2 `capath`

The directory path use by openssl for trusted CA certificates.

A directory of trusted certificates. The certificates should have names of the form : hash.0 or have symbolic links to them of this form (”hash” is the hashed certificate subject name : see the -hash option of the openssl x509 utility). This directory should be the same as the directory SSLCACertificatePath specified for mod_ssl module for Apache.

7.12.3 `cafile`

This parameter sets the all-in-one file where you can assemble the Certificates of Certification Authorities (CA) whose clients you deal with. These are used for Client Authentication. Such a file is simply the concatenation of the various PEM-encoded Certificate files, in order of preference. This can be used alternatively and/or additionally to `capath`.

7.12.4 `key_passwd`

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

7.12.5 `chk_cert_expiration_task`

States the model version used to create the task which regularly checks the certificate expiration dates and warns users whose certificate have expired or are going to. To know more about tasks, see 17.8, page 164.

7.12.6 `crl_update_task`

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

7.13 Antivirus plug-in

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

7.13.1 `antivirus_path`

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

Example :

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

7.13.2 `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/kavscanner
antivirus_args  -Y -O- -MP -I0
```

Exemple with Sophos :

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

Exemple with Clam :

```
antivirus_path  /usr/local/bin/clamscan
antivirus_args  --stdout
```

7.13.3 `antivirus_notify`

`sender` — `nobody`

(Default value: `sender`)

This parameter tells if *Sympa* should notify the email sender when a virus has been detected.

Chapitre 8

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, SQLite, 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.

Sympa stores three kind of information in the database, each in one table :
– User preferences and passwords are stored in the user_table table
– List subscription informations are stored in the subscriber_table table, along with subscription options. This table also contains the cache for included users (if using include2 mode).
– List administrative informations are stored in the admin_table table if using include2 mode, along with owner and editor options. This table also contains the cache for included owners and editors.

8.1 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].

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

8.2 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 29 for installation details of these modules.

8.3 Creating a sympa DataBase

8.3.1 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.

The table concerning owners and editors, the admin_table, is made on the same way as the subscriber_table but is used only in include2 mode. It constains owner and editor options (list concerned, administrative role, date of "subscription", reception option, private info, gecos and profile option for owners).

8.3.2 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, SQLite, 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),
attributes_user varchar(255),
PRIMARY KEY (email_user)
);

CREATE TABLE subscriber_table (
   list_subscriber        varchar (50) NOT NULL,
user_subscriber varchar (100) NOT NULL,
robot_subscriber varchar (80) NOT NULL,
date_subscriber datetime NOT NULL,
update_subscriber datetime,
visibility_subscriber varchar (20),
reception_subscriber varchar (20),
topics_subscriber varchar (200),
bounce_subscriber varchar (35),
bounce_score_subscriber smallint (6),
bounce_address_subscriber varchar (100),
comment_subscriber varchar (150),
subscribed_subscriber  int(1),
included_subscriber  int(1),
include_sources_subscriber varchar(50),
PRIMARY KEY (list_subscriber, user_subscriber, robot_subscriber),
INDEX (user_subscriber,list_subscriber,robot_subscriber)
);

CREATE TABLE admin_table (
list_admin  varchar(50) NOT NULL,
  user_admin  varchar(100) NOT NULL,
  robot_admin  varchar(80) NOT NULL,
role_admin  enum('listmaster','owner','editor') NOT NULL,
date_admin  datetime NOT NULL,
update_admin  datetime,
reception_admin  varchar(20),
comment_admin  varchar(150),
subscribed_admin   int(1),
included_admin   int(1),
include_sources_admin   varchar(50),
info_admin    varchar(150),
profile_admin   enum('privileged','normal'),
PRIMARY KEY (list_admin, user_admin, robot_admin, role_admin),
INDEX (list_admin, user_admin,robot_admin,role_admin)
);

CREATE TABLE netidmap_table (
        netid_netidmap             varchar (100) NOT NULL,
serviceid_netidmap     varchar (100) NOT NULL,
robot_netidmap             varchar (80) NOT NULL,
        email_netidmap              varchar (100),
        PRIMARY KEY (netid_netidmap, serviceid_netidmap, robot_netidmap)
);

CREATE TABLE logs_table (
id_logs bigint(20) NOT NULL,
date_logs int(11) NOT NULL,
robot_logs varchar(80),
list_logs varchar(50),
action_logs varchar(50) NOT NULL,
parameters_logs varchar(100),
target_email_logs varchar(100),
user_email_logs varchar(100),
msg_id_logs varchar(255),
status_logs varchar(10) NOT NULL,
error_type_logs varchar(150),
client_logs varchar(100),
daemon_logs varchar(10) NOT NULL,
PRIMARY KEY (id_logs)
);
```

– SQLiteL database creation script

```
CREATE TABLE user_table (
   email_user          varchar (100) NOT NULL,
   gecos_user          varchar (150),
   password_user varchar (40),
cookie_delay_user integer,
lang_user varchar (10),
attributes_user varchar(255),
PRIMARY KEY (email_user)
);

CREATE TABLE subscriber_table (
   list_subscriber        varchar (50) NOT NULL,
user_subscriber varchar (100) NOT NULL,
robot_subscriber varchar (80) NOT NULL,
date_subscriber timestamp NOT NULL,
update_subscriber timestamp,
visibility_subscriber varchar (20),
reception_subscriber varchar (20),
topics_subscriber      varchar (200),
bounce_subscriber varchar (35),
bounce_address_subscriber varchar (100),
comment_subscriber varchar (150),
subscribed_subscriber  boolean,
included_subscriber  boolean,
include_sources_subscriber varchar(50),
bounce_score_subscriber integer,
PRIMARY KEY (list_subscriber, user_subscriber, robot_subscriber)
);
CREATE INDEX subscriber_idx ON subscriber_table (user_subscriber,list_subscriber,robot_su

CREATE TABLE admin_table (
list_admin  varchar(50) NOT NULL,
  user_admin  varchar(100) NOT NULL,
  robot_admin  varchar(80) NOT NULL,
role_admin  varchar(15) NOT NULL,
date_admin  timestamp NOT NULL,
update_admin  timestamp,
reception_admin  varchar(20),
comment_admin  varchar(150),
subscribed_admin   boolean,
included_admin   boolean,
include_sources_admin   varchar(50),
info_admin    varchar(150),
profile_admin   varchar(15),
PRIMARY KEY (list_admin, user_admin, robot_admin, role_admin)
);
CREATE INDEX admin_idx ON admin_table(list_admin, user_admin, robot_admin, role_admin);

CREATE TABLE netidmap_table (
        netid_netidmap              varchar (100) NOT NULL,
serviceid_netidmap     varchar (100) NOT NULL,
robot_netidmap             varchar (80) NOT NULL,
        email_netidmap              varchar (100),
        PRIMARY KEY (netid_netidmap, serviceid_netidmap, robot_netidmap)
);
CREATE INDEX netidmap_idx ON netidmap_table(netid_netidmap, serviceid_netidmap, robot_net

CREATE TABLE logs_table (
id_logs integer NOT NULL,
date_logs integer NOT NULL,
robot_logs varchar(80),
list_logs varchar(50),
action_logs varchar(50) NOT NULL,
parameters_logs varchar(100),
target_email_logs varchar(100),
user_email_logs varchar(100),
msg_id_logs varchar(255),
status_logs varchar(10) NOT NULL,
error_type_logs varchar(150),
client_logs varchar(100),
daemon_logs varchar(10) NOT NULL,
PRIMARY KEY (id_logs)
);
CREATE INDEX logs_idx ON logs_table(id_logs);
```

– 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),
attributes_user varchar (255),
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,
robot_subscriber varchar (80) NOT NULL,
date_subscriber timestamp with time zone NOT NULL,
update_subscriber  timestamp with time zone,
visibility_subscriber varchar (20),
reception_subscriber varchar (20),
topics_subscriber varchar (200),
bounce_subscriber varchar (35),
bounce_score_subscriber int4,
bounce_address_subscriber varchar (100),
comment_subscriber varchar (150),
subscribed_subscriber  smallint,
included_subscriber  smallint,
include_sources_subscriber varchar(50),
CONSTRAINT ind_subscriber PRIMARY KEY (list_subscriber, user_subscriber, robot_
);
CREATE INDEX subscriber_idx ON subscriber_table (user_subscriber,list_subscribe

DROP TABLE admin_table;
CREATE TABLE admin_table (
list_admin  varchar(50) NOT NULL,
  user_admin  varchar(100) NOT NULL,
  robot_admin  varchar(80) NOT NULL,
role_admin  varchar(15) NOT NULL,
date_admin  timestamp with time zone NOT NULL,
update_admin  timestamp with time zone,
reception_admin  varchar(20),
comment_admin  varchar(150),
subscribed_admin   smallint,
included_admin   smallint,
include_sources_admin   varchar(50),
info_admin    varchar(150),
profile_admin   varchar(15),
        CONSTRAINT ind_admin PRIMARY KEY (list_admin, user_admin, robot_admin, role_admir
);
CREATE INDEX admin_idx ON admin_table(list_admin, user_admin,robot_admin, role_admin);

DROP TABLE netidmap_table;
CREATE TABLE netidmap_table (
        netid_netidmap              varchar (100) NOT NULL,
serviceid_netidmap     varchar (100) NOT NULL,
robot_netidmap            varchar (80) NOT NULL,
        email_netidmap              varchar (100),
        CONSTRAINT ind_netidmap PRIMARY KEY (netid_netidmap, serviceid_netidmap, robot_ne
);
CREATE INDEX netidmap_idx ON netidmap_table(netid_netidmap, serviceid_netidmap, robot_net

DROP TABLE logs_table;
CREATE TABLE logs_table (
id_logs bigint NOT NULL,
date_logs int4 NOT NULL,
robot_logs varchar (80),
list_logs varchar (50),
action_logs varchar (50) NOT NULL,
parameters_logs varchar (100),
target_email_logs varchar (100),
user_email_logs varchar (100),
msg_id_logs varchar (255),
status_logs varchar (10) NOT NULL,
error_type_logs varchar (150),
client_logs varchar (100),
daemon_logs varchar (10) NOT NULL,
        CONSTRAINT ind_logs PRIMARY KEY (id_logs)
);
CREATE INDEX logs_idx ON logs_table(id_logs);
```

– 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    ,
    attributes_user    varchar(255)           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,
    robot_subscriber       varchar(80)           not null,
    date_subscriber        datetime               not null,
    update_subscriber      datetime               null,
    visibility_subscriber  varchar(20)            null    ,
    reception_subscriber   varchar(20)            null    ,
    topics_subscriber    varchar(200)           null,
    bounce_subscriber      varchar(35)            null    ,
    bounce_score_subscriber numeric               null    ,
    comment_subscriber     varchar(150)           null    ,
    subscribed_subscriber  numeric   null    ,
    included_subscriber    numeric                null    ,
    include_sources_subscriber varchar(50)        null    ,
    constraint ind_subscriber primary key (list_subscriber, user_subscriber, ro
)
go

create index list_subscriber_fk on subscriber_table (list_subscriber)
go

create index user_subscriber_fk on subscriber_table (user_subscriber)
go

create index robot_subscriber_fk on subscriber_table (robot_subscriber)
go

create table admin_table
(
list_admin  varchar(50)       not null,
  user_admin  varchar(100)    not null,
  robot_admin  varchar(80)    not null,
role_admin  varchar(15)       not null,
date_admin  datetime    not null,
update_admin  datetime          null,
reception_admin  varchar(20)       null,
comment_admin  varchar(150)   null,
subscribed_admin   numeric           null,
included_admin   numeric           null,
include_sources_admin   varchar(50)       null,
info_admin    varchar(150)      null,
profile_admin   varchar(15)       null,
       constraint ind_admin primary key (list_admin, user_admin,robot_admin,role_admin)
)
go

create index list_admin_fk on admin_table (list_admin)
go

create index user_admin_fk on admin_table (user_admin)
go

create index robot_admin_fk on admin_table (robot_admin)
go

create index role_admin_fk on admin_table (role_admin)
go

create table netidmap_table
(
        netid_netidmap             varchar (100) NOT NULL,
serviceid_netidmap    varchar (100) NOT NULL,
robot_netidmap            varchar (80) NOT NULL,
        email_netidmap             varchar (100),
        constraint ind_netidmap primary key (netid_netidmap, serviceid_netidmap, robot_ne
)
go

create index netid_netidmap_fk on netidmap_table (netid_netidmap)
go

create index serviceid_netidmap_fk on netidmap_table (serviceid_netidmap)
go

create index robot_netidmap_fk on netidmap_table (robot_netidmap)
go

CREATE TABLE logs_table (
id_logs numeric NOT NULL,
date_logs numeric NOT NULL,
robot_logs varchar(80),
list_logs varchar(50),
action_logs varchar(50) NOT NULL,
parameters_logs varchar(100),
target_email_logs varchar(100),
user_email_logs varchar(100),
msg_id_logs varchar(255),
status_logs varchar(10) NOT NULL,
error_type_logs varchar(150),
client_logs varchar(100),
daemon_logs varchar(10) NOT NULL,
constraint ind_logs primary key (id_logs)
)
go

create index id_logs_fk on logs_table (id_logs)
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),
attributes_user varchar2(500),
        CONSTRAINT ind_user PRIMARY KEY (email_user)
);
CREATE TABLE subscriber_table (
        list_subscriber         varchar2(50) NOT NULL,
        user_subscriber         varchar2(100) NOT NULL,
        robot_subscriber         varchar2(80) NOT NULL,
        date_subscriber         date NOT NULL,
update_subscriber date,
        visibility_subscriber   varchar2(20),
        reception_subscriber    varchar2(20),
topics_subscriber varchar2(200),
  bounce_subscriber       varchar2 (35),
bounce_score_subscriber number,
bounce_address_subscriber       varchar2 (100),
  comment_subscriber      varchar2 (150),
subscribed_subscriber  number NULL    constraint  cons_subscribed_subscriber CHECK (subsc
included_subscriber  number NULL    constraint  cons_included_subscriber CHECK (included_
include_sources_subscriber varchar2(50),
        CONSTRAINT ind_subscriber PRIMARY KEY (list_subscriber,user_subscriber,robot_subs
);
CREATE TABLE admin_table (
list_admin  varchar2(50) NOT NULL,
  user_admin  varchar2(100) NOT NULL,
  robot_admin  varchar2(80) NOT NULL,
role_admin  varchar2(20) NOT NULL,
date_admin  date NOT NULL,
update_admin  date,
reception_admin  varchar2(20),
comment_admin  varchar2(150),
subscribed_admin   number NULL    constraint  cons_subscribed_admin CHECK (subscribed_adm
included_admin   number NULL    constraint  cons_included_admin CHECK (included_admin in
include_sources_admin   varchar2(50),
info_admin    varchar2(150),
profile_admin   varchar2(20),
       CONSTRAINT ind_admin PRIMARY KEY (list_admin,user_admin,robot_admin,role_admin)
);

CREATE TABLE netidmap_table (
        netid_netidmap              varchar2 (100) NOT NULL,
serviceid_netidmap     varchar2 (100) NOT NULL,
robot_netidmap             varchar2 (80) NOT NULL,
        email_netidmap              varchar2 (100),
        CONSTRAINT ind_netidmap PRIMARY KEY (netid_netidmap, serviceid_netidmap, robot_ne
);

CREATE TABLE logs_table (
id_logs number NOT NULL,
date_logs number NOT NULL,
robot_logs varchar2 (80),
list_logs varchar2 (50),
action_logs varchar2 (50) NOT NULL,
parameters_logs varchar2 (100),
target_email_logs varchar2 (100),
user_email_logs varchar2 (100),
msg_id_logs varchar2 (255),
status_logs varchar2 (10) NOT NULL,
error_type_logs varchar2 (150),
client_logs varchar2 (100),
daemon_logs varchar2 (10) NOT NULL,
CONSTRAINT ind_admin PRIMARY KEY (id_logs)
);

!
```

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

Example :

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

8.4 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 MySQL :

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

8.5 Importing subscribers data

8.5.1 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 4.1, page 35).

8.5.2 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 easiest 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 big, a timeout may occur during the FastCGI execution (Note that you can set a longer timeout with the `-idle-timeout` option of the `FastCgiServer` Apache configuration directive). In this case, or if you have not installed *WWSympa*, you should use the `load_subscribers.pl` script.

8.6 Management of the include cache

You may dynamically add a list of subscribers, editors or owners to a list with Sympa's **include2** user data source. Sympa is able to query multiple data sources (RDBMS, LDAP directory, flat file, a local list, a remote list) to build a mailing list.

Sympa used to manage the cache of such *included* subscribers in a DB File (**include** mode) but now stores subscribers, editors and owners in the database (**include2** mode). These changes brought the following advantages :
- Sympa processes are smaller when dealing with big mailing lists (in include mode)
- Cache update is now performed regularly by a dedicated process, the task manager
- Mixed lists (included + subscribed users) can now be created
- Sympa can now provide reception options for *included* members
- Bounces information can be managed for *included* members
- Sympa keeps track of the data sources of a member (available on the web REVIEW page)
- *included* members can also subscribe to the list. It allows them to remain in the list though they might no more be included.

8.7 Extending database table format

You can easily add other fields to the three tables, they will not disturb *Sympa* because it lists explicitely the field it expects 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 7.10.11, page 74 and 7.10.12, page 74).

8.8 *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 9

WWSympa, Sympa's web interface

WWSympa is *Sympa*'s web interface.

9.1 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 17.1, page 159). 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

9.2 Web server setup

9.2.1 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 achieve this :

– SetuidPerl : this is the default method but might be insecure. If you don't set the **- -enable_secure** configure option, `wwsympa.fcgi` is installed with the SetUID bit set. On most systems you will need to install the suidperl package.

– Sudo : use **sudo** to run `wwsympa.fcgi` as user sympa. Your Apache configuration should use `wwsympa_sudo_wrapper.pl` instead of `wwsympa.fcgi`. You should edit your `/etc/sudoers` file (with visudo command) as follows :

```
apache ALL = (sympa)  NOPASSWD: /usr/local/sympa-os/bin/wwsympa.fcgi
```

– Dedicated Apache server : run a dedicated Apache server with sympa.sympa as uid.gid (The Apache default is apache.apache).

– Apache suExec : use an Apache virtual host 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 The User and Group directive have to be set before the FastCgiServer directive is encountered.

– C wrapper : 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 "/usr/local/sympa-os/bin/wwsympa.fcgi"

int main(int argn, char **argv, char **envp) {
```

```
          argv[0] = WWSYMPA;
          execve(WWSYMPA,argv,envp);
      }
```

9.2.2 Installing wwsympa.fcgi in your Apache server

You first need to set an alias to the directory where Sympa stores static contents (CSS, members pictures, documentation) directly delivered by Apache

```
Example :
   Alias /static-sympa /usr/local/sympa-os/static_content
```

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

```
Example :
   ScriptAlias /sympa /usr/local/sympa-os/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 /usr/local/sympa-os/bin/wwsympa.fcgi -processes 2
<Location /sympa>
      SetHandler fastcgi-script
</Location>

ScriptAlias /sympa /usr/local/sympa-os/bin/wwsympa.fcgi
```

If you are using **sudo** (see evious subsection), then replace `wwsympa.fcgi` calls with `wwsympa_sudo_wrapper.pl`.

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

9.2.3 Using FastCGI

FastCGI is an extention to CGI that provides persistency for CGI programs. It is extemely useful with *WWSympa* since source code interpretation and all initialisation tasks are performed only once, at server startup ; then file wwsympa.fcgi instances are waiting for clients requests.

WWSympa can also work without FastCGI, depending on the **use_fast_cgi** parameter (see 9.3.15, page 99).

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*

9.3 wwsympa.conf parameters

9.3.1 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.

9.3.2 archive_default_index thrd — mail

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

9.3.3 archived_pidfile

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

9.3.4 bounce_path

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

9.3.5 bounced_pidfile

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

9.3.6 cookie_expire

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

9.3.7 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.

9.3.8 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).

9.3.9 icons_url

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

9.3.10 log_facility

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

9.3.11 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

9.3.12 htmlarea_url

(Default value: `undefined`)
Relative URL to the (superb) online html editor HTMLarea. If you have installed javascript application you can use it when editing html document in the shared document repository. In order to activate this pluggin the value of this parameter should point to the root directory where HTMLarea is installed. HTMLarea is a free opensource software you can download here : http ://sf.net/projects/itools-htmlarea/

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

9.3.14 title

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

9.3.15 use_fast_cgi 0 — 1

(Default value: `1`)
Choice of whether or not to use FastCGI. On listes.cru.fr, using FastCGI increases WW-Sympa 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.

9.4 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. MhOnArc is called to produce not a complete html document, but only a part of it to be included in a complete document (starting with <HTML> and terminating with </HTML> ;-)). The best way is to use the MhOnArc resource file provided in the *WWSympa* distribution and to modify it for your needs.

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

1. `/usr/local/sympa-os/expl/mylist/mhonarc-ressources` in order to create a specific archive look for a particular list
2. or `/usr/local/sympa-os/etc/mhonarc-ressources`

9.5 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 : `/usr/local/sympa-os/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
quota 10000
```

If web_archive is defined for a list, every message distributed by this list is copied to `/usr/local/sympa-os/spool/outgoing/`. (No need to create nonexistent subscribers to receive copies of messages). In this example disk quota (expressed in Kbytes) for the archive is limited to 10 Mo.

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 `/usr/local/sympa-os/spool/outgoing`, and make sure that the owner of this file is *Sympa*.

```
example : su sympa -c "touch /usr/local/sympa-os/spool/outgoing/.rebuil
```

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

Furthermore, if you want to get list's archives, you can do it via the `List-admin menu-> Archive Management`

9.6 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, SQLite, 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.

9.7 Logging in as listmaster

Once *Sympa* is running you should log in on the web interface as a privileged user (listmaster) to explore the admin interface, create mailing lists.

Multiple email addresses can be declared as listmaster via the `sympa.conf` (or `robot.conf`) `listmaster` configuration parameter (see 7, page 49). Note that listmasters on the main robot (declared in `sympa.conf`) also have listmaster privileges on the virtual hosts but they will not receive the various mail notifications (list creation, warnings,...) regarding these virtual hosts.

The listmasters should log in with their canonical email address as an identifier (not *listmaster@my.host*). The associated password is not declared in sympa.conf ; it will be allocated by *Sympa* when first hitting the **Send me a password** button on the web interface. As for any user, the password can then be modified via the **Preferenced** menu.

Note that you must start the `sympa.pl` process with the web interface ; it is in responsible for delivering mail messages including password reminders.

Chapitre 10

Sympa Internationalization

10.1 Catalogs and templates

Sympa is designed to allow easy internationalization of its user interface (service mail messages and web interface). All translations for one language are gathered in a single PO file that can be manipulated by standard GNU gettext tools[1].

Documentation and ressources about software translations : http ://translate.sourceforge.net/doc/[2]

Sympa previously (until Sympa 4.1.x) used XPG4 messages catalogue format. Web and mail templates were language specific. The new organization both provide a unique file to work on for translators and a standard format supported by many software. Sympa templates refer to translatable strings using the `loc` TT2 filter.

Examples :

```
[%|loc%]User Email[%END%]

[%|loc(list.name,user.email)%]You have subscribed to list %1 with email address %2[%END%]
```

Sympa had previously been translated into 15 languages more or less completely. We have automatically extracted the translatable strings from previous templates but this process is awkward and is only seen as a bootstrap for translators. Therefore Sympa distribution will not include previous translations until a skilled translator has reviewed and updated the corresponding PO file.

[1]http ://www.gnu.org/software/gettext/#TOCintroduction
[2]http ://translate.sourceforge.net/doc/

10.2 Translating Sympa GUI in your language

Instructions for translating Sympa are maintained on Sympa web site : http ://www.sympa.org/howtotranslate.html

10.3 Defining language-specific templates

The default Sympa templates are language independant, refering to catalogue entries for translations. When customizing either web or mail templates, you can define different templates for different languages. The template should be located in a ll_CC subdirectory of `web_tt2` or `mail_tt2` with the language code.

Example :

```
/web_tt2/home.tt2
/web_tt2/de_DE/home.tt2
/web_tt2/fr_FR/home.tt2
```

This mecanism also applies to `comment.tt2` files used by create list templates.

Web templates can also make use of the `locale` variable to make templates multilingual :

Example :

```
[% IF locale == 'fr_FR' %]
Personnalisation
[% ELSE %]
Customization
[% END %]
```

10.4 Translating topics titles

Topics are defined in a `topics.conf` file. In this file, each entry can be given a title in different languages, see 17.5, page 163.

10.5 Handling of encodings

Until version 5.3, Sympa web pages were encoded in each language's encoding (iso-8859-1 for French, utf-8 for Japanese,...) whereas every web page is now encoded in utf-8. Thanks to the `Encode` Perl module, Sympa can now juggle with the filesystem encoding, each message catalog's encoding and its web encoding (utf-8).

If your operating system uses a character encoding different from utf-8, then you should declare it using the `filesystem_encoding` sympa.conf parameter (see **??**, page **??**). It is required to do so because Sympa has no way to find out what encoding is used for its configuration files. Once this encoding is known, every template or configuration parameter can be read properly for the web and also saved properly when edited from the web interface.

Note that the shared documents (see23, page 233) filenames are Q-encoded to make their storage encoding neutral. This encoding is transparent for the end-users.

Chapitre 11

Sympa RSS channel

This service is provided by *WWSympa* (*Sympa*'s web interface). Here is the root of *WWSympa*'s rss channel :

(Default value: `http ://<host>/wws/rss`)
Example: `https ://my.server/wws/rss`

The access control of RSS queries proceed on the same way as *WWSympa* actions referred to. *Sympa* provides the following RSS features :
– the latest created lists on a robot (`latest_lists`) ;
– the most active lists on a robot(`active_lists`) ;
– the latest messages of a list (`active_arc`) ;
– the latest shared documents of a list (`latest_d_read`) ;

11.1 `latest_lists`

This provides the latest created lists.

Example: `http ://my.server/wws/rss/latest_lists ?for=3&count=6`
This provides the 6 latest created lists for the last 3 days.

Example: `http ://my.server/wws/rss/latest_lists/computing ?count=6`
This provides the 6 latest created lists with topic "computing".

Parameters :

- `for` : period of interest (expressed in days). This is a CGI parameter. It is optional but one of the two parameters "for" or "count" is required.
- `count` : maximum number of expected records. This is a CGI parameter. It is optional but one of the two parameters "for" or "count" is required.
- topic : the topic is indicated in the path info (see example below with topic computing). This parameter is optional.

11.2 `active_lists`

This provides the most active lists, based on the number of distributed messages (number of received messages).

Example: `http ://my.server/wws/rss/active_lists ?for=3&count=6`
This provides the 6 most active lists for the last 3 days.

Example: `http ://my.server/wws/rss/active_lists/computing ?count=6`
This provides the 6 most active lists with topic "computing".

Parameters :
- `for` : period of interest (expressed in days). This is a CGI parameter. It is optional but one of the two parameters "for" or "count" is required.
- `count` : maximum number of expected records. This is a CGI parameter. It is optional but one of the two parameters "for" or "count" is required.
- topic : the topic is indicated in the path info (see example below with topic computing). This parameter is optional.

11.3 `latest_arc`

This provides the latest messages of a list.

Example: `http ://my.server/wws/rss/latest_arc/mylist ?for=3&count=6`
This provides the 6 latest messages received on the mylistlist for the last 3 days.

Parameters :
- list : the list is indicated in the path info. This parameter is mandatory.
- `for` : period of interest (expressed in days). This is a CGI parameter. It is optional but one of the two parameters "for" or "count" is required.
- `count` : maximum number of expected records. This is a CGI parameter. It is optional but one of the two parameters "for" or "count" is required.

11.4 `latest_d_read`

This provides the latest updated and uploaded shared documents of a list.

Example: `http ://my.server/wws/rss/latest_d_read/mylist ?for=3&count=6`
This provides the 6 latest documents uploaded or updated on the mylistlist for the last 3 days.

Parameters :
– list : the list is indicated in the path info. This parameter is mandatory.
– `for` : period of interest (expressed in days). This is a CGI parameter. It is optional but one of the two parameters “for” or “count” is required.
– `count` : maximum number of expected records. This is a CGI parameter. It is optional but one of the two parameters “for” or “count” is required.

Chapitre 12

Sympa SOAP server

12.1 Introduction

SOAP is one protocol (generally over HTTP) that can be used to provide **web services**. Sympa SOAP server allows to access a Sympa service from within another program, written in any programming language and on any computer. SOAP encapsulates procedure calls, input parameters and resulting data in an XML data structure. The Sympa SOAP server's API is published in a **WSDL** document, retreived via Sympa's web interface.

The SOAP server provides a limited set of high level functions including `login`, `which`, `lists`, `subscribe`, `signoff` and list creation. Other functions might be implemented in the future. One of the important implementation constraint is to provide services for proxy application with a correct authorization evaluation processus where authentication may differ from classic web method. The following cases can be used to access to the service :

- The client first ask for a login and later service request provide the `sympa-user` cookie.
- The client authenticate the end user providing the `sympa-user` http cookie. This can be used in order to share the an authenticated session betwing Sympa and some other application running on the same server as wwsympa. The soap method used is `getUserEmailByCookieRequest`.
- The client provide user email and password and request a service in a single soap access using the `authenticateAndRun` soap service.
- The client is a trusted by Sympa as a proxy application and is authorized to set some variables that will be used by Sympa during the authorization scenario evaluation. Trusted application have there own password and the variables they can set are listed in a configuration file name `trusted_applications.conf`. See 12.4 page 112.

In any case scenario authorization is used with same rules as mail interface or usual web interface.

The SOAP server uses SOAP : :Lite Perl library. The server is running as a daemon (thanks to FastCGI), receiving the client SOAP requests via a web server (Apache for example).

12.2 Web server setup

You **NEED TO** install FastCGI for the SOAP server to work properly because it will run as a daemon.

Here is a sample piece of your Apache `httpd.conf` with a SOAP server configured :

```
FastCgiServer /usr/local/sympa-os/bin/sympa_soap_server.fcgi -processes 1
ScriptAlias /sympasoap /usr/local/sympa-os/bin/sympa_soap_server.fcgi

<Location /sympasoap>
      SetHandler fastcgi-script
</Location>
```

12.3 Sympa setup

The only mandatory parameter you need to set in `sympa.conf`/`robot.conf` files is the `soap_url` that defines the URL of the SOAP service corresponding to the ScriptAlias you've previously setup in Apache config.

This parameter is used to publish the SOAP service URL in the WSDL file (defining the API) but also for the SOAP server to deduce what Virtual Host is concerned by the current SOAP request (a single SOAP server will serve all Sympa virtual hosts).

12.4 trust remote application

The SOAP service `authenticateRemoteAppAndRun` is used in order to allow some remote application such as a web portal to request Sympa service as a proxy for the end user. In such case, Sympa will not authenticate the end user itself but instead it will trust a particular application to act as a proxy.

This configuration file `trusted_applications.conf` can be created in the robot `etc/` subdirectory or in `/usr/local/sympa-os/etc` directory depending on the scope you want for it (the source package include a sample of file `trusted_applications.conf` in directory `soap`). This file is constructed with paragraphs separated by empty line and stating with key word `trusted_application`.

A sample `trusted_applications.conf` file is provided with Sympa sources. Each paragraph defines a remote trusted application with keyword/value pairs

- `name` : the name of the application. Used with password for authentication; the `remote_application_name` variable is set for use in authorization scenarios.
- `md5password` : the MD5 digest of the application password. You can compute the digest as follows : `sympa.pl -md5_digest=<the password>`.
- `proxy_for_variables` : a comma separated list of variables that can be set by the remote application and that will be used by Sympa SOAP server when evaluating an authorization scenario. If you list `USER_EMAIL` in this parameter, then the remote application can act as a user. Any other variable such as `remote_host` can be listed.

You can test your SOAP service using the `sympa_soap_client.pl` sample script as follows :

```
/usr/local/sympa-os/bin/sympa_soap_client.pl --soap_url=http://my.server/sympasoap --servi

/usr/local/sympa-os/bin/sympa_soap_client.pl --soap_url=http://myserver/sympasoap --servic
```

Availible services are :
- info ¡list¿
- which
- lists
- review ¡list¿
- amI ¡function¿
- subscribe ¡list¿
- signoff ¡list¿
- add ¡list¿¡email¿
- del ¡list¿¡email¿
- createList ¡list¿...
- closeList ¡list¿
- login ¡email¿¡password¿
- casLogin ¡proxyTicket¿
- checkCookie

12.5 The WSDL service description

Here is what the WSDL file looks like before it is parsed by WWSympa :

```
<?xml version="1.0"?>
<definitions name="Sympa"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
targetNamespace="[% conf.wwsympa_url %]/wsdl"
xmlns:tns="[% conf.wwsympa_url %]/wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsdl="[% conf.soap_url %]/wsdl">
```

```
<!-- types part -->


<types>
<schema targetNamespace="[% conf.wwsympa_url %]/wsdl"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://www.w3.org/2001/XMLSchema">

<complexType name="ArrayOfLists">
<complexContent>
<restriction base="SOAP-ENC:Array">
<attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="tns:listType[]"/>
</restriction>
</complexContent>
</complexType>

<complexType name="ArrayOfString">
<complexContent>
<restriction base="SOAP-ENC:Array">
<attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="string[]"/>
</restriction>
</complexContent>
</complexType>

<complexType name="listType">
           <all>
<element name="listAddress" minOccurs="1" type="string"/>
<element name="homepage" minOccurs="0" type="string"/>
<element name="isSubscriber" minOccurs="0" type="boolean"/>
<element name="isOwner" minOccurs="0" type="boolean"/>
<element name="isEditor" minOccurs="0" type="boolean"/>
<element name="subject" minOccurs="0" type="string"/>
           </all>
</complexType>
</schema>
</types>


<!-- message part -->

<message name="infoRequest">
        <part name="listName" type="xsd:string"/>
</message>

<message name="infoResponse">
<part name="return" type="tns:listType"/>
</message>
```

```
<message name="complexWhichRequest">
</message>

<message name="complexWhichResponse">
<part name="return" type="tns:ArrayOfLists"/>
</message>

<message name="whichRequest">
</message>

<message name="whichResponse">
<part name="return" type="tns:ArrayOfString"/>
</message>

<message name="amIRequest">
<part name="list" type="xsd:string"/>
<part name="function" type="xsd:string"/>
<part name="user" type="xsd:string"/>
</message>

<message name="amIResponse">
<part name="return" type="xsd:boolean"/>
</message>

<message name="reviewRequest">
<part name="list" type="xsd:string"/>
</message>

<message name="reviewResponse">
<part name="return" type="tns:ArrayOfString"/>
</message>

<message name="signoffRequest">
<part name="list" type="xsd:string"/>
<part name="email" type="xsd:string" xsd:minOccurs="0"/>
</message>

<message name="signoffResponse">
<part name="return" type="xsd:boolean"/>
</message>

<message name="subscribeRequest">
<part name="list" type="xsd:string"/>
<part name="gecos" type="xsd:string" xsd:minOccurs="0"/>
</message>

<message name="addRequest">
<part name="list" type="xsd:string"/>
<part name="email" type="xsd:string"/>
```

```
<part name="gecos" type="xsd:string"  xsd:minOccurs="0"/>
<part name="quiet" type="xsd:boolean"  xsd:minOccurs="0"/>
</message>


<message name="addResponse">
<part name="return" type="xsd:boolean"/>
</message>

<message name="delRequest">
<part name="list" type="xsd:string"/>
<part name="email" type="xsd:string"/>
<part name="quiet" type="xsd:boolean"  xsd:minOccurs="0"/>
</message>


<message name="delResponse">
<part name="return" type="xsd:boolean"/>
</message>

<message name="createListRequest">
<part name="list" type="xsd:string"/>
<part name="subject" type="xsd:string"/>
<part name="template" type="xsd:string"/>
<part name="description" type="xsd:string"/>
<part name="topics" type="xsd:string"/>
</message>

<message name="createListResponse">
<part name="return" type="xsd:boolean"/>
</message>

<message name="closeListRequest">
<part name="list" type="xsd:string"/>
</message>

<message name="closeListResponse">
<part name="return" type="xsd:boolean"/>
</message>

<message name="subscribeResponse">
<part name="return" type="xsd:boolean"/>
</message>

<message name="loginRequest">
<part name="email" type="xsd:string"/>
<part name="password" type="xsd:string"/>
</message>

<message name="loginResponse">
```

```
<part name="return" type="xsd:string"/>
</message>

<message name="getUserEmailByCookieRequest">
<part name="cookie" type="xsd:string"/>
</message>

<message name="getUserEmailByCookieResponse">
<part name="return" type="xsd:string"/>
</message>

<message name="authenticateAndRunRequest">
<part name="email" type="xsd:string"/>
<part name="cookie" type="xsd:string"/>
<part name="service" type="xsd:string"/>
<part name="parameters" type="tns:ArrayOfString" xsd:minOccurs="0"/>
</message>

<message name="authenticateAndRunResponse">
<part name="return" type="tns:ArrayOfString" xsd:minOccurs="0"/>
</message>

<message name="authenticateRemoteAppAndRunRequest">
<part name="appname" type="xsd:string"/>
<part name="apppassword" type="xsd:string"/>
<part name="vars" type="xsd:string"/>
<part name="service" type="xsd:string"/>
<part name="parameters" type="tns:ArrayOfString" xsd:minOccurs="0"/>
</message>

<message name="authenticateRemoteAppAndRunResponse">
<part name="return" type="tns:ArrayOfString" xsd:minOccurs="0"/>
</message>

<message name="casLoginRequest">
<part name="proxyTicket" type="xsd:string"/>
</message>

<message name="casLoginResponse">
<part name="return" type="xsd:string"/>
</message>

<message name="listsRequest">
<part name="topic" type="xsd:string" xsd:minOccurs="0"/>
<part name="subtopic" type="xsd:string" xsd:minOccurs="0"/>
</message>

<message name="listsResponse">
<part name="listInfo" type="xsd:string"/>
</message>
```

```
<message name="complexListsRequest">
</message>

<message name="complexListsResponse">
<part name="return" type="tns:ArrayOfLists"/>
</message>

<message name="checkCookieRequest">
</message>

<message name="checkCookieResponse">
<part name="email" type="xsd:string"/>
</message>


<!-- portType part -->

<portType name="SympaPort">
<operation name="info">
<input message="tns:infoRequest" />
<output message="tns:infoResponse" />
</operation>
<operation name="complexWhich">
<input message="tns:complexWhichRequest" />
<output message="tns:complexWhichResponse" />
</operation>
<operation name="which">
<input message="tns:whichRequest" />
<output message="tns:whichResponse" />
</operation>
<operation name="amI">
<input message="tns:amIRequest" />
<output message="tns:amIResponse" />
</operation>
<operation name="add">
<input message="tns:addRequest" />
<output message="tns:addResponse" />
</operation>
<operation name="del">
<input message="tns:delRequest" />
<output message="tns:delResponse" />
</operation>
<operation name="createList">
<input message="tns:createListRequest" />
<output message="tns:createListResponse" />
</operation>
<operation name="closeList">
<input message="tns:closeListRequest" />
<output message="tns:closeListResponse" />
```

```
</operation>
<operation name="review">
<input message="tns:reviewRequest" />
<output message="tns:reviewResponse" />
</operation>
<operation name="subscribe">
<input message="tns:subscribeRequest" />
<output message="tns:subscribeResponse" />
</operation>
<operation name="signoff">
<input message="tns:signoffRequest" />
<output message="tns:signoffResponse" />
</operation>
<operation name="login">
<input message="tns:loginRequest" />
<output message="tns:loginResponse" />
</operation>
<operation name="casLogin">
<input message="tns:casLoginRequest" />
<output message="tns:casLoginResponse" />
</operation>
<operation name="getUserEmailByCookie">
<input message="tns:getUserEmailByCookieRequest" />
<output message="tns:getUserEmailByCookieResponse" />
</operation>
<operation name="authenticateAndRun">
<input message="tns:authenticateAndRunRequest" />
<output message="tns:authenticateAndRunResponse" />
</operation>
<operation name="authenticateRemoteAppAndRun">
<input message="tns:authenticateRemoteAppAndRunRequest" />
<output message="tns:authenticateRemoteAppAndRunResponse" />
</operation>
<operation name="lists">
<input message="tns:listsRequest" />
<output message="tns:listsResponse" />
</operation>
<operation name="complexLists">
<input message="tns:complexListsRequest" />
<output message="tns:complexListsResponse" />
</operation>
<operation name="checkCookie">
<input message="tns:checkCookieRequest" />
<output message="tns:checkCookieResponse" />
</operation>
</portType>


<!-- Binding part -->
```

```
<binding name="SOAP" type="tns:SympaPort">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="info">
<soap:operation soapAction="urn:sympasoap#info"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="complexWhich">
<soap:operation soapAction="urn:sympasoap#complexWhich"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="which">
<soap:operation soapAction="urn:sympasoap#which"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="amI">
<soap:operation soapAction="urn:sympasoap#amI"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
```

```
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="add">
<soap:operation soapAction="urn:sympasoap#add"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="del">
<soap:operation soapAction="urn:sympasoap#del"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="createList">
<soap:operation soapAction="urn:sympasoap#createList"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="closeList">
<soap:operation soapAction="urn:sympasoap#closeList"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
```

```
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="review">
<soap:operation soapAction="urn:sympasoap#review"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="subscribe">
<soap:operation soapAction="urn:sympasoap#subscribe"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="signoff">
<soap:operation soapAction="urn:sympasoap#signoff"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="login">
<soap:operation soapAction="urn:sympasoap#login"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
```

```
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="casLogin">
<soap:operation soapAction="urn:sympasoap#casLogin"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="getUserEmailByCookie">
<soap:operation soapAction="urn:sympasoap#getUserEmailByCookie"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="authenticateAndRun">
<soap:operation soapAction="urn:sympasoap#authenticateAndRun"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="authenticateRemoteAppAndRun">
<soap:operation soapAction="urn:sympasoap#authenticateRemoteAppAndRun"/>
<input>
```

```
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="lists">
<soap:operation soapAction="urn:sympasoap#lists"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="complexLists">
<soap:operation soapAction="urn:sympasoap#complexLists"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
<operation name="checkCookie">
<soap:operation soapAction="urn:sympasoap#checkCookie"/>
<input>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded"
namespace="urn:sympasoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
</binding>
```

```
<!-- service part -->


<service name="SympaSOAP">
<port name="SympaPort" binding="tns:SOAP">
<soap:address location="[% conf.soap_url %]"/>
</port>
</service>

</definitions>
```

12.6 Client-side programming

Sympa is distributed with 2 sample clients written in Perl and in PHP. Sympa SOAP server has also been successfully tested with a UPortal Chanel as a Java client (using Axis). The sample PHP SOAP client has been installed on our demo server : http ://demo.sympa.org/sampleClient.php.

Depending on your programming language and the SOAP library you're using, you will either directly contact the SOAP service (as with Perl SOAP : :Lite library) or first load the WSDL description of the service (as with PHP nusoap or Java Axis). Axis is able to create a stub from the WSDL document.

The WSDL document describing the service should be fetch through WWSympa's dedicated URL : **http ://your.server/sympa/wsdl**.

Note : the **login()** function maintains a login session using HTTP cookies. If you are not able to maintain this session by analysing and sending appropriate cookies under SOAP, then you should use the **authenticateAndRun()** function that does not require cookies to authenticate.

12.6.1 Writing a Java client with Axis

First, download jakarta-axis (http ://ws.apache.org/axis/)

You must add the libraries provided with jakarta axis (v ¿1.1) to you CLASSPATH. These libraries are :

– axis.jar
– saaj.jar
– commons-discovery.jar
– commons-logging.jar
– xercesImpl.jar
– jaxrpc.jar
– xml-apis.jar
– jaas.jar
– wsdl4j.jar
– soap.jar

Next, you have to generate client java classes files from the sympa WSDL url. Use the following command :

```
java org.apache.axis.wsdl.WSDL2Java -av WSDL_URL
```

For example :

```
java org.apache.axis.wsdl.WSDL2Java -av  http://demo.sympa.org/sympa/wsdl
```

Exemple of screen output during generation of java files :

```
Parsing XML file:  http://demo.sympa.org/sympa/wsdl
Generating org/sympa/demo/sympa/msdl/ListType.java
Generating org/sympa/demo/sympa/msdl/SympaPort.java
Generating org/sympa/demo/sympa/msdl/SOAPStub.java
Generating org/sympa/demo/sympa/msdl/SympaSOAP.java
Generating org/sympa/demo/sympa/msdl/SympaSOAPLocator.java
```

If you need more information or more generated classes (to have the server-side classes or junit testcase classes for example), you can get a list of switches :

```
java org.apache.axis.wsdl.WSDL2Java -h
```

The reference page is :
http ://ws.apache.org/axis/java/reference.html

Take care of Test classes generated by axis, there are not useable as is. You have to stay connected between each test. To use junit testcases, before each soap operation tested, you must call the authenticated connexion to sympa instance.

Here is a simple Java code that invokes the generated stub to perform a casLogin() and a which() on the remote Sympa SOAP server :

```
SympaSOAP loc = new SympaSOAPLocator();
((SympaSOAPLocator)loc).setMaintainSession(true);
SympaPort tmp = loc.getSympaPort();
String _value = tmp.casLogin(_ticket);
String _cookie = tmp.checkCookie();
String[] _abonnements = tmp.which();
```

Chapitre 13

Authentication

Sympa needs to authenticate users (subscribers, owners, moderators, listmaster) on both its mail and web interface to then apply appropriate privileges (authorization process) to subsequent requested actions. *Sympa* is able to cope with multiple authentication means on the client side and when using user+password it can validate these credentials against LDAP authentication backends.

When contacted on the mail interface *Sympa* has 3 authentication levels. Lower level is to trust the `From:` SMTP header field. A higher level of authentication will require that the user confirms his/her message. The strongest supported authentication method is S/MIME (note that *Sympa* also deals with S/MIME encrypted messages).

On the *Sympa* web interface (*WWSympa*) the user can authenticate in 4 different ways (if appropriate setup has been done on *Sympa* serveur). Default authentication mean is via the user's email address and a password managed by *Sympa* itself. If an LDAP authentication backend (or multiple) has been defined, then the user can authentication with his/her LDAP uid and password. *Sympa* is also able to delegate the authentication job to a web Single SignOn system ; currently CAS (the Yale University system) or a generic SSO setup, adapted to SSO products providing an Apache module. When contacted via HTTPS, *Sympa* can make use of X509 client certificates to authenticate users.

The authorization process in *Sympa* (authorization scenarios) refers to authentication methods. The same authorization scenarios are used for both mail and web accesss ; therefore some authentication methods are considered as equivalent : mail confirmation (on the mail interface) is equivalent to password authentication (on the web interface) ; S/MIME authentication is equivalent to HTTPS with client certificate authentication. Each rule in authorization scenarios requires an authentication method (`smtp`,`md5` or `smime`) ; if the required authentication method was not used, a higher authentication mode can be requested.

13.1 S/MIME and HTTPS authentication

Chapter 27.2 (page 250) deals with *Sympa* and S/MIME signature. *Sympa* uses `OpenSSL` library to work on S/MIME messages, you need to configure some related *Sympa* parameters : 27.4.3 (page 251).

Sympa HTTPS authentication is based on Apache+mod_SSL that provide the required authentication information via CGI environment variables. You will need to edit Apache configuration to allow HTTPS access and require X509 client certificate. Here is a sample Apache configuration

```
SSLEngine on
SSLVerifyClient optional
SSLVerifyDepth  10
...
<Location /sympa>
   SSLOptions +StdEnvVars
   SetHandler fastcgi-script
</Location>
```

If you are using the SubjAltName, then you additionaly need to export the certificate data because of a mod_ssl bug. You will also need to install the textindex Crypt-OpenSSL-X509 CPAN module. Add this option to the Apache configuration file :

```
   SSLOptions +ExportCertData
```

13.2 Authentication with email address, uid or alternate email address

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 an intranet without duplicating user passwords.

This way users can indifferently 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 attempt an anonymous bind to the directory to get the user's DN, then *Sympa* will bind with the DN and the user's ldap_password in order to perform 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 *Sympa* to change your email address via the two pages : which and pref.

13.3 Generic SSO authentication

The authentication method has first been introduced to allow interraction with Shibboleth, Internet2's inter-institutional authentication system. But it should be usable with any SSO system that provides an Apache authentication module being able to protect a specified URL on the site (not the whole site). Here is a sample httpd.conf that shib-protects the associated Sympa URL :

```
...
<Location /sympa/sso_login/inqueue>
  AuthType shibboleth
  require affiliation ~ ^member@.+
</Location>
...
```

Sympa will get user attributes via environment variables. In the most simple case the SSO will provide the user email address. If not, Sympa can be configured to verify an email address provided by the user hiself or to look for the user email address in a LDAP directory (the search filter will make use of user information inherited from the SSO Apache module).

To plug a new SSO server in your Sympa server you should add a **generic_sso** paragraph (describing the SSO service) in your `auth.conf` configuration file (See 13.5.3, page 137). Once this paragraph has been added, the SSO service name will be automatically added to the web login menu.

Apart from the user email address, the SSO can provide other user attributes that *Sympa* will store in the user_table DB table (for persistancy) and make them available in the [user_attributes] structure that you can use within authorization scenarios (see 14.1, page 144) or in web templates via the [% user.attributes %] structure.

13.4 CAS-based authentication

CAS is Yale university SSO software. Sympa can use CAS authentication service.

The listmaster should define at least one or more CAS servers (**cas** paragraph) in `auth.conf`. If **non_blocking_redirection** parameter was set for a CAS server then Sympa will try a transparent login on this server when the user accesses the web interface. If one CAS server redirect the user to Sympa with a valid ticket Sympa receives a user ID from the CAS server. It then connects to the related LDAP directory to get the user email address. If no CAS server returns a valid user ID, Sympa will let the user either select a CAS server to login or perform a Sympa login.

13.5 auth.conf

The `/usr/local/sympa-os/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 wwsympa.fcgi afterwards.

The `/usr/local/sympa-os/etc/auth.conf` is organised in paragraphs. Each paragraph describes an authentication service with all required parameters to perform an authentication using this service. Current version of *Sympa* can perform authentication through LDAP directories, using an external Single Sign-On Service (like CAS or Shibboleth), or using internal user_table.

The login page contains 2 forms : the login form and the SSO. When users hit the login form, each ldap or user_table authentication paragraph is applied unless email adress input from form match the `negative_regexp` or do not match `regexp`. `negative_regexp` and `regexp` can be defined for earch ldap or user_table authentication service so administrator can block some authentication methode for class of users.

The segond form in login page contain the list of CAS server so user can choose explicitely his CAS service.

Each paragraph start with one of the keyword cas, ldap or user_table

The `/usr/local/sympa-os/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 considered as paragraphs separators. There should only be one directive per line, but their order in the paragraph is of no importance.

Example :

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


cas
base_url https://sso-cas.cru.fr
non_blocking_redirection        on
auth_service_name cas-cru
ldap_host ldap.cru.fr:389
        ldap_get_email_by_uid_filter    (uid=[uid])
ldap_timeout 7
ldap_suffix dc=cru,dc=fr
ldap_scope sub
ldap_email_attribute mail

## The URL corresponding to the service_id should be protected by the SSO (Shibboleth in t
## The URL would look like http://yourhost.yourdomain/sympa/sso_login/inqueue in the follo
generic_sso
        service_name       InQueue Federation
        service_id         inqueue
        http_header_prefix HTTP_SHIB
        email_http_header  HTTP_SHIB_EMAIL_ADDRESS

## The email address is not provided by the user home institution
generic_sso
        service_name               Shibboleth Federation
        service_id                 myfederation
        http_header_prefix         HTTP_SHIB
        netid_http_header          HTTP_SHIB_EMAIL_ADDRESS
```

```
internal_email_by_netid    1
force_email_verify         1

ldap
regexp univ-rennes1\.fr
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
use_ssl                         1
ssl_version                     sslv3
ssl_ciphers                     MEDIUM:HIGH

ldap

host ldap.univ-nancy2.fr:392,ldap1.univ-nancy2.fr:392,ldap2.univ-nancy2.fr:392
timeout 20
bind_dn                         cn=sympa,ou=people,dc=cru,dc=fr
bind_password                   sympaPASSWD
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
        authentication_info_url         http://sso.univ-nancy2.fr/


user_table
negative_regexp  ((univ-rennes1)|(univ-nancy2))\.fr
```

13.5.1 user_table paragraph

The user_table paragraph is related to sympa internal authentication by email and password. It is the simplest one the only parameters are `regexp` or `negative_regexp` which are perl regular expressions applied on a provided email address to select or block this authentication method for a subset of email addresses.

13.5.2 ldap paragraph

- `regexp` and `negative_regexp` Same as in user_table paragraph : if a provided email address (does not apply to an uid), then the regular expression will be applied to find out if this LDAP directory can be used to authenticate a subset of users.
- 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. Server replication is supported by listing several servers separated by commas.
  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 (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`
- bind_dn

  If anonymous bind is not allowed on the LDAP server, a DN and password can be used.
- bind_password

  This password is used, combined with the bind_dn above.
- get_dn_by_uid_filter

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

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

- get_dn_by_email_filter

  Defines the search filter corresponding to the email addresses (canonic and alternative).(RFC 2254 compliant). If you want to apply the filter on the user, use the variable ' [sender] '. It will 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_address ... In the previous example the canonic email is 'mail'.

– alternative_email_attribute

  The name of the attribute for the alternate email in your directory : for instance alternate_email, mailalternateaddress, ... You make a list of these attributes separated by commas.
  With this list *Sympa* creates a cookie which contains various information : 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 and subscriptions. That is to say you want to use a unique address 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.
- authentication_info_url

  Defines the URL of a document describing LDAP password management. When hitting Sympa's *Send me a password* button, LDAP users will be redirected to this URL.
- use_ssl
  If set to `1`, connection to the LDAP server will use SSL (LDAPS).
- ssl_version
  This defines the version of the SSL/TLS protocol to use. Defaults of Net : :LDAPS to `sslv2/3`, other possible values are `sslv2`, `sslv3`, and `tlsv1`.
- ssl_ciphers
  Specify which subset of cipher suites are permissible for this connection, using the standard OpenSSL string format. The default value of Net : :LDAPS for ciphers is `ALL`, which permits all ciphers, even those that don't encrypt !

13.5.3 generic_sso paragraph

- service_name
  This is the SSO service name that will be proposed to the user in the login banner menu.
- service_id
  This service ID is used as a parameter by sympa to refer to the SSO service (instead of the service name).
  A corresponding URL on the local web server should be protected by the SSO system ; this URL would look like **http ://yourhost.yourdomain/sympa/sso_login/inqueue** if the service_id is **inqueue**.
- http_header_prefix
  Sympa gets user attributes from environment variables comming from the web server. These variables are then stored in the user_table DB table for later use in authorization scenarios (in structure). Only environment variables starting with the defined prefix will kept.
- email_http_header
  This parameter defines the environment variable that will contain the authenticated user's email address.

The following parameters define how Sympa can verify the user email address, either provided by the SSO or by the user himself :

- internal_email_by_netid
  If set to 1 this parameter makes Sympa use its netidmap table to associate NetIDs to user email address.

- netid_http_header
  This parameter defines the environment variable that will contain the user's identifier. This netid will then be associated with an email address either provided by the user.
- force_email_verify
  If set to 1 this parameter makes Sympa verify the user's email address. If the email address was not provided by the authentication module, then the user is requested to provide a valid email address.

The following parameters define how Sympa can retrieve the user email address ; **these are only useful if the email_http_header entry was not defined :**

- ldap_host
  The LDAP host Sympa will connect to fetch user email. The ldap_host include the port number and it may be a comma separated list of redondant host.
- ldap_bind_dn
  The DN used to bind to this server. Anonymous bind is used if this parameter is not defined.
- ldap_bind_password
  The password used unless anonymous bind is used.
- ldap_suffix
  The LDAP suffix used when seraching user email
- ldap_scope
  The scope used when seraching user email, possible values are `sub`, `base`, and `one`.
- ldap_get_email_by_uid_filter
  The filter to perform the email search. It can refer to any environment variables inherited from the SSO module, as shown below. Example :

```
        ldap_get_email_by_uid_filter    (mail=[SSL_CLIENT_S_DN_Email])
```

- ldap_email_attribute
  The attribut name to be used as user canonical email. In the current version of sympa only the first value returned by the LDAP server is used.
- ldap_timeout
  The time out for the search.
- ldap_use_ssl
  If set to `1`, connection to the LDAP server will use SSL (LDAPS).
- ldap_ssl_version
  This defines the version of the SSL/TLS protocol to use. Defaults of Net : :LDAPS to `sslv2/3`, other possible values are `sslv2`, `sslv3`, and `tlsv1`.
- ldap_ssl_ciphers
  Specify which subset of cipher suites are permissible for this connection, using the OpenSSL string format. The default value of Net : :LDAPS for ciphers is `ALL`, which permits all ciphers, even those that don't encrypt !

13.5.4 cas paragraph

- auth_service_name
  The friendly user service name as shown by *Sympa* in the login page.

- host (OBSOLETE)
  This parameter has been replaced by **base_url** parameter
- base_url

  The base URL of the CAS server.
- non_blocking_redirection

  This parameter concern only the first access to Sympa services by a user, it activate or not the non blocking redirection to the related cas server to check automatically if the user as been previously authenticated with this CAS server. Possible values are **on** **off**, default is **on**. The redirection to CAS is use with the cgi parameter **gateway=1** that specify to CAS server to always redirect the user to the origine URL but just check if the user is logged. If active, the SSO service is effective and transparent, but in case the CAS server is out of order the access to Sympa services is impossible.
- login_uri (OBSOLETE)
  This parameter has been replaced by **login_path** parameter.
- login_path (OPTIONAL)
  The login service path
- check_uri (OBSOLETE)
  This parameter has been replaced by **service_validate_path** parameter
- service_validate_path (OPTIONAL)
  The ticket validation service path
- logout_uri (OBSOLETE)
  This parameter has been replaced by **logout_path** parameter
- logout_path (OPTIONAL)
  The logout service path
- proxy_path (OPTIONAL)
  The proxy service path, used by Sympa SOAP server only.
- proxy_validate_path (OPTIONAL)
  The proxy validate service path, used by Sympa SOAP server only.
- ldap_host
  The LDAP host Sympa will connect to fetch user email when user uid is return by CAS service. The ldap_host include the port number and it may be a comma separated list of redondant host.
- ldap_bind_dn
  The DN used to bind to this server. Anonymous bind is used if this parameter is not defined.
- ldap_bind_password
  The password used unless anonymous bind is used.
- ldap_suffix
  The LDAP suffix used when seraching user email
- ldap_scope
  The scope used when seraching user email, possible values are `sub`, `base`, and `one`.
- ldap_get_email_by_uid_filter
  The filter to perform the email search.
- ldap_email_attribute
  The attribut name to be use as user canonical email. In the current version of sympa only the first value returned by the LDAP server is used.
- ldap_timeout
  The time out for the search.

- ldap_use_ssl
  If set to `1`, connection to the LDAP server will use SSL (LDAPS).
- ldap_ssl_version
  This defines the version of the SSL/TLS protocol to use. Defaults of Net : :LDAPS to `sslv2/3`, other possible values are `sslv2`, `sslv3`, and `tlsv1`.
- ldap_ssl_ciphers
  Specify which subset of cipher suites are permissible for this connection, using the OpenSSL string format. The default value of Net : :LDAPS for ciphers is `ALL`, which permits all ciphers, even those that don't encrypt !

13.6 Sharing *WWSympa* authentication with other applications

If your are not using a web Single SignOn system 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 :
  `sympauser=<user_email>:<checksum>`
  where <user_email> is the user's complete e-mail address, and <checksum> are the 8 last bytes of the 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 referer URL when an action is performed. Here is a sample HTML anchor :
  `<A HREF="/sympa/loginrequest/referer">Login page</A>`
  You can also have your own HTML page submitting data to `wwsympa.fcgi` CGI. If you're doing so, you can set the `referer` variable to another URI. You can also set the `failure_referer` to make WWSympa redirect the client to a different URI if login fails.
- 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.

13.7 Provide a Sympa login form in another application

You can easily trigger a Sympa login from within another web page. The login form should look like this :

```
<FORM ACTION="http://listes.cru.fr/sympa" method="post">
      <input type="hidden" name="previous_action" value="arc" />
      Access web archives of list
      <select name="previous_list">
      <option value="sympa-users" >sympa-users</option>
      </select><br/>

      <input type="hidden" name="action" value="login" />
      <label for="email">email address :
      <input type="text" name="email" id="email" size="18" value="" /></label><br />
      <label for="passwd" >password :
      <input type="password" name="passwd" id="passwd" size="8" /></label> <br/>
      <input class="MainMenuLinks" type="submit" name="action_login" value="Login and acce
</FORM>
```

The example above does not only perform the login action but also redirects the user to another sympa page, a list web archives here. The `previous_action` and `previous_list` variable define the action that will be performed after the login is done.

Chapitre 14

Authorization scenarios

List parameters controlling the behavior of commands are linked to different authorization scenarios. For example : the `send private` parameter is related to the send.private scenario. There are four possible locations for a authorization scenario. When *Sympa* seeks to apply an authorization scenario, it looks first in the related list directory `/usr/local/sympa-os/expl/<list>/scenari`. If it does not find the file there, it scans the current robot configuration directory `/usr/local/sympa-os/etc/my.domain.org/scenari`, then the site's configuration directory `/usr/local/sympa-os/etc/scenari`, and finally `/usr/local/sympa-os/bin/etc/scenari`, which is the directory installed by the Makefile.

An authorization scenario is a small configuration language to describe who can perform an operation and which authentication method is requested for it. An authorization scenario is an ordered set of rules. The goal is to provide a simple and flexible way to configure authorization and required authentication method for each operation.

Each authorization 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 27.4.4, page 251).
- 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 r\'eserv\'ee au propri\'etaire avec authentification
title.es eliminacin reservada slo para el propietario, necesita autentificacin


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

14.1 rules specifications

An authorization 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/)
| search (<named_filter_file>)
                | 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 t
                | newer (<date>, <date>)    # true if first date is posterior to
                | CustomCondition::<package_name> (<var>*)

<var> ::= [email] | [sender] | [user-><user_key_word>] | [previous_email]
                  | [remote_host] | [remote_addr] | [user_attributes-><user_attr
    | [subscriber-><subscriber_key_word>] | [list-><list_key_word>] | [env-><env_
  | [conf-><conf_key_word>] | [msg_header-><smtp_key_word>] | [msg_body]
    | [msg_part->type] | [msg_part->body] | [msg_encrypted] | [is_bcc] | [curren
  | [topic-auto] | [topic-sender,] | [topic-editor] | [topic] | [topic-needed]
  | <string>

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

[sender] ::= email address of the current user (used on web or mail interface). I

[previous_email] ::= old email when changing subscription email in preference pag

[msg_encrypted] ::= set to 'smime' if the message was S/MIME encrypted

[topic-auto] ::= topic of the message if it has been automatically tagged

[topic-sender] ::= topic of the message if it has been tagged by sender
```

```
[topic-editor] ::= topic of the message if it has been tagged by editor

[topic]  ::= topic of the message

[topic-needed] ::= the message has not got any topic and message topic are required for th

/perl_regexp/ ::= a perl regular expression. Don't forget to escape special characters (^,
Check http://perldoc.perl.org/perlre.html for regular expression syntax.

<date> ::= '<date_element> [ +|- <date_element>]'

<date_element> ::= <epoch_date> | <var> | <date_expr>

<epoch_date> ::= <integer>

<date_expr> ::= <integer>y<integer>m<integer>d<integer>h<integer>min<integer>sec

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

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

<auth> ::= smtp|md5|smime

<action> ::=   do_it [,notify]
             | do_it [,quiet]
     | reject(reason=<reason_key>) [,quiet]
     | reject(tt2=<tpl_name>) [,quiet]
             | request_auth
             | owner
     | editor
     | editorkey[,quiet]
     | listmaster

<reason_key> ::= match a key in mail_tt2/authorization_reject.tt2 template corresponding t
                 an information message about the reason of the reject of the user

<tpl_name> ::= corresponding template (<tpl_name>.tt2) is send to the sender

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

<user_attributes_key_word> ::= one of the user attributes provided by the SSO system via e

<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 | total
```

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

<named_filter_file> ::= filename ending with .ldap , .sql or .txt

<package_name> ::= name of a perl package in /etc/custom_conditions/ (small lett
```

(Refer to 17.8, page 164 for date format definition)

The function to evaluate scenario is described in section 29.2.6, page 275.

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 evaluating the authorization scenario for sending messages. 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.

The difference between editor and editorkey is, that with editor the message is simply forwarded to the moderaotr. He then can forward it to the list, if he wishes. editorkey assigns a key to the message and sends it to the moderator together with the message. So the moderator can just send back the key to distribute the message. Please note, that moderation from the webinterface is only possible when using editorkey, because otherwise there is no copy of the message saved on the server.

A bunch of authorization scenarios 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 authorization scenarios for your own need. In this case, don't forget to restart *Sympa* and wwsympa because authorization scenarios are not reloaded dynamicaly.

These standard authorization scenarios are located in the `/usr/local/sympa-os/bin/etc/scenari/` directory. Default scenarios are named <command>.default.

You may also define and name your own authorization scenarios. Store them in the `/usr/local/sympa-os/etc/scenari` directory. They will not be overwritten by Sympa release. Scenarios can also be defined for a particular virtual host (using directory `/usr/local/sympa-os/etc/<robot>/scenari`) or for a list (`/usr/local/sympa-os/expl/<robot>/<list>/scenari`). **Sympa will not dynamically detect that a list config should be reloaded after a scenario has been changed on disk.**

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 authorization scenario in any list configuration file, for example :

```
subscribe rennes1
```

14.2 Named Filters

At the moment Named Filters are only used in authorization scenarios. 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, an SQL database or an flat text file, thanks to an authorization scenario.

Note that the only a subset of variable available in the scenario context are available here (including [sender] and [listname]).

14.2.1 LDAP Named Filters Definition

People are selected through an LDAP filter defined in a configuration file. This file must have the extension '.ldap'. It is stored in `/usr/local/sympa-os/etc/search_filters/`.

You must give several informations in order to create a LDAP 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. *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 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.
- bind_dn
  If anonymous bind is not allowed on the LDAP server, a DN and password can be used.
- bind_password
  This password is used, combined with the bind_dn above.

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
```

14.2.2 SQL Named Filters Definition

People are selected through an SQL filter defined in a configuration file. This file must have the extension '.sql'. It is stored in `/usr/local/sympa-os/etc/search_filters/`.

To create an SQL Named Filter, you have to configure SQL host, database and options, the same way you did it for the main Sympa database in sympa.conf. Of course you can use different database and options. Sympa will open a new Database connection to execute your statement.

Please refer to section 7.10, page 72 for a detailed explanation of each parameter.

Here, all database parameters have to be grouped in one `sql_named_filter_query` paragraph.

- db_type
  `Format : db_type mysql | SQLite | Pg | Oracle | Sybase` Database management system used. Mandatory and Case sensitive.
- db_host
  Database host name. Mandatory.
- db_name
  Name of database to query. Mandatory.
- statement
  Mandatory. The SQL statement to execute to verify authorization. This statement must returns 0 to refuse the action, or anything else to grant privileges. The `SELECT COUNT(*)...` statement is the perfect query for this parameter. The keyword in the SQL query will be replaced by the sender's email.

– Optional parameters
  Please refer to main sympa.conf section for description.
  – db_user
  – db_password
  – db_options
  – db_env
  – db_port
  – db_timeout

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

```
sql_named_filter_query
db_type         mysql
db_name         people
db_host         dbserver.rennes1.fr
db_user         sympa
db_passwd       pw_sympa_mysqluser
statement       SELECT count(*) as c FROM users WHERE mail=[sender] AND EmployeeTyp
```

14.2.3 Search Condition

The search condition is used in authorization scenarios which are defined and described in (see 14)

The syntax of this rule is :

```
search(example.ldap)      smtp,smime,md5    -> do_it
search(blacklist.txt)     smtp,smime,md5    -> do_it
```

The variable used by 'search' is the name of the LDAP Configuration file or a txt matching enumeration

+Note that *Sympa* processes maintain a cache of processed search conditions to limit access to the LDAP directory or SQL server ; each entry has a lifetime of 1 hour in the cache.

When using .txt file extention, the file is read looking for a line that match the second parameter (usually the user email address). Each line is a string where the char * can be used once to mach any block. This feature is used by the blacklist implicit scenario rule. (see **??**)

The method of authentication does not change.

14.3 scenario inclusion

Scenarios can also contain includes :

```
        subscribe
            include commonreject
            match(, /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. Note that you will need to restart Sympa processes to force reloading of list config files.

14.4 blacklist implicit rule

For each service listed in parameter `use_blacklist` (see 7.4.4), the following implicit scenario rule is added at the beginning of the scenario :

```
search(blacklist.txt)  smtp,md5,pgp,smime -> reject,quiet
```

The goal is to block message or other service request from unwanted users. The blacklist can be defined for the robot or for the list. The one at the list level is to managed by list owner or list editor via the web interface.

14.5 Custom perl package conditions

You can use a perl package of your own to evaluate a custom condition. It could be usefull if you have very complex tasks to accomplish to evaluate your condition (web services queries...). You write a perl module, place it in the CustomCondition namespace, with one verify fonction that have to return 1 to grant access, undef to throw an error, or anything else to refuse the authorization.

This perl module :
– must be placed in a subdirectoy `'custom_conditions'` of the `'etc'` directory of your sympa installation, or of a robot

– its filename must be lowercase

– must be placed in the CustomCondition namespace

– must contains one `'verify'` static fonction

– will receive all condition arguments as parameters

For example, lets write the smallest custom condition that always returns 1.

```
/home/sympa/etc/custom_conditions/yes.pm :

    #!/usr/bin/perl

    package CustomCondition::yes;

    use strict;
    use Log; # optional : we log parameters

    sub verify {
      my @args = @_;
      foreach my $arg (@args) {
        do_log ('debug3', 'arg: %s', $arg);
      }
      # I always say 'yes'
      return 1;
    }
    ## Packages must return true.
    1;
```

We can use this custom condition that way :

```
CustomCondition::yes(,,)      smtp,smime,md5    -> do_it
true()                               smtp,smime -> reject
```

Note that the `,,` are optionnal, but it's the way you can pass information to your package. Our yes.pm will print their values in the logs.

Remember that the package name has to be small letters, but the 'CustomCondition' namespace is case sensitive. If your package return undef, the sender will receive an 'internal error' mail. If it returns anything else but '1', the sender will receive a 'forbidden' error.

14.6 Hidding scenario files

Because *Sympa* is distributed with many default scenario files, you may want to hidde some of them to list owners (to make list admin menus shorter and readable). To hidde a scenario file you should create an empty file with the `:ignore` suffix. Depending on where this file has been created will make it ignored at either a global, robot or list level.

Example :

> /usr/local/sympa-os/etc/my.domain.org/scenari/send.intranetorprivate :ignore

The `intranetorprivate send` scenario will be hidden (on the web admin interface), at the my.domain.orgrobot level only.

Chapitre 15

virtual host

Sympa is designed to manage multiple distinct mailing list servers on a single host with a single Sympa installation. Sympa virtual hosts are like Apache virtual hosting. Sympa virtual host 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 redefined for each robot except general Sympa installation parameters (binary and spool location, smtp engine, antivirus plugging,...).

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

Note that the main limitation of virtual hosts in Sympa is that you cannot create 2 lists with the same name (local part) among your virtual hosts.

15.1 How to create a virtual host

You don't need to install several Sympa servers. A single `sympa.pl` daemon and one or more fastcgi servers can serve all virtual host. 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:      "| /usr/local/sympa-os/bin/queue sympa@your.virtual.domai
listmaster@your.virtual.domain: "| /usr/local/sympa-os/bin/queue listmaster@your.virtual.
```

```
bounce+*@your.virtual.domain:          "| /usr/local/sympa-os/bin/bouncequeue s
```

- 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 host. You don't need to run dedicated fascgi server for each virtual host.
  Examples :

  ```
  FastCgiServer /usr/local/sympa-os/bin/wwsympa.fcgi -processes 3 -idle-timeout 1
  .....
  <VirtualHost 195.215.92.16>
    ServerAdmin webmaster@your.virtual.domain
    DocumentRoot /var/www/your.virtual.domain
    ServerName your.virtual.domain

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

    ScriptAlias /sympa /usr/local/sympa-os/bin/wwsympa.fcgi

    Alias /static-sympa /usr/local/sympa-os/your.virtual.domain/static_content

  </VirtualHost>
  ```
- Create a `/usr/local/sympa-os/etc/your.virtual.domain/robot.conf` configuration file for the virtual host. Its format is a subset of `sympa.conf` and is described in the next section ; a sample `robot.conf` is provided.
- Create a `/usr/local/sympa-os/expl/your.virtual.domain/` directory that will contain the virtual host mailing lists directories. This directory should have the *sympa* user as its owner and must have read and write access for this user.

  ```
  # su sympa -c 'mkdir /usr/local/sympa-os/expl/your.virtual.domain'
  # chmod 750 /usr/local/sympa-os/expl/your.virtual.domain
  ```

15.2 robot.conf

A robot is named by its domain, let's say my.domain.organd defined by a directory `/usr/local/sympa-os/etc/my.domain.org`. This directory must contain at least a `robot.conf` file. This files has the same format as `/usr/local/sympa-os/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' environment variable in wwsympa.fcgi to determine the current Virtual Host. You can a path at the end of this parameter if you are running multiple virtual hosts on the same host.

  ```
  Examples: \\
  http_host  myhost.mydom
  http_host  myhost.mydom/sympa
  ```

- `host`
  This is the equivalent of the `host` sympa.conf parameter. The default for this parameter is the name of the virtual host (ie the name of the subdirectory).
- `wwsympa_url`
  The base URL of WWSympa
- `soap_url`
  The base URL of Sympa's SOAP server (if it is running ; see 12, page 111)
- `cookie_domain`
- `email`
- `title`
- `default_home`
- `create_list`
- `lang`
- `supported_lang`
- `log_smtp`
- `listmaster`
- `max_size`
- `css_path`
- `css_url`
- `static_content_path`
- `static_content_url`
- `pictures_feature`
- `pictures_max_size`
- `logo_html_definition`
- `color_0`, color_1 ... color_15
- deprecated color definition `dark_color`, `light_color`, `text_color`, `bg_color`, `error_color`, `selected_color`, `shaded_color`

These settings overwrite the equivalent global parameter defined in `/usr/local/sympa-os/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.

15.2.1 Robot customization

In order to customize the web look and feel, you may edit the CSS definition. CSS are defined in a template named css.tt2. Any robot can use static css file for making Sympa web interface faster. Then you can edit this static definition and change web style. Please refer to `css_path` `css_url`. You can also quickly introduce a logo in left top corner of all pages configuring `logo_html_definition` parameter in robot.conf file.

In addition, if needed, you can customize each virtual host using its set of templates and authorization scenarios.

`/usr/local/sympa-os/etc/my.domain.org/web_tt2/`,

`/usr/local/sympa-os/etc/my.domain.org/mail_tt2/`,
`/usr/local/sympa-os/etc/my.domain.org/scenari/` directories are searched when loading templates or scenari before searching into `/usr/local/sympa-os/etc` and `/usr/local/sympa-os/bin/etc`. This allows to define different privileges and a different GUI for a Virtual Host.

15.3 Managing multiple virtual hosts

If you are managing more than 2 virtual hosts, then you might cinsider moving all the mailing lists in the default robot to a dedicated virtual host located in the `/usr/local/sympa-os/expl/my.domain.org/` directory. The main benefit of this organisation is the ability to define default configuration elements (templates or authorization scenarios) for this robot without inheriting them within other virtual hosts.

To create such a virtual host, you need to create `/usr/local/sympa-os/expl/my.domain.org/` and `/usr/local/sympa-os/etc/my.domain.org/` directories ; customize `host`, `http_host` and `wwsympa_url` parameters in the `/usr/local/sympa-os/etc/my.domain.org/robot.conf` with the same values as the default robot (as defined in `sympa.conf` and `wwsympa.conf` files).

Chapitre 16

Interaction between *Sympa* and other applications

16.1 Soap

See 12, page 111.

16.2 RSS channel

See 11, page 107.

16.3 Sharing *WWSympa* authentication with other applications

See 13.6, page 140.

16.4 Sharing data with other applications

You may extract subscribers, owners and editors for a list from any of :
– a text file
– a Relational database

– a LDAP directory
See `user_data_source` list parameter 21.2.1, page 204.

The three tables can have more fields than the one used by *Sympa*, by defining these additional fields, they will be available from within *Sympa*'s authorization scenarios and templates (see 7.10.11, page 74 and 7.10.12, page 74).

See data inclusion file 18.7, page 172.

16.5 Subscriber count

subscriber_count

The number of subscribers of a list can be get from an external application by requesting function 'subscriber_count' on the Web interface.

Example: `http ://my.server/wws/subscriber_count/mylist`

Chapitre 17

Customizing *Sympa/WWSympa*

17.1 Template file format

Template files within *Sympa* used to be in a proprietary format that has been replaced with the TT2[1] template format.

You will find detailed documentation about the TT2 syntax on the web site : **http ://www.tt2.org**

Here are some aspects regarding templates that are specific to *Sympa* :
- References to PO catalogue are noted with the **[% loc %]** tag that may include parameters. Example: `[%|loc(list.name,list.host)%]Welcome to list %1%2[%END%]`.
- Few exceptions apart, templates cannot insert or parse a file given its full or relative path, for security reason. Only the file name should be provided ; the TT2 parser will then use the INCLUDE_PATH provided by *Sympa* to find the relevant file to insert/parse.
- The **qencode** filter should be used if a template includes SMTP header fields that should be Q-encoded. Example: `[% FILTER qencode %]Message à modérer[%END%]`
- You can write different versions of a template file in different language, each of them being located in a subdirectory of the **tt2** directory. Example: `/mail_tt2/fr_FR/helpfile.tt2`

[1] http ://www.tt2.org

17.2 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 TT2 template format ; every file has a **.tt2** extension. See 17.1, page 159.

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. `/usr/local/sympa-os/expl/<list>/mail_tt2/<lang>/<action>.tt2`.
2. `/usr/local/sympa-os/expl/<list>/mail_tt2/<action>.tt2`.
3. `/usr/local/sympa-os/etc/my.domain.org/mail_tt2/<lang>/<action>.tt2`.
4. `/usr/local/sympa-os/etc/my.domain.org/mail_tt2/<action>.tt2`.
5. `/usr/local/sympa-os/etc/mail_tt2/<lang>/<action>.tt2`.
6. `/usr/local/sympa-os/etc/mail_tt2/<action>.tt2`.
7. `/usr/local/sympa-os/bin/etc/mail_tt2/<lang>/<action>.tt2`.
8. `/usr/local/sympa-os/bin/etc/mail_tt2/<action>.tt2`.

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

17.2.1 helpfile.tt2

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

17.2.2 lists.tt2

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 = lists %]
[% l.key %]@[% l.value.host %] : [% l.value.subject %]

[% END %]
```

17.2.3 global_remind.tt2

This file is sent in response to a `REMIND *` command. (see 28.2, page 258) 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 = lists %]

 [% l %] : [% conf.wwsympa\_url \%]/info/[% l %]

[% END %]

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

17.2.4 your_infected_msg.tt2

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

17.3 Web template files

You may define your own web template files, different from the standard ones. *WW-Sympa* 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 `/usr/local/sympa-os/expl/mylist/web_tt2` directory ; your site web templates in `~/usr/local/sympa-os/etc/web_tt2` directory.

Note that web colors are defined in *Sympa*'s main Makefile (see 3.3, page 31).

17.4 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.

If you are willing to provide Sympa into your native language, please check the **translation howto** (`http ://www.sympa.org/howtotranslate.html`) ;

17.4.1 *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 `/usr/local/sympa-os/locale` directory.

To tell *Sympa* to use a particular message catalog, you can should set the `lang` parameter in `sympa.conf`.

17.4.2 List internationalization

The `lang` list parameter defines the language for a list. It is currently used by *WW-Sympa* 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.

17.4.3 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.

17.5 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 (This parameter is different from `msg_topic` used to tag list messages)

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

```
<topic1_name>
title <topic1 title>
title.fr <topic french title>
visibility <topic1 visibility>
....
<topicn_name/subtopic_name>
title <topicn title>
title.de <topicn german 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 authorization 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.

17.6 Authorization scenarios

See 14, page 143.

17.7 Loop detection

Sympa uses multiple heuristics 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

17.8 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 `/usr/local/sympa-os/bin/etc/global_task_models` and `/usr/local/sympa-os/bin/etc/list_task_models` directories. Others can be designed by the listmaster.

A task is global or related to a list.

17.8.1 List task creation

You define in the list config file the model and the version you want to use (see 21.3.5, page 213). Then the task manager daemon will automatically create the task by looking for the appropriate model file in different directories in the following order :

1. `/usr/local/sympa-os/expl/<list name>/`
2. `/usr/local/sympa-os/etc/list_task_models/`
3. `/usr/local/sympa-os/bin/etc/list_task_models/`

See also 18.10, page 176, to know more about standard list models provided with *Sympa*.

17.8.2 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. `/usr/local/sympa-os/etc/global_task_models/`
2. `/usr/local/sympa-os/bin/etc/global_task_models/`

17.8.3 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 18.8, see page 173) 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 Authorization Scenarios 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.
- purge_orphan_bounces()
  Clean bounces by removing unsubscribed-users archives.
- eval_bouncers()
  Evaluate all bouncing users of all list and give them a score from 0 to 100. (0 = no bounces for this user, 100 is for users who should be removed).
- process_bouncers()
  Execute actions defined in list configuration on each bouncing users, according to their score.

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

17.8.4 Model file examples

- remind.annual.task

- expire.annual.task

- crl_update.daily.task

```
title.gettext daily update of the certificate revocation list

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

Chapitre 18

Mailing list definition

This chapter describes what a mailing list is made of within Sympa environment.

18.1 Mail aliases

See list aliases section, 18.1, page 169)

18.2 List configuration file

The configuration file for the `mylist` list is named `/usr/local/sympa-os/expl/my.domain.org/mylist/config` (or `/usr/local/sympa-os/expl/mylist/config` if no virtual host is defined). *Sympa* reloads it into memory whenever this file has changed on disk. The file can either be edited via the web interface or directly via your favourite text editor.

If you have set the `cache_list_config` sympa.conf parameter (see 7.9.1, page 71), a binary version of the config (`/usr/local/sympa-os/expl/my.domain.org/mylist/config.bin` is maintained to allow a faster restart of daemons (this is especialy usefull for sites managing lots of lists).

Be careful to provide read access for *Sympa* user to this file !

You will find a few configuration files in the `sample` directory.

List configuration parameters are described in the list creation section, 21, page 199.

18.3 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
```

18.4 Subscribers file

Be carefull : Since version 3.3.6 of *Sympa*, a RDBMS is required for internal data storage. Flat file should not be use anymore except for testing purpose. *Sympa* will not use this file if the list is configured with `include` or `database` `user_data_source`.

The `/usr/local/sympa-os/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. In this mode subscription to message topics is available. See the `SET LISTNAME SUMMARY` (28.1, page 256), the `SET LISTNAME NOMAIL` command (28.1, page 257), and the `digest` parameter (21.4.9, page 220).
- `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 (28.1, page 257) for details.

18.5 Info file

`/usr/local/sympa-os/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.

18.6 Homepage file

`/usr/local/sympa-os/expl/mylist/homepage` is the HTML text on the *WW-Sympa* info page for the list.

18.7 Data inclusion file

Sympa will use these files only if the list is configured in `include2 user_data_source` mode. Every file has the .incl extension. More over, these files must be declared in paragraphs `owner_include` or `editor_inlude` in the list configuration file without the .incl extension (see 21, page 199). This files can be template file.

Sympa looks for them in the following order :

1. `/usr/local/sympa-os/expl/mylist/data_sources/<file>.incl`.
2. `/usr/local/sympa-os/etc/data_sources/<file>.incl`.
3. `/usr/local/sympa-os/etc/my.domain.org/data_sources/<file>.incl`.

These files are used by Sympa to load administrative data in a relational database : Owners or editors are defined *intensively* (definition of criteria owners or editors must satisfy). Includes can be performed by extracting e-mail addresses using an SQL or LDAP query, or by including other mailing lists.

A data inclusion file is composed of paragraphs separated by blank lines and introduced by a keyword. Valid paragraphs are `include_file`, `include_list`, `include_remote_sympa_list`, `include_sql_query` and `include_ldap_query`. They are described in the list configuration parameters chapitre, 21, page 199.

When this file is a template, used variables are array elements (`param` array). This array is instantiated by values contained in the subparameter `source_parameter` of `owner_include` or `editor_inlude`.

Example :

– in the list configuration file :

```
owner_include
source myfile
source_parameters mysql,rennes1,stduser,mysecret,studentbody,student
```

– in myfile.incl :

```
include_sql_query
db_type [% param.0 %]
host sqlserv.admin.univ-[% param.1 %].fr
user [% param.2 %]
passwd [% param.3 %]
        db_name [% param.4 %]
sql_query SELECT DISTINCT email FROM [% param.5 %]
```

– resulting data inclusion file :

```
include_sql_query
db_type mysql
host sqlserv.admin.univ-rennes1.fr
        user stduser
        passwd mysecret
        db_name studentbody
        sql_query SELECT DISTINCT email FROM student
```

18.8 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 .tt2 extension. See 17.1, page 159.

Sympa looks for these files in the following order :

1. `/usr/local/sympa-os/expl/mylist/mail_tt2/<file>.tt2`
2. `/usr/local/sympa-os/etc/mail_tt2/<file>.tt2`.
3. `/usr/local/sympa-os/bin/etc/mail_tt2/<file>.tt2`.

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 [% INSERT %] directive.

Example :

```
Dear [% user.email %],

Welcome to list [% list.name %]@[% list.host %].

Presentation of the list :
[% INSERT 'info' %]

The owners of [% list.name %] are :
[% FOREACH ow = list.owner %]
   [% ow.value.gecos %] <[% ow.value.email %]>
[% END %]
```

18.8.1 welcome.tt2

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

18.8.2 bye.tt2

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

18.8.3 removed.tt2

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

18.8.4 reject.tt2

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.

18.8.5 invite.tt2

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

18.8.6 remind.tt2

This file contains a message sent to each subscriber when one of the list owners sends the `REMIND` command (see 28.2, page 258).

18.8.7 summary.tt2

Template for summaries (reception mode close to digest), see 28.1, page 256.

18.8.8 list_aliases.tt2

Template that defines list mail alises. It is used by the alias_manager script.

18.9 Stats file

`/usr/local/sympa-os/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
– Last update date (epoch format) of the subscribers cache in DB, used by lists in **include2** mode only

18.10 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 17.8, page 164.

18.10.1 remind.annual.task

Every year *Sympa* will send a message (the template `remind.tt2`) to all subscribers of the list to remind them of their subscription.

18.10.2 expire.annual.task

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

18.11 Message header and footer

You may create `/usr/local/sympa-os/expl/mylist/message.header` and `/usr/local/sympa-os/expl/mylist/message.footer` files. Their content is added, respectively at the beginning and at the end of each message before the distribution process. You may also include the content-type of the appended part (when `footer_type` list parameter s set to **mime**) by renaming the files to `message.header.mime` and `message.footer.mime`.

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).

Under certain circumstances, Sympa will NOT add headers/footers, here is its algorythm :

```
if message is not multipart/signed
        if footer_type==append
        if message is text/plain
       append header/footer to it
else if message is multipart AND first part is text/plain
       append header/footer to first part

        if footer_type==mime
        if message is not multipart/alternative
       add header/footer as a new MIME part
```

18.11.1 Archive directory

The /usr/local/sympa-os/expl/mylist/archives/ directory contains the archived messages for lists which are archived ; see 21.6.1, page 226. The files are named in accordance with the archiving frequency defined by the archive parameter.

Chapitre 19

List creation, edition and removal

The list creation can be done by two ways, according to listmaster needs :
– instanciation family to create and manage large number of related lists. In this case, lists are linked to their family all along their life (Moreover you can let sympa automatically create lists when needed. See 20.3, page 194).
– command line creation of individual list with `sympa.pl` or on the Web interface according to privileges defined by listmasters. Here lists are free from their model creation.

Management of mailing lists by list owners is usually done via the Web interface : 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 keeps logs of the creation and all modifications to a list as part of the list's `config` file (old configuration files are archived). A complete installation requires some careful planning, although default values should be acceptable for most sites.

19.1 List creation

Mailing lists can have many different uses. *Sympa* offers a wide choice of parameters to adapt a list behavior to different situations. Users might have difficulty selecting all the correct parameters to make the list configuration, so instead of selecting each parameters, list configuration is made with a list profile. This is an almost complete list configuration, but with a number of unspecified fields (such as owner e-mail) to be replaced by *Sympa* at list creation time. It is easy to create new list templates by modifying existing ones. (Contributions to the distribution are welcome...)

19.1.1 Data for list creation

To create a list, some data concerning list parameters are required :
- **listname** : name of the list,
- **subject** : subject of the list (a short description),
- **owner(s)** : by static definition and/or dynamic definition. In case of static definition, the parameter **owner** and its subparameter **email** are required. For dynamic definition, the parameter **owner_include** and its subparameter **source** are required, indicating source file of data inclusion.
- **list creation template** : the typical list profile.

Moreover of these required data, provided values are assigned to vars being in the list creation template. Then the result is the list configuration file :

On the Web interface, these data are given by the list creator in the web form. On command line these data are given by an xml file.

19.1.2 XML file format

The xml file provides information on :

- the list name,
- values to assign vars in the list creation template
- the list description in order to be written in the list file info
- the name of the list creation template (only for list creation on command line with sympa.pl, in a family context, the template is specified by the family name)

Here is an example of XML document that you can map with the following example of list creation template. :

```
<?xml version="1.0" ?>
<list>
<listname>example</listname>
   <type>my_profile</type>
   <subject>a list example</subject>
   <description/>
   <status>open</status>
   <shared_edit>editor</shared_edit>
     <shared_read>private</shared_read>
<language>fr</language>
<owner multiple="1">
   <email>serge.aumont@cru.fr</email>
   <gecos>C.R.U.</gecos>
</owner>
<owner multiple="1">
   <email>olivier.salaun@cru.fr</email>
</owner>
```

```
<owner_include multiple="1">
   <source>my_file</source>
</owner_include>
<sql>
   <type>oracle</type>
   <host>sqlserv.admin.univ-x.fr</host>
   <user>stdutilisateur</user>
   <pwd>monsecret</pwd>
   <name>les_etudiants</name>
   <query>SELECT DISTINCT email FROM etudiant</query>
</sql>
</list>
```

```
subject [% subject %]

status [% status %]

[% IF topic %]
topics [% topic %]

[% END %]
visibility noconceal

send privateoreditorkey

Web_archive
  access public

subscribe open_notify

shared_doc
  d_edit [% shared_edit %]
  d_read [% shared_read %]

lang [% language %]

[% FOREACH o = owner %]
owner
  email [% o.email %]
  profile privileged
  [% IF o.gecos %]
  gecos [% o.gecos %]
  [% END %]

[% END %]
[% IF moderator %]
   [% FOREACH m = moderator %]
editor
```

```
    email [% m.email %]

     [% END %]
  [% END %]

  [% IF sql %]
  include_sql_query
    db_type [% sql.type %]
    host [% sql.host %]
    user [% sql.user %]
    passwd [% sql.pwd %]
    db_name [% sql.name %]
    sql_query [% sql.query %]

  [% END %]
  ttl 360
```

The XML file format should comply with the following rules :
- The root element is `<list>`
- One XML element is mandatory : `<listname>` contains the name of the list. That not excludes mandatory parameters for list creation (`listname, subject,owner.email and/or owner_include.source`).
- `<type>` : this element contains the name of template list creation, it is used for list creation on command line with sympa.pl. In a family context, this element is no used.
- `<description>` : the text contained in this element is written in list `info` file(it can be a CDATA section).
- For other elements, its name is the name of the var to assign in the list creation template.
- Each element concerning multiple parameters must have the `multiple` attribute set to "1", example : `<owner multiple=''1''>`.
- For composed and multiple parameters, sub-elements are used. Example for `owner` parameter : `<email>` and `<gecos>` elements are contained in the `<owner>`element. An element can only have homogeneous content.
- A list requires at least one owner, defined in the XML input file with one of the following elements :
  - `<owner multiple=''1''> <email> ... </email> </owner>`
  - `<owner_include multiple=''1''> <source> ... </source> </owner_include>`

19.2 List families

See chapter 20, page 187

19.3 List creation on command line with `sympa.pl`

This way to create lists is independent of family.

Here is a sample command to create one list :.

> sympa.pl –create_list –robot my.domain.org–input_file /path/to/my_file.xml

The list is created under the `my_robot` robot and the list is described in the file `my_file.xml`. The XML file is described before, see 19.1.2, page 180.

By default, the status of the created list is `open`.

typical list profile (list template creation)

The list creator has to choose a profile for the list and put its name in the XML element `<type>`.

List profiles are stored in `/usr/local/sympa-os/etc/create_list_templates` or in `/usr/local/sympa-os/bin/etc/create_list_templates` (default of distrib).

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 `/usr/local/sympa-os/etc/create_list_templates` and `/usr/local/sympa-os/bin/etc/create_list_templates` directory, then the local profile will be used by WWSympa.

19.4 Creating and editing mailing using the web

The management of mailing lists 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.

19.4.1 List creation on the Web interface

Listmasters are responsible for validating new mailing lists and, depending on the configuration chosen, might be the only ones who can fill out the create list form.The listmaster is defined in `sympa.conf` and others are defined at the virtual host level. By default, any authenticated user can request a list creation but newly created are then validated by the listmaster.

List rejection message and list creation notification message are both templates that you can customize (`list_rejected.tt2` and `list_created.tt2`).

19.4.2 Who can create lists on the Web interface

This is defined by the `create_list` sympa.conf parameter (see 7.1.19, page 54). This parameter refers to a **create_list** authorization scenario. It will determine if the *create list* button is displayed and if it requires a listmaster confirmation.

The authorization scenario can accept any condition concerning the [sender] (i.e. WW-Sympa user), and it returns `reject`, `do_it` or `listmaster` as an action.

Only in cases where a user is authorized by the create_list authorization 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.

19.4.3 typical list profile and Web interface

As on command line creation, the list creator has to choose a list profile and to fill in the owner's e-mail and the list subject together with a short description. But in this case, you don't need any XML file. Concerning these typical list profiles, they are described before, see 19.3, page 183. You can check available profile. On the Web interface, another way to control publicly available profiles is to edit the `create_list.conf` file (the default for this file is in the `/usr/local/sympa-os/bin/etc` directory, and you may create your own customized version in `/usr/local/sympa-os/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
```

19.4.4 List edition

For each parameter, you may specify (via the `/usr/local/sympa-os/etc/edit_list.conf` configuration file) who has the right to edit the parameter concerned ; the default `/usr/local/sympa-os/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 a 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.

Concerning list edition in a family context, see 20.2.8, page 194

19.5 Removing a list

You can remove a list either from the command line or using the web interface.

`sympa.pl` provides an option to remove a mailing list, see the example below :

```
sympa.pl –remove_list=mylist@mydomain
```

Privileged owners can remove a mailing list through the list admin part of the web interface. Removing the mailing list consists in removing its subscribers from the database and setting its status to *closed*.Once removed, the list can still be restored by the listmaster ; list members are saved in a `subscribers.closed.dump` file.

Chapitre 20

Lists Families

A list can have from three parameters to many tens of them. Some listmasters need to create a set of lists that have the same profile. In order to simplify the apprehension of these parameters, list families define a lists typology. Families provide a new level for defaults : in the past, defaults in Sympa were global and most sites using Sympa needed multiple defaults for different group of lists. Moreover families allow listmaster to delegate a part of configuration list to owners, in a controlled way according to family properties. Distribution will provide defaults families.

20.1 Family concept

A family provides a model for all of its lists. It is specified by the following characteristics :

– a list creation template providing a common profile for each list configuration file.
– an degree of independence between the lists and the family : list parameters edition rights and constraints on these parameters can be *free* (no constraint), *controlled* (a set of available values is defined for these parameters) or *fixed* (the value for the parameter is imposed by the family). That prevents lists from diverging from the original and it allows list owner customizations in a controlled way.
– a filiation kept between lists and family all along the list life : family modifications are applied on lists while keeping listowners customizations.

Here is a list of operation performed on a family :

– definition : definition of the list creation template, the degree of independence and family customizations.
– instantiation : lists creation or modifications of existing lists while respecting family properties. The set of data defining the lists is an XML document.

– modification : modification of family properties. The modification is effective at the next instantiation time, that have consequences on every list.
– closure : closure of each list.
– adding one list to a family.
– closing one family list.
– modifying one family list.

20.2 Using family

20.2.1 Definition

Families can be defined at the robot level, at the site level or on the distribution level (where default families are provided). So, you have to create a sub directory named after the family's name in a `families` directory :

Examples :

```
/home/sympa/etc/families/my_family
/home/sympa/etc/my_robot/families/my_family
```

In this directory you must provide these files :
– `config.tt2` (mandatory)
– `param_constraint.conf` (mandatory)
– `edit_list.conf`
– customizable files

config.tt2

This is a list creation template, this file is mandatory. It provides default values for parameters. This file is an almost complete list configuration, with a number of missing fields (such as owner e-mail) to be replaced by data obtained at the time of family instantiation. It is easy to create new list templates by modifying existing ones. See 18.8, page 173 and 17.1, page 159.

Example :

```
subject [% subject %]

status [% status %]

[% IF topic %]
topics [% topic %]

[% END %]
```

```
visibility noconceal

send privateoreditorkey

web_archive
  access public

subscribe open_notify

shared_doc
  d_edit [% shared_edit %]
  d_read [% shared_read %]

lang [% language %]

[% FOREACH o = owner %]
owner
  email [% o.email %]
  profile privileged
  [% IF o.gecos %]
  gecos [% o.gecos %]
  [% END %]

[% END %]
[% IF moderator %]
   [% FOREACH m = moderator %]
editor
  email [% m.email %]

   [% END %]
[% END %]

[% IF sql %]
include_sql_query
  db_type [% sql.type %]
  host [% sql.host %]
  user [% sql.user %]
  passwd [% sql.pwd %]
  db_name [% sql.name %]
  sql_query [% sql.query %]

[% END %]
ttl 360
```

param_constraint.conf

This file is obligatory. It defines constraints on parameters. There are three kind of constraints :
- *free* parameters : no constraint on these parameters, they are not written in the `param_constraint.conf` file.
- *controlled* parameters : these parameters must select their values in a set of available values indicated in the `param_constraint.conf` file.
- *fixed* parameters : these parameters must have the imposed value indicated in the `param_constraint.conf` file.

The parameters constraints will be checked at every list loading.

WARNING : Some parameters cannot be constrained, they are : `msg_topic.keywords` (see 21.4.13, page 221),`owner_include.source_parameter` (see 21.1.6, page 202), `editor_include.source_parameter` (see 21.1.2, page 200). About `digest` parameter (see 21.4.9, page 220) , just days can be constrained.

Example :

```
lang                fr,us
archive.period      days,week,month
visibility          conceal,noconceal
shared_doc.d_read   public
shared_doc.d_edit   editor
```

edit_list.conf

This is an optional file. It defines which parameters/files are editable by owners. See 19.4.4, page 184. If the family does not have this file, *Sympa* will look for the one defined on robot level, server site level or distribution level. (This file already exists without family context)
Notes that by default parameter family_name is not writable, you should not change this edition right.

customizable files

Families provides a new level of customization for scenarios (see 14, page 143), templates for service messages (see 17.2, page 160) and templates for web pages (see 17.3 , page 162). *Sympa* looks for these files in the following level order : list, family, robot, server site or distribution.

Example of custom hierarchy :

```
/usr/local/sympa-os/etc/families/myfamily/mail_tt2/
/usr/local/sympa-os/etc/families/myfamily/mail_tt2/bye.tt2
/usr/local/sympa-os/etc/families/myfamily/mail_tt2/welcome.tt2
```

20.2.2 Instantiation

Instantiation permits to generate lists.You must provide an XML file that is composed of lists description, the root element is *family* and is only composed of *list* elements. List elements are described in section 19.1.2, page 180. Each list is described by the set of values for affectation list parameters.

Here is an sample command to instantiate a family :

```
sympa.pl --instantiate\_family my_family --robot \samplerobot --input\_file /path/to/my\_f
```

This means lists that belong to family `my_family` will be created under the robot `my_robot` and these lists are described in the file `my_file.xml`. Sympa will split this file into several xml files describing lists. Each list XML file is put in each list directory.

Example :

```
<?xml version="1.0" ?>
<family>
  <list>
    <listname>liste1</listname>
    <subject>a list example</subject>
    <description/>
    <status>open</status>
    <shared_edit>editor</shared_edit>
    <shared_read>private</shared_read>
    <language>fr</language>
    <owner multiple="1">
      <email>serge.aumont@cru.fr</email>
      <gecos>C.R.U.</gecos>
    </owner>
    <owner multiple="1">
      <email>olivier.salaun@cru.fr</email>
    </owner>
    <owner_include multiple="1">
      <source>my_file</source>
    </owner_include>
    <sql>
      <type>oracle</type>
      <host>sqlserv.admin.univ-x.fr</host>
      <user>stdutilisateur</user>
      <pwd>monsecret</pwd>
      <name>les_etudiants</name>
      <query>SELECT DISTINCT email FROM etudiant</query>
    </sql>
  </list>
  <list>
    <listname>liste2</listname>
    <subject>a list example</subject>
    <description/>
```

```
        <status>open</status>
        <shared_edit>editor</shared_edit>
        <shared_read>private</shared_read>
        <language>fr</language>
        <owner multiple="1">
          <email>serge.aumont@cru.fr</email>
          <gecos>C.R.U.</gecos>
        </owner>
        <owner multiple="1">
          <email>olivier.salaun@cru.fr</email>
        </owner>
        <owner_include multiple="1">
          <source>my_file</source>
        </owner_include>
        <sql>
          <type>oracle</type>
          <host>sqlserv.admin.univ-x.fr</host>
          <user>stdutilisateur</user>
          <pwd>monsecret</pwd>
          <name>les_etudiants</name>
          <query>SELECT DISTINCT email FROM etudiant</query>
        </sql>
      </list>
        ...
    </family>
```

Each instantiation describes lists. Compared to the previous instantiation, there are three cases :
- lists creation : new lists described by the new instantiation
- lists modification : lists already existing but possibly changed because of changed parameters values in the XML file or because of changed family's properties.
- lists removal : lists nomore described by the new instantiation. In this case, the listmaster must valid his choice on command line. If the list is removed, it is set in status `family_closed`, or if the list is recovered, the list XML file from the previous instantiation is got back to go on as a list modification then.

After list creation or modification, parameters constraints are checked :
- *fixed* parameter : the value must be the one imposed.
- *controlled* parameter : the value must be one of the set of available values.
- *free* parameter : there is no checking.

diagram

In case of modification (see diagram), allowed customizations can be preserved :
- (1) : for every modified parameters (via Web interface), noted in the `config_changes` file, values can be collected in the old list configuration file, according to new family properties :
  - *fixed* parameter : the value is not collected.
  - *controlled* parameter : the value is collected only if constraints are respected.
  - *free* parameter : the value is collected.
- (2) : a new list configuration file is made with the new family properties
- (3) : collected values are set in the new list configuration file.

Notes :
- For each list problem (as family file error, error parameter constraint, error instanciation ...), the list is set in status `error_config` and the listmaster is notified. He will have to do necessary to put list in use.
- For each list closing in family context, the list is set in status `family_closed` and the owner is notified.
- For each overwritten list customization, the owner is notified.

20.2.3 Modification

To modify a family, you have to edit family files manually. The modification will be effective while the next instanciation.
WARNING : The family modification must be done just before an instantiation. If it is not, alive lists wouldn't respect new family properties and they would be set in status error_config immediately.

20.2.4 Closure

Closes every list (installed under the indicated robot) of this family : lists status are set to `family_closed`, aliases are removed and subscribers are removed from DB. (a dump is created in the list directory to allow restoration of the list).

Here is a sample command to close a family :

```
sympa.pl --close_family my_family --robot \samplerobot
```

20.2.5 Adding one list

Adds a list to the family without instantiate all the family. The list is created as if it was created during an instantiation, under the indicated robot. The XML file describes the list and the root element is `<list>`. List elements are described in section 19.3, page 183.

Here is a sample command to add a list to a family :

```
sympa.pl --add\_list my\_family --robot \samplerobot  --input\_file /path/to/my\_file.xml
```

20.2.6 Removing one list

Closes the list installed under the indicated robot : the list status is set to `family_closed`, aliases are removed and subscribers are removed from DB. (a dump is created in the list directory to allow restoring the list).

Here is a sample command to close a list family (same as an orphan list) :

```
sympa.pl --close_list my_list@\samplerobot
```

20.2.7 Modifying one list

Modifies a family list without instantiating the whole family. The list (installed under the indicated robot) is modified as if it was modified during an instantiation. The XML file describes the list and the root element is `<list>`. List elements are described in section 19.3, page 183.

Here is a sample command to modify a list to a family :

```
sympa.pl --modify\_list my\_family --robot \samplerobot --input\_file /path/to/
```

20.2.8 List parameters edition in a family context

According to file `edit_list.conf`, edition rights are controlled. See 19.4.4, page 184. But in a family context, constraints parameters are added to edition right as it is summarized in this array :

array

Note : In order to preserve list customization for instantiation, every modified parameter (via the Web interface) is noted in the `config_changes` file.

20.3 Automatic list creation

You can benefit from the family concept to let Sympa automatically create lists for you. Let us suppose that you want to open a list according to specified criteria (age, geographical site...) within your organization. Maybe that would result in too many lists, and many of them would never be used.

Automatic list creation allows you to define those potential lists through family parameters, but they won't be created yet. The mailing list creation is trigerred when Sympa receives a message addressed to this list.

To enable automatic list creation you'll have to :
– Configure your MTA to queue messages for these lists in an appropriate spool
– Define a family associated to such lists
– Configure Sympa to enable the feature

20.3.1 Configuring your MTA

To do so, we have to configure our MTA for it to add a custom header field to the message. The easiest way is to customize your aliases manager, so mails for automatic lists aren't delivered to the normal `queue` program, but to the `familyqueue` dedicated one. For example, you can decide that the name of those lists will start with the `auto-` pattern, so you can process them separately from other lists you are hosting.

`familyqueue` expects 2 arguments : the list name and the family name (whereas the `queue` program only expects the list address).

Let's start with a use case : we need to communicate to groups of co-workers, depending on their age and their occupation. We decide that, for example, if I need to write to all CTOs who are fifty years old, I will use the auto-cto.50@lists.domain.com mailing list. The occupation and age informations are stored in our ldap directory (but of course we could use any Sympa data source : sql, files...). We will create the age-occupation family.

First of all we configure our MTA to deliver mail to `'auto-*'` to `familyqueue` for the `age-occupation` family.

```
/etc/postfix/main.cf
    ...
    transport_maps = regexp:/etc/postfix/transport_regexp

/etc/postfix/transport_regexp
    /^.*+owner\@lists\.domain\.com$/      sympabounce:
    /^auto-.*\@lists\.domain\.com$/       sympafamily:
    /^.*\@lists\.domain\.com$/            sympa:

/etc/postfix/master.cf
    sympa     unix  -       n       n       -       -       pipe
      flags=R user=sympa argv=/usr/local/sympa-os/bin/queue ${recipient}
    sympabounce  unix  -       n       n       -       -       pipe
      flags=R user=sympa argv=/usr/local/sympa-os/bin/bouncequeue ${user}
    sympafamily  unix  -       n       n       -       -       pipe
      flags=R user=sympa argv=/usr/local/sympa-os/bin/familyqueue ${user} age-occupation
```

A mail addressed to *auto-cto.50@lists.domain.com* will be queued to the `/usr/local/sympa-os/spool/automatic` spool, defined by the `queueautomatic` `sympa.conf` parameter (see 7.6.11, page 63). The mail will first be processed by an instance of `sympa.pl` process dedicated to automatic list creation, then the mail will be sent to the newly created mailing list.

20.3.2 Defining the list family

We need to create the appropriate `etc/families/age-occupation/config.tt2`. All the magic comes from the TT2 language capabilities. We define on-the-fly the LDAP source, thanks to TT2 macros.

```
/home/sympa/etc/families/age-occupation/config.tt2
    ...
    user_data_source include2

    [%
    occupations = {
        cto = { title=>"chief technical officer", abbr=>"CHIEF TECH OFF" },
        coo = { title=>"chief operating officer", abbr=>"CHIEF OPER OFF" },
        cio = { title=>"chief information officer", abbr=>"CHIEF INFO OFF" },
    }
    nemes = listname.split('-');
    THROW autofamily "SYNTAX ERROR : listname must begin with 'auto-' " IF (neme
    tokens = nemes.1.split('\.');
    THROW autofamily "SYNTAX ERROR : wrong listname syntax" IF (tokens.size != 2
    age = tokens.1 div 10;
    %]

    custom_subject [[% occupations.${tokens.0}.abbr %] OF [% tokens.1 %]]

    subject Every [% tokens.1 %] years old [% occupations.${tokens.0}.title %]

    include_ldap_query
    attrs mail
    filter (&(objectClass=inetOrgPerson)(employeeType=[% occupations.${tokens.0}
    name ldap
    port 389
    host ldap.domain.com
    passwd ldap_passwd
    suffix dc=domain,dc=com
    timeout 30
    user cn=root,dc=domain,dc=com
    scope sub
    select all
```

The main variable you get is the name of the current mailing list via the **listname** variable as used in the example above.

20.3.3 Configuring Sympa

Now we need to enable automatic list creation in Sympa. To do so, we have to
- set the `automatic_list_feature` parameter to on and define who can create automatic lists via the `automatic_list_creation` (points to an automatic_list_creation scenario).
- set the `queueautomatic` `sympa.conf` parameter to the spool location where we want these messages to be stored (it has to be different from the `/usr/local/sympa-os/spool/msg` spool).

You can make Sympa delete automatic lists that were created with zero list members ; to do so you shoukd set the `automatic_list_removal` parameter to `if_empty`.

```
/home/sympa/etc/sympa.conf
    ...
    automatic_list_feature  on
    automatic_list_creation public
    queueautomatic          /usr/local/sympa-os/spool/automatic
    automatic_list_removal    if_empty
```

While writing your own automatic_list_creation scenarios, be aware that :
- when the scenario is evaluated, the list is not yet created ; therefore you can't use the list-related variables.
- You can only use 'smtp' and 'smime' authentication method in scenario rules (You cannot request the md5 challenge). Moreover only `do_it` and `reject` actions are available.

Now you can send message to auto-cio.40 or auto-cto.50, and the lists will be created on the fly.

You will receive an 'unknown list' error if either the syntax is incorrect or the number of subscriber is zero.

Chapitre 21

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` (or `owner_include`) and `subject`. All other parameters have a default value.

> *keyword value*

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

21.1 List description

21.1.1 editor

The `config` file contains one `editor` paragraph per moderator (or editor). It concerns static editor definition. For dynamic definition and more information about editors see 21.1.2, page 200.

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` parameter (see 21.3.8, page 214) 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 21.1.5, page 201), even when several moderators are defined.

21.1.2 editor_include

The `config` file contains one `editor_include` paragraph per data inclusion file (see 18.7, page 172). It concerns dynamic editor definition : inclusion of external data. For static editor definition and more information about moderation see 21.1.1, page 199.

Example :

```
editor_include
reception mail
source myfile
source_parameters a,b,c
```

The syntax of this directive is the same as that of the `owner_include` parameter (see 21.1.6, page 202), even when several moderators are defined.

21.1.3 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 `/usr/local/sympa-os/etc/sympa.conf`.

21.1.4 lang

(Default value: `lang robot parameter`)

Example :

```
lang en_US
```

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

See 17.4, page 162 for available languages.

21.1.5 owner

The `config` file contains one `owner` paragraph per owner. It concerns static owner definition. For dynamic definition see 21.1.6, page 202.

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 28.2, page 258) and `DELETE` (see 28.2, page 258) commands on behalf of other users.

When the `subscribe` parameter (see 21.3.1, page 211) 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.

21.1.6 owner_include

The `config` file contains one `owner_include` paragraph per data inclusion file (see 18.7, page 172. It concerns dynamic owner definition : inclusion of external data. For static owner definition and more information about owners see 21.1.5, page 201.

Example :

```
owner_include
source myfile
source_parameters a,b,c
reception nomail
profile normal
```

The `owner_include` directive is followed by one or several lines giving details regarding the owner(s) included characteristics :

- `source myfile`
  This is an mandatory field : it indicates the data inclusion file myfile.incl. This file can be a template. In this case, it will be interpreted with values given by subparameter `source_parameter`. Note that the `source` parameter should NOT include the *.incl* file extension ; the myfile.incl file should be located in the `data_sources` directory.
- `source_parameters a,b,c`
  It contains values enumeration that will be affected to the `param` array used in the template file (see 18.7, page 172). This parameter is uncompellable.
- `reception nomail`
  Optional attribute for owner(s) who does not wish to receive mails.
- `profile privileged | normal`
  Profile of the owner(s).

21.1.7 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.

21.1.8 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. This parameter is different from (`msg_topic`) parameter used to tag mails.

21.1.9 visibility

(Default value: `conceal`)

`visibility` parameter is defined by an authorization scenario (see 14, page 143)

This parameter indicates whether the list should feature in the output generated in response to a `LISTS` command.

– `visibility conceal`

– `visibility intranet`

– `visibility noconceal`

– `visibility secret`

21.2 Data source related

21.2.1 user_data_source

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

`user_data_source file | database | include | include2`

Sympa allows the mailing list manager to choose how *Sympa* loads subscriber and administartive data. User 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. This can be used for instance to share subscriber data 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 a 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 databasešection (8, page 79).
- `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_remote_sympa_list`, `include_sql_query` and `include_ldap_query`.
- `user_data_source include2`
  This is a replacement for the include mode. In this mode, the members cache is no more maitained in a DB FIle but in the main database instead. The behavior of the cache is detailed in the database chapter (see 8.6, page 91). This is the only mode that run the database for administrative data in the database

21.2.2 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.

21.2.3 include_list

`include_list listname`

This parameter will be interpreted only if `user_data_source` is set to `include` or `include2`. All subscribers of list `listname` become members 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.

Example: `include_list local-list`

Example: `include_list other-local-list@other-local-robot`

21.2.4 include_remote_sympa_list

`include_remote_sympa_list`

Sympa can contact another *Sympa* service using https to fetch a remote list in order to include each member of a remote list as subscriber. You may include as many lists as required, using one `include_remote_sympa_list` paragraph for each included list. Be careful, however, not to give rise an infinite loop making cross includes.

For this operation, one *Sympa* site act as a server while the other one act as client. On the server side, the only setting needed is to give permition to the remote *Sympa* to review the list. This is controled by the review authorization scenario.

From the client side you must define the remote list dump URI.

– `remote_host` *remote_host_name*
– `port` *port* (Default 443)
– `path` *absolute path* (In most cases, for a list name foo /sympa/dump/foo)

Because https offert a easy and secure client authentication, https is the only one protocole currently supported. A additional parameter is needed : the name of the certificate (and the private key) to be used :

– `cert list` the certificate to be use is the list certificate (the certificate subject distinguished name email is the list adress). Certificate and private key are located in the list directory.
– `cert robot` the certificate used is then related to sympa itself : the certificate subject distinguished name email look like sympa@my.domain and files are located in virtual host etc dir if virtual host is used otherwise in `/usr/local/sympa-os/etc`.

21.2.5 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 database type (mysql, SQLite, 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_port` *port*
  If not using the default RDBMS port, you can specify it.
- `db_name` *sympa_db_name*
  The hostname of the database system.
- `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 !
- `connect_options` *option1=x ;option2=y*
  This parameter is optional and specific to each RDBMS.
  These options are appended to the connect string.
  Example :

  ```
  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*
  This parameter is optional ; it is needed for some RDBMS (Oracle).
  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
  ```
- `name` *short name*
  This parameter is optional.
  It provides a human-readable name to this datasource. It will be used within the REVIEW page to indicate what datasource each list member comes from (usefull when having multiple data sources).
- `f_dir` */var/csvdir*
  This parameter is optional, only used when accessing a CSV datasource.
  When connecting to a CSV datasource, this parameter indicates the directory where the CSV files are located.

Example :

```
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
```

21.2.6 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 or a comma separated list of host :port. The second form is usefull if you are using some replication ldap host.
  Example :

  ```
        host ldap.cru.fr:389,backup-ldap.cru.fr:389
  ```

- `port` *ldap_directory_port* (OBSOLETE)
  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`.
- `use_ssl` *yes—no*
  If set to yes, LDAPS protocol is used.
- `ssl_version` *sslv2—sslv3—tls* (Default value: `sslv3`)
  If using SSL, this parameter define if SSL or TLS is used.
- `ssl_version` *ciphers used* (Default value: `ALL`)
  If using SSL, this parameter specifies which subset of cipher suites are permissible for this connection, using the standard OpenSSL string format. The default value of Net : :LDAPS for ciphers is ALL, which permits all ciphers, even those that don't encrypt !
- `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 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.

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
```

21.2.7 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 or a comma separated list of host :port. The second form is usefull if you are using some replication ldap host.
  Example :

  ```
  host ldap.cru.fr:389,backup-ldap.cru.fr:389
  ```

- `port` *ldap_directory_port* (OBSOLETE)
  Port on which the Directory accepts connections (this parameter is ignored if host definition include port specification).
- `user` *ldap_user_name*
  Username with read access to the LDAP directory.
- `passwd` *LDAP_user_password*
  Password for `user`.
- `use_ssl` *yes—no*
  If set to yes, LDAPS protocol is used.

- `ssl_version` *sslv2—sslv3—tls* (Default value: `sslv3`)
  If using SSL, this parameter define if SSL or TLS is used.
- `ssl_version` *ciphers used* (Default value: `ALL`)
  If using SSL, this parameter specifies which subset of cipher suites are permissible for this connection, using the standard OpenSSL string format. The default value of Net : :LDAPS for ciphers is ALL, which permits all ciphers, even those that don't encrypt !
- `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.univ.fr
port 389
suffix1 ou=Groups,dc=univ,dc=fr
scope1 one
filter1 (&(objectClass=groupOfUniqueNames) (| (cn=cri)(cn=ufrmi)))
attrs1 uniquemember
select1 all
suffix2 [attrs1]
scope2 base
filter2 (objectClass=n2pers)
attrs2 mail
select2 first
```

21.2.8 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 with an optional user description, separated from the email address by spaces (lines beginning with a ”#” are ignored).

Sample included file :

```
## Data for Sympa member import
john.smith@sample.edu  John Smith - math department
sarah.hanrahan@sample.edu  Sarah Hanrahan - physics department
```

21.2.9 include_remote_file

`include_remote_file`

This parameter (organized as a paragraph) does the same as the `include_file` parameter, except that it gets a remote file. This paragraph is used only if `user_data_source` is set to `include`. Using this method you should be able to include any *exotic* data source that is not supported by Sympa. The paragraph is made of the following entries :

- `url` *url_of_remote_file*
  This is the URL of the remote file to include.
- `user` *user_name*
  This entry is optional, only used if HTTP basic authentication is required to access the remote file.
- `passwd` *user_passwd*
  This entry is optional, only used if HTTP basic authentication is required to access the remote file.

Example :

```
include_remote_file
url     http://www.myserver.edu/myfile
user    john_netid
passwd  john_passwd
```

21.3 Command related

21.3.1 subscribe

(Default value: `open`)

`subscribe` parameter is defined by an authorization scenario (see 14, page 143)

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

- `subscribe auth`
- `subscribe auth_notify`
- `subscribe auth_owner`
- `subscribe closed`
- `subscribe intranet`
- `subscribe intranetorowner`

– `subscribe` `open`

– `subscribe` `open_notify`

– `subscribe` `open_quiet`

– `subscribe` `owner`

– `subscribe` `smime`

– `subscribe` `smimeorowner`

21.3.2 unsubscribe

(Default value: `open`)

`unsubscribe` parameter is defined by an authorization scenario (see 14, page 143)

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

– `unsubscribe` `auth`

– `unsubscribe` `auth_notify`

– `unsubscribe` `closed`

– `unsubscribe` `open`

– `unsubscribe` `open_notify`

– `unsubscribe` `owner`

21.3.3 add

(Default value: `owner`)

`add` parameter is defined by an authorization scenario (see 14, page 143)

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

– `add auth`

– `add closed`

– `add owner`

– `add owner_notify`

21.3.4 del

(Default value: `owner`)

`del` parameter is defined by an authorization scenario (see 14, page 143)

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

– `del auth`

– `del closed`

– `del owner`

– `del owner_notify`

21.3.5 remind

(Default value: `owner`)

`remind` parameter is defined by an authorization scenario (see 14, page 143)

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

– `remind listmaster`

– `remind owner`

21.3.6 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

21.3.7 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

21.3.8 send

(Default value: `private`)

`send` parameter is defined by an authorization scenario (see 14, page 143)

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

- `send closed`
- `send editorkey`
- `send editorkeyonly`
- `send editorkeyonlyauth`
- `send intranet`

– send `intranetorprivate`

– send `newsletter`

– send `newsletterkeyonly`

– send `private`

– send `private_smime`

– send `privateandeditorkey`

– send `privateandnomultipartoreditorkey`

– send `privatekey`

– send `privatekeyandeditorkeyonly`

– send `privateoreditorkey`

– send `privateorpublickey`

– send `public`

– send `public_nobcc`

– send `publickey`

– send `publicnoattachment`

– send `publicnomultipart`

21.3.9 review

(Default value: `owner`)

`review` parameter is defined by an authorization scenario (see 14, page 143)

This parameter specifies who can use `REVIEW` (see 28.1, page 256), administrative requests.

Predefined authorization scenarios are :

– `review closed`

– `review intranet`

– `review` `listmaster`

– `review` `owner`

– `review` `private`

– `review` `public`

21.3.10 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 an authorization scenario (see 14, page 143)

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

Predefined authorization scenarios are :

– `d_read` `owner`

– `d_read` `private`

– `d_read` `private-https`

– `d_read` `public`

d_edit

(Default value: `owner`)

`d_edit` parameter is defined by an authorization scenario (see 14, page 143)

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

Predefined authorization scenarios are :

- `d_edit` `editor`
- `d_edit` `owner`
- `d_edit` `private`
- `d_edit` `private-https`
- `d_edit` `public`

Example :

```
shared_doc
d_read public
d_edit private
```

quota

`quota` *number-of-Kbytes*

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

21.4 List tuning

21.4.1 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
```

21.4.2 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 `/usr/local/sympa-os/etc/sympa.conf` file.

21.4.3 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 :`

21.4.4 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.`

21.4.5 rfc2369_header_fields

(Default value: `rfc2369_header_fields sympa.conf parameter`)
`rfc2369_header_fields` *help,archive*

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

21.4.6 loop_prevention_regex

(Default value: `loop_prevention_regex sympa.conf parameter`)
`loop_prevention_regex` *mailer-daemon—sympa—listserv—majordomo—smartlist—mailman*

This regular expression is applied to messages sender address. If the sender address matches the regular expression, then the message is rejected. The goal of this parameter is to prevent loops between Sympa and other robots.

21.4.7 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). This string will be surrounded by [] characters.

The custom subject can also refer to list variables ([list-¿sequence] in the example bellow).

Example: `custom_subject sympa-users`.

Example: `custom_subject newsletter num [list->sequence]`.

21.4.8 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. `/usr/local/sympa-os/expl/mylist/message.footer.mime` will be ignored. Headers/footers may be appended to text/plain messages only.

21.4.9 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 (ie around midnight), *Sympa* may not be able to process it in time. Therefore do not setuse a digest time later than 23 :00.

N.B. : In family context, `digest` can be constrainted only on days.

21.4.10 `digest_max_size`

(Default value: `25`)

Maximum number of messages in a digest. If the number of messages exceeds this limit, then multiple digest messages are sent to each recipient.

21.4.11 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
```

21.4.12 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
```

21.4.13 msg_topic

The `msg_topic` parameter starts a paragraph to define a message topic used to tag a message. Foreach message topic, you have to define a new paragraph.(See 22.1, page 231)

Example :

```
msg_topic
name os
keywords linux,mac-os,nt,xp
title Operating System
```

Parameter `msg_topic.name` and `msg_topic.title` are mandatory. `msg_topic.title` is used on the web interface ("other" is not allowed for msg_topic.name parameter). The `msg_topic.keywords` parameter allows to select automatically message topic by searching keywords in the message.

N.B. : In a family context, `msg_topic.keywords` parameter is uncompellable.

21.4.14 msg_topic_keywords_apply_on

The `msg_topic_keywords_apply_on` parameter defines on which part of the message is used to perform automatic tagging.(See 22.1, page 231)

Example :

```
msg_topic_key_apply_on subject
```

Its values can be : subject | body| subject_and_body.

21.4.15 msg_topic_tagging

The `msg_topic_tagging` parameter indicates if the tagging is optional or required for a list. (See 22.1, page 231)

Example :

```
msg_topic_tagging optional
```

Its values can be : optional | required

21.4.16 pictures_feature

(Default value: `pictures_feature robot parameter`)

`pictures_feature` *on — off*

This enables the feature that allows list members to upload a picture that will be shown on review page.

21.4.17 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 21.1.5, page 201).

Example: `cookie secret22`

21.4.18 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.

21.5 Bounce related

21.5.1 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
```

21.5.2 bouncers_level1

– `rate`
  (Default value: `bouncers_level1_rate config parameter`)
  Each bouncing user have a score (from 0 to 100).This parameter define the lower score for a user to be a ¨level1 bouncing user·. For example, with default values : Users with a score between 45 and 80 are level1 bouncers.
– `action`
  (Default value: `bouncers_level1_action config parameter`)
  This parameter define which task is automaticaly applied on level 1 bouncing users : for exemple, automaticaly notify all level1 users.
– `Notification`
  (Default value: `owner`)
  When automatic task is executed on level 1 bouncers, a notification email can be send to listowner or listmaster. This email contain the adresses of concerned users and the name of the action executed.

21.5.3 bouncers_level2

– `rate`
  (Default value: `bouncers_level2_rate config parameter`)
  Each bouncing user have a score (from 0 to 100).This parameter define the lower score for a user to be a ¨level 2 bouncing user·. For example, with default values : Users with a score between 75 and 100 are level 2 bouncers.
– `action`
  (Default value: `bouncers_level1_action config parameter`)
  This parameter define which task is automaticaly applied on level 2 bouncing users : for exemple, automaticaly notify all level1 users.
– `Notification`
  (Default value: `owner`)
  When automatic task is executed on level 2 bouncers, a notification email can be send to listowner or listmaster. This email contain the adresses of concerned users and the name of the action executed.

Example :

```
## All bouncing adresses with a score between 75 and 100
## will be unsubscribed, and listmaster will recieve an email
Bouncers level 2
rate :75 Points
action : remove\_bouncers
Notification : Listmaster
```

21.5.4 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 remove the subscriber immediately in the case of a bounce. See `welcome_return_path` `sympa.conf` parameter (7.8.2, page 67).

21.5.5 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 (7.8.3, page 67).

21.5.6 verp_rate

(Default value: `verp_rate host parameter`)

See 24.1,page 242 for more information on VERP in Sympa.

When `verp_rate` is null VERP is not used ; if `verp_rate` is 100% VERP is alway in use.

VERP requires plussed aliases to be supported and the bounce+* alias to be installed.

21.6 Archive related

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

Mail archives can be retrieved via a mail command send to the robot, they are stored in `/usr/local/sympa-os/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`.

21.6.1 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 directory at the root of the list directory (`/usr/local/sympa-os/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.

21.6.2 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 WW-

Sympa 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 10000
```

access

`access_web_archive` parameter is defined by an authorization scenario (see 14, page 143)

Predefined authorization scenarios are :

– access `closed`

– access `intranet`

– access `listmaster`

– access `owner`

– access `private`

– access `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.

21.6.3 archive_crypted_msg

(Default value: `cleartext`)

`archive_crypted_msg` cleartext | decrypted

This parameter defines Sympa behavior while archiving S/MIME crypted messages. If set to `cleartext` the original crypted form of the message will be archived ; if set to `decrypted` a decrypted message will be archived. Note that this apply to both mail and web archives ; also to digests.

21.7 Spam protection

21.7.1 spam_protection

(Default value: `spam_protection robot parameter`)

There is a need to protection Sympa web site against spambot which collect email adresse in public web site. Various method are available into Sympa and you can choose it with `spam_protection` and `web_archive_spam_protection` parameters. Possible value are :

– javascript : the adresse is hidden using a javascript. User who enable javascript can see a nice mailto adresses where others have nothing.
– at : the @ char is replaced by the string " AT ".
– none : no protection against spammer.

21.7.2 web_archive_spam_protection

(Default value: `web_archive_spam_protection robot parameter`)

Idem `spam_protection` but restricted to web archive. A additional value is available : cookie which mean that users must submit a small form in order to receive a cookie before browsing archives. This block all robot, even google and co.

21.8 Intern parameters

21.8.1 family_name

This parameter indicates the name of the family that the list belongs to.

Example :

```
family_name my_family
```

21.8.2 latest_instantiation

This parameter indicates the date of the latest instantiation.

Example :

```
latest_instantiation
email serge.aumont@cru.fr
date 27 jui 2004 at 09:04:38
date_epoch 1090911878
```

Chapitre 22

Reception mode

22.1 Message topics

A list can be configured to have message topics (this notion is different from topics used to class mailing lists). Users can subscribe to these message topics in order to receive a subset of distributed messages : a message can have one or more topics and subscribers will receive only messages that have been tagged with a topic they are subscribed to. A message can be tagged automatically, by the message sender or by the list moderator.

22.1.1 Message topic definition in a list

Available message topics are defined by list parameters. Foreach new message topic, create a new `msg_topic` paragraph that defines the name and the title of the topic. If a thread is identified for the current message then the automatic procedure is performed. Else, to use automatic tagging, you should define keywords (See (21.4.13, page 221) To define which part of the message is used for automatic tagging you have to define `msg_topic_keywords_apply_on` list parameter (See 21.4.14, page 222). Tagging a message can be optional or it can be required, depending on the `msg_topic_tagging` list parameter (See (21.4.15,page 222).

22.1.2 Subscribing to message topic for list subscribers

This functionnality is only available via "normal" reception mode. Subscribers can select message topic to receive messages tagged with this topic. To receive messages that were not tagged, users can subscribe to the topic "other". Message topics selected by a subscriber are stored in *Sympa* database (subscriber_table table).

22.1.3 Message tagging

First of all, if one or more `msg_topic.keywords` are defined, *Sympa* tries to tag messages automatically. To trigger manual tagging, by message sender or list moderator, on the web interface, *Sympa* uses authorization scenarios : if the resulted action is "editorkey" (for example in scenario send.editorkey), the list moderator is asked to tag the message. If the resulted action is "request_auth" (for example in scenario send.privatekey), the message sender is asked to tag the message. The following variables are available as scenario variables to customize tagging : topic, topic-sender, topic-editor, topic-auto, topic-needed. (See (14, page 143) If message tagging is required and if it was not yet performed, *Sympa* will ask to the list moderator.

Tagging a message will create a topic information file in the `/usr/local/sympa-os/spool/topic/` spool. Its name is based on the listname and the Message-ID. For message distribution, a "X-Sympa-Topic" field is added to the message to allow members to use mail filters.

Chapitre 23

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 and to enforce security in the shared space, each document is linked to a set of specific control information : its access rights.

A list's shared documents are stored in the `/usr/local/sympa-os/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

23.1 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 applied on a directory, opens it and lists its contents (only those sub-documents the user is authorized to "see").
  - If applied on a file, downloads it, and in the case of a viewable file (*text/plain*, *text/html*, or image), displays it.
- The edit operation allows :
  - Subdirectory creation
  - File uploading
  - File unzipping
  - Description of a document (title and basic information)
  - On-line editing of a text file
  - Document (file or directory) removal. If on 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.

23.2 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` .

23.2.1 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
```

23.3 The predefined authorization scenarios

23.3.1 The public scenario

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

23.3.2 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.

23.3.3 The scenario owner

The scenario **owner** is the most restrictive scenario for a shared space. Only the list-master, 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.

23.3.4 The scenario editor

The scenario **editor** is for a moderated shared space for editing. Every suscriber of the list is allowed to editing a document. But this document will have to be installed or rejected by the editor of the list. Documents awaiting for moderation are visible by their author and the editor(s) of the list in the shared space. The editor has also an interface with all documents awaiting. When there is a new document, the editor is notiied and when the document is installed, the author is notiied too. In case of reject, the editor can notify the author or not.

23.4 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.

23.4.1 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.

23.4.2 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.

23.4.3 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 authorization 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>)
```

23.5 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 without moderation. 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 applied to a folder, it has to be empty.
- action D_UPLOAD
  Uploads a file into a directory that can be edited.
- action D_UNZIP
  Unzip a file into a directory that can be edited without moderation. The whole file hierarchy contained in the zip file is installed into the directory.
- 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.

23.6 Template files

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

23.6.1 d_read.tt2

The default page for reading a document. If for a file, displays it (if viewable) or downloads it. If for a directory, displays all readable subdocuments, each of which will feature buttons corresponding to the different actions this subdocument allows. If the directory is editable, displays buttons to describe it or upload a file to it. If the directory is editable without moderation, it displays button to create a new subdirector or to upload a zip file in order to install a file hierarchy. If access to the document is editable, displays a button to edit the access to it.

23.6.2 d_editfile.tt2

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

23.6.3 d_control.tt2

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

23.6.4 d_upload.tt2

This page to upload a file is only used when the name of the file already exists.

23.6.5 d_properties.tt2

This page is used to edit description file and to rename it.

Chapitre 24

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 (note that the -owner suffix can be customized, see 7.8.4, page 67), which should be sent to the `bouncequeue` program via aliases :

```
\samplelist-owner: "|/usr/local/sympa-os/bin/bouncequeue \samplelist"
```

`bouncequeue` (see 2.2, page 23) stores bounces in a `/usr/local/sympa-os/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.

With these informations, the automatic bounce management is possible :

– The automatic task `eval_bouncer` gives a score foreach bouncing user. The score, between 0 to 100, allows the classification of bouncing users in two levels. (Level 1 or 2). According to the level, automatic actions are executed periodicaly by `process_bouncers` task.

– The score evaluation main parameters are :
  `Bounces count` : The number of bouncing messages received by sympa for the user.
  `Type rate` : Bounces are classified depending on the type of errors generated on the user side. If the error type is "mailbox is full" (ie a temporary 4.2.2 error type) the type rate will be 0.5 whereas permanent errors (5.x.x) have a type rate equal to 1.
  `Regularity rate` : This rate tells if the bounces where received regularly, compared to list traffic. The list traffic is deduced from `msg_count` file data.

```
        The score formula is  :

        Score = bounce_count * type_rate * regularity_rate
```

  To avoid making decisions (ie defining a score) without enough relevant data, the score is not evaluated if :
  – The number of the number of received bounces is lower than `minimum_bouncing_count` (see 7.8.9, page 68)
  – The bouncing period is shorter than `minimum_bouncing_period` (see 7.8.10, page 69)

  Bouncing list members entries get expired after a given period of time. The default period is 10 days but it can be customized if you write a new **expire_bounce** task (see 7.8.5 ,page 68).
– You can define the limit between each level via the **List configuration pannel**, in subsection **Bounce settings**. (see 21.5.2) The principle consists in associating a score interval with a level.
– You can also define which action must be applied on each category of user.(see 21.5.2) Each time an action will be done, a notification email will be send to the person of your choice. (see 21.5.2)

24.1 VERP

VERP (Variable Envelop Return Path) is used to ease automatic recognition of subscribers email address when receiving a bounce. If VERP is enabled, the subscriber address is encoded in the return path itself so Sympa bounce management processus (bounced) will use the address the bounce was received for to retreive the subscriber email. This is very usefull because sometimes, non delivery report don't contain the initial subscriber email address but an alternative address where messages are forwarded. VERP is the only solution to detect automaticaly these subscriber errors but the cost of VERP is significant, indeed VERP requires to distribute a separate message for each subscriber and break the bulk emailer grouping optimization.

In order to benefit from VERP and keep distribution process fast, Sympa enables VERP only for a share of the list members. If texttt `verp_rate` (see 7.8.1,page 67) is 10% then after 10 messages distributed in the list all subscribers have received at least one message where VERP was enabled. Later distribution message enable VERP also for all users where some bounce wer collected and analysed by previous VERP mechanism.

If VERP is enabled, the format of the messages return path are as follows :

```
Return-Path: <bounce+user==a==userdomain==listname@listdomain>
```

Note that you need to set a mail alias for the generic bounce+* alias (see 6.1, page 45).

24.2 ARF

ARF (Abuse Feedback Reporting Format) is standard for reporting abuse. It is implemented mainly in AOL email user interface. Aol server propose to mass mailer to received automatically the users complain by formated messages. Because many subscribers don't want to remind how to unsubscribe tey use ARF when provided by user interface. It may usefull to configure the ARF managment in Sympa. It really simple : all what you have to do is to create a new alias for each virtual robot as the following :

```
abuse-feedback-report:       "| /usr/local/sympa-os/bin/bouncequeue sympa@\samplerobot"\\
```

Then register this address as your loop back email address with ISP (for exemple AOL). This way messages to that email adress are processed by bounced deamon and opt-out opt-out-list abuse and automatically processed. If bounce service can remove a user the message report feedback is forwarded to the list owner. Unrecognize message are forwarded to the listmaster.

Chapitre 25

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 7.13, page 77. *Sympa* is already compatible with McAfee/uvscan, Fsecure/fsav, Sophos, AVP, Trend Micro/VirusWall and Clam Antivirus. For each mail received, *Sympa* extracts its MIME parts in the `/usr/local/sympa-os/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.tt2` 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 26

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 instead of sympa internal storage of password see 13.5, page 132
- named filters used in authorization scenario condition see 14.2, page 147
- LDAP extraction of list subscribers (see 21.2.1)
- LDAP extraction of list owners or editors see 18.7, page 172
- mail aliases stored in LDAP see 6.3, page 48

Chapitre 27

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.

27.1 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 processe that distributes messages is compatible with end user signing processes. Sympa simply skips the message footer attachment (ref 18.11, page 176) to prevent any body corruption which would break the signature.

27.2 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.

27.3 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.

27.4 S/Sympa configuration

27.4.1 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).

27.4.2 managing user certificates

User certs are automatically catched by Sympa when receiving a signed s/mime message so if Sympa needs to send encrypted message to this user it can perform encryption using this certificate. This works fine but it's not conpliant with the PKI theory : Sympa should be able to search for user certificates using PKI certificate directory (LDAP) .

That's why Sympa tests the key usage certificate attribute to known if the certificate allows both encryption and signature.

Certificates are stored as PEM files in the `/usr/local/sympa-os/expl/X509-user-certs/` directory. Files are named user@some.domain@enc or user@some.domain@sign (@enc and @sign suffix are used according to certificates usage. No other tool is provided by Sympa in order to collect this certificate repository but you can easily imagine your own tool to create those files.

27.4.3 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` and `capath` and / or `cafile`.

- `openssl` : the path for the OpenSSL binary file, usually `/usr/local/ssl/bin/openSSL`
- `cafile` (or `capath`) : the path of a bundle (or path of the directory) of trusted CA certificates The file `~/usr/local/sympa-os/bin/etc/cabundle.crt` included in Sympa distribution can be used.
  The `cafile` file (or the `capath` directory) should be shared with your Apache+mod_ssl configuration. This is required because Sympa's web interface gets user certificates information from Apache mod_ssl module.
- `key_password` : the password used to protect all list private keys. xxxxxxx

27.4.4 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 authorization scenario for the operation you want to secure. (see 14, page 143).

When receiving a message, *Sympa* applies the authorization 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 authorization scenario does not recognize this authentication method, the operation requested will be rejected. Consequently, authorization scenarios distributed prior to version 2.7 are not compatible with the OpenSSL configuration of Sympa. All standard authorization scenarios (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 limit\'e aux abonn\'es, v\'erif de la signature smime

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

It as also possible to mix various authentication methods in a single authorization 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 r\'serv\'e aux abonn\'es avec authentification MD5 pr\'ealable

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

27.4.5 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 an 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 using `/usr/local/sympa-os/bin/p12topem.pl` script. This script allows you to install a PKCS#12 bundle file containing a private key and a certificate using the appropriate format.

It is a good idea to have a look at the OpenCA (http ://www.openssl.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` file or the `capath` Sympa configuration parameter.

The list private key must be installed in a file named `/usr/local/sympa-os/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 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 :) Mozilla or internet explorer allows certificates to be exported to a file.

Here is a way to install a certificat for a list :

- Get a list certificate is to obtain an personal e-mail certificate for the canonical list address in your browser as if it was your personal certificate,
- export the intended certificate it. The format used by Netscape is "pkcs#12". Copy this file to the list home directory.
- convert the pkcs#12 file into a pair of pem files : `cert.pem` and `private_key` using the `/usr/local/sympa-os/bin/p12topem.pl` script. Use `p12topem.pl -help` for details.
- be sure that `cert.pem` and `private_key` are owned by sympa with "r" access.
- 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.

27.5 Managing certificates with tasks

You may automate the management of certificates with two global task models provided with *Sympa*. See 17.8, page 164 to know more about tasks. Report to 7.12.4, page 76 to configure your *Sympa* to use these facilities.

27.5.1 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 `/usr/local/sympa-os/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.

27.5.2 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 28

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 have three-letter abbreviations (e.g. `REV` instead of `REVIEW`).

28.1 User commands

- `HELP`
  Provides instructions for the use of *Sympa* commands. The result is the content of the `helpfile.tt2` template file.
- `INFO` *listname*

Provides the parameters of the specified list (owner, subscription mode, etc.) and its description. The result is the content of `~welcome[.mime]`.

- `LISTS`
  Provides the names of lists managed by *Sympa*. This list is generated dynamically, using the `visibility` (see 21.1.9, page 203). The `lists.tt2` template defines the message return by the `LISTS` command.
- `REVIEW` *listname*
  Provides the addresses of subscribers if the run mode authorizes it. See the `review` parameter (21.3.9, page 215) 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 `stats` file.
  Access to this command is controlled by the `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 (21.3.9, page 215).
- `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, 21.3.1, page 211), 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, 21.3.1, page 211).
- `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 21.4.9, page 220). See the `SET LISTNAME MAIL` command (28.1, page 257) and the `reception` parameter (18.4, page 171).
- `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 (28.1, page 257) and the `reception` (18.4, page 171).
- `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` (18.4, page 171).
- `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` (18.4, page 171).
- `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` (18.4, page 171).
- `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` (18.4, page 171).
- `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 (28.1, page 257) and the `reception` parameter (18.4, page 171). Moreover, this mode allows message topic subscription (22.1, page 231)
- `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 (28.1, page 257) and the `visibility` parameter (21.1.9, page 203).
- `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 (28.1, page 257) and `visibility` parameter (21.1.9, page 203).
- `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`.

28.2 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 comands are :

– `ADD` *listname user@host firstname name*
  Add command similar to `SUBSCRIBE`. You can avoid user notification by using the `QUIET` prefix (ie : `QUIET ADD`).
– `DELETE` *listname user@host*
  Delete command similar to `SIGNOFF`. You can avoid user notification by using the `QUIET` prefix (ie : `QUIET DELETE`).
– `REMIND` *listname*
  `REMIND` is used by list owners in order to send an individual service reminder message to each subscriber. This message is made by parsing the remind.tt2 file.
– `REMIND` *
  `REMIND` is used by the listmaster 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.tt2 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 !

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

28.3 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 (21.3.8, page 214).

– `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 7.6.4, page 62), 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.

Chapitre 29

Internals

This chapter describes these modules (or a part of) :
– `src/mail.pm` : low level of mail sending
– `src/List.pm` : list processing and informations about structure and access to list configuration parameters
– `src/sympa.pm` : the main script, for messages and mail commands processing.
– `src/Commands.pm` : mail commands processing
– `src/wwsympa.pm` : web interface
– `src/report.pm` : notification and error reports about requested services (mail and web)
– `src/tools.pm` : various tools
– `src/Message.pm` : Message object used to encapsule a received message.

29.1 mail.pm

This module deals with mail sending and does the SMTP job. It provides a function for message distribution to a list, the message can be encrypted. There is also a function to send service messages by parsing tt2 files, These messages can be signed. For sending, a call to sendmail is done or the message is pushed in a spool according to calling context.

29.1.1 public functions

mail_file(), mail_message(), mail_forward(), set_send_spool(), reaper().

mail_file()

Message is written by parsing a tt2 file (or with a string). It writes mail headers if they are missing and they are encoded. Then the message is sent by calling mail : :sending() function (see 29.1.2, page 264).

IN :

1. `filename` : string - tt2 filename | ” - no tt2 filename sent
2. `rcpt` (+) : SCALAR | ref(ARRAY) - SMTP ”RCPT To :” field
3. `data` (+) : ref(HASH) - used to parse tt2 file, contains header values, keys are :
   - `return_path` (+) : SMTP ”MAIL From :” field *if send by SMTP*, q ”X-Sympa-From :” field *if send by spool*
   - `to` : ”To :” header field *else it is $rcpt*
   - `from` : ”From :” field *if $filename is not a full msg*
   - `subject` : ”Subject :” field *if $filename is not a full msg*
   - `replyto` : ”Reply-to :” field *if $filename is not a full msg*
   - `headers` : ref(HASH), keys are other mail headers
   - `body` : body message *if not $filename*
   - `lang` : tt2 language *if $filename*
   - `list` : ref(HASH) *if sign_mode='smime'* - keys are :
     - `name` : list name
     - `dir` : list directory
4. `robot` (+) : robot
5. `sign_mode` : 'smime'- the mail is signed with smime | undef - no signature

OUT : 1

mail_message()

Distributes a message to a list. The message is encrypted if needed, in this case, only one SMTP session is used for each recepient otherwise, recepient are grouped by domain for sending (it controls the number recepient arguments to call sendmail). Message is sent by calling mail : :sendto() function (see 29.1.2, page 264).

IN :

1. `message` (+) : ref(Message) - message to distribute
2. `from` (+) : message from
3. `robot` (+) : robot
4. `rcpt` (+) : ARRAY - recepients

OUT : `$numsmtp` = number of sendmail process | undef

mail_forward()

Forward a message by calling mail : :sending() function (see 29.1.2, page 264).

IN :

1. `msg` (+) : ref(Message) | ref(MIME : :Entity) | string - message to forward
2. `from` (+) : message from
3. `rcpt` (+) : ref(SCALAR) | ref(ARRAY) - recepients
4. `robot` (+) : robot

OUT : 1 | undef

set_send_spool()

Used by other processes than sympa.pl to indicate to send message by writing message in spool instead of calling mail : :smtpto() function (see 29.1.2, page 265). The concerned spool is set in `$send_spool` global variable, used by mail : :sending() function (see 29.1.2, page 264).

IN :

1. `spool` (+) : the concerned spool for sending.

OUT : -

reaper()

Non blocking function used to clean the defuncts child processes by waiting for them and then decreasing the counter. For exemple, this function is called by mail : :smtpto() (see 29.1.2, page 265), main loop of sympa.pl, task_manager.pl, bounced.pl.

IN :

1. `block`

OUT : the pid of the defunct process | -1 *if there is no such child process.*

29.1.2 private functions

sendto(), sending(), smtpto().

sendto()

Encodes subject header. Encrypts the message if needed. In this case, it checks if there is only one recepient. Then the message is sent by calling mail : :sending() function (see 29.1.2, page 264).

IN :

1. `msg_header` (+) : ref(MIME : :Head) - message header
2. `msg_body` (+) : message body
3. `from` (+) : message from
4. `rcpt` (+) : ref(SCALAR) | ref(ARRAY) - message recepients (ref(SCALAR) for encryption)
5. `robot` (+) : robot
6. `encrypt` : 'smime_crypted' the mail is encrypted with smime | undef - no encryption

OUT : 1 - sending by calling smtpto (sendmail) | 0 - sending by push in spool | undef

sending()

Signs the message according to `$sign_mode`. Chooses sending mode according to context. If `$send_spool` global variable is empty, the message is sent by calling mail : :smtpto() function (see 29.1.2, page 265) else the message is written in spool `$send_spool` in order to be handled by sympa.pl process (because only this is allowed to make a fork). When the message is pushed in spool, these mail headers are added :
– "X-Sympa-To :" : recepients
– "X-Sympa-From :" : from
– "X-Sympa-Checksum :" : to check allowed program to push in spool
A message pushed in spool like this will be handled later by sympa : :DoSendMessage() function (see 29.3, page 279)

IN :

1. `msg` (+) : ref(MIME : :Entity) | string - message to send
2. `rcpt` (+) : ref(SCALAR) | ref(ARRAY) - recepients (for SMTP : "RCPT To :" field)
3. `from` (+) : for SMTP : "MAIL From :" field | for spool sending : "X-Sympa-From" field
4. `robot` (+) : robot
5. `listname` : listname | "
6. `sign_mode` (+) : 'smime' | 'none'for signing
7. `sympa_email` : for the file name for spool sending

OUT : 1 - sending by calling smtpto() (sendmail) | 0 - sending by pushing in spool | undef

smtpto()

Calls to sendmail for the recipients given as argument by making a fork and an exec. Before, waits for number of children process $<$ number allowed by sympa.conf by calling mail : :reaper() function (see 29.1.1, page **??**).

IN :

1. `from` (+) : SMTP ”MAIL From :” field
2. `rcpt` (+) : ref(SCALAR)) | ref(ARRAY) - SMTP ”RCPT To :” field
3. `robot` (+) : robot

OUT : `mail : :$fh` - file handle on opened file for ouput, for SMTP ”DATA” field | undef

29.2 List.pm

This module includes list processing functions.

Here are described functions about :
– Message distribution in a list
– Sending using templates
– Service messages
– Notification message
– Topic messages
– Scenario evaluation
Follows a description of structure and access on list parameters.

29.2.1 Functions for message distribution

distribute_message(), send_msg(), send_msg_digest().

These functions are used to message distribution in a list.

distribute_msg()

Prepares and distributes a message to a list :
– updates the list stats
– Loads information from message topic file if exists and adds X-Sympa-Topic header
– hides the sender if the list is anonymoused (list config : anonymous_sender) and changes name of msg topic file if exists.

– adds custom subject if necessary (list config : custom_subject)
– archives the message
– changes the reply-to header if necessary (list config : reply_to_header)
– removes unwanted headers if present (config : remove_headers))
– adds useful headers (X-Loop,X-Sequence,Errors-to,Precedence,X-no-archive - list config : custom_header)
– adds RFC 2919 header field (List-Id) and RFC 2369 header fields (list config : rfc2369_header_fields)
– stores message in digest if the list accepts digest mode (encrypted message can't be included in digest)
– sends the message by calling List : :send_msg() (see 29.2.1, page 266).

IN :

1. `self` (+) : ref(List) - the list concerned by distribution
2. `message` (+) : ref(Message) - the message to distribute

OUT : result of List : :send_msg() function (number of sendmail process)

send_msg()

This function is called by List : :distribute_msg() (see 29.2.1, page 265) to select subscribers according to their reception mode and to the "Content-Type" header field of the message. Sending are grouped according to their reception mode :
– normal : add a footer if the message is not protected (then the message is "altered") In a message topic context, selects only one who are subscribed to the topic of the message to distribute (calls to select_subcribers_for_topic(), see **??**, page **??**).
– notice
– txt : add a footer
– html : add a footer
– urlize : add a footer and create an urlize directory for Web access
The message is sent by calling List : :mail_message() (see 29.1.1, page 262). If the message is "smime_crypted" and the user has not got any certificate, a message service is sent to him.

IN :

1. `self` (+) : ref(List) - the list concerned by distribution
2. `message` (+) : ref(Message) - the message to distribute

OUT : `$numsmtp` : addition of mail : :mail_message() function results (= number of sendmail process) | undef

send_msg_digest()

Sends a digest message to the list subscribers with reception digest, digestplain or summary : it creates the list of subscribers in various digest modes and then creates the list of messages. Finally sending to subscribers is done by calling List : :send_file() function (see 29.2.2, page 267) with mail template "digest", "digestplain" or "summary".

IN :

1. `self` (+) : ref(List) - the concerned list

OUT :
– 1 *if sending*
– 0 *if no subscriber for sending digest, digestplain or summary*
– undef

29.2.2 Functions for template sending

send_file(), send_global_file().

These functions are used by others to send files. These files are made from template given in parameters.

send_file()

Sends a message to a user, relative to a list. It finds the $tpl.tt2 file to make the message. If the list has a key and a certificat and if openssl is in the configuration, the message is signed. The parsing is done with variable $data set up first with parameter $context and then with configuration, here are set keys :
– *if $who=SCALAR then*
  – `user.password`
  – *if $user key is not defined in $context then* `user.email`(:= $who), `user.lang` (:= list lang) *and if the user is in DB then* `user.attributes` (:= attributes in DB user_table) are defined
  – *if $who is subscriber of $self then* `subscriber.date` `subscriber.update_date` *and if exists then* `subscriber.bounce` `subscriber.first_bounce` are defined
– `return_path` : used for SMTP “MAIL From” field or ”X-Sympa-From :” field
– `lang` : the user lang or list lang or robot lang
– `fromlist` : ”From :” field, pointed on list
– `from` : ”From :” field, pointed on list *if no defined in $context*
– `replyto` : *if openssl is is sympa.conf and the list has a key ('private_key') and a certificat ('cert.pem') in its directory*
– `boundary` : boundary for multipart message *if no contained in $context*
– `conf.email conf.host conf.sympa conf.request conf.listmaster conf.wwsympa_url conf.title` : updated with robot config
– `list.lang list.name list.domain list.host list.subject list.dir list.owner`(ref(ARRAY)) : updated with list config
The message is sent by calling mail : :mail_file() function (see 29.1.1, page 262).

IN :

1. `self` (+) : ref(List)
2. `tpl` (+) : template file name without .tt2 extension ($tpl.tt2)

3. `who` (+) : SCALAR | ref(ARRAY) - recepient(s)
4. `robot` (+) : robot
5. `context` : ref(HASH) - for the $data set up

OUT : 1 | undef

send_global_file()

Sends a message to a user not relative to a list. It finds the $tpl.tt2 file to make the message. The parsing is done with variable $data set up first with parameter $context and then with configuration, here are set keys :

– `user.password user.lang`
– *if $user key is not defined in $context then* `user.email` (:= $who)
– `return_path` : used for SMTP "MAIL From" field or "X-Sympa-From :" field
– `lang` : the user lang or robot lang
– `from` : "From :" field, pointed on SYMPA *if no defined in $context*
– `boundary` : boundary for multipart message *if no defined in $context*
– `conf.email conf.host conf.sympa conf.request conf.listmaster conf.wwsympa_url conf.title` : updated with robot config
– `conf.version` : *Sympa* version
– `robot_domain` : the robot

The message is sent by calling mail : :mail_file() function (see 29.1.1, page 262).

IN :

1. `tpl` (+) : template file name (filename.tt2), without .tt2 extension
2. `who` (+) : SCALAR | ref(ARRAY) - recepient(s)
3. `robot` (+) : robot
4. `context` : ref(HASH) - for the $data set up

OUT : 1 | undef

29.2.3 Functions for service messages

archive_send(), send_to_editor(), request_auth(), send_auth().

These functions are used to send services messgase, correponding to a result of a command.

archive_send()

Sends an archive file ($file) to $who. The archive is a text file, independant from web archives. It checks if the list is archived. Sending is done by callingList : :send_file() (see 29.2.2, page 267) with mail template "archive".

IN :

1. `self` (+) : ref(List) - the concerned list
2. `who` (+) : recepient
3. `file` (+) : name of the archive file to send

OUT : - | undef

send_to_editor()

Sends a message to the list editor for a request concerning a message to distribute. The message awaiting for moderation is named with a key and is set in the spool queuemod. The key is a reference on the message for editor. The message for the editor is sent by calling List : :send_file() (see 29.2.2, page 267) with mail template “moderate”. In msg_topic context, the editor is asked to tag the message.

IN :

1. `self` (+) : ref(List) - the concerned list
2. `method` : ’md5’ - for ”editorkey” | ’smtp’ - for ”editor”
3. `message` (+) : ref(Message) - the message to moderate

OUT : $modkey - the moderation key for naming message waiting for moderation in spool queuemod. | undef

request_auth()

Sends an authentification request for a requested command. The authentification request contains the command to be send next and it is authentified by a key. The message is sent to user by calling List : :send_file() (see 29.2.2, page 267) with mail template “request_auth”.

IN :

1. `self` : ref(List) *not required if $cmd = “remind”.*
2. `email`(+) : recepient, the requesting command user
3. `cmd` :
   – *if $self then* ’signoff’ | ’subscribe’ | ’add’ | ’del’ | ’remind’
   – *else* ’remind’
4. `robot` (+) : robot
5. `param` : ARRAY
   – 0 : used *if $cmd =’subscribe’* | *’add’* | *’del’* | *’invite’*
   – 1 : used *if $cmd =’add’*

OUT : 1 | undef

send_auth()

Sends an authentifiaction request for a message sent for distribution. The message for distribution is copied in the authqueue spool to wait for confirmation by its sender . This message is named with a key. The request is sent to user by calling List : :send_file() (see 29.2.2, page 267) with mail template "send_auth". In msg_topic context, the sender is asked to tag his message.

IN :

1. `self`(+) : ref(List) - the concerned list
2. `message`(+) : ref(Message) - the message to confirm

OUT : $modkey, the key for naming message waiting for confirmation in spool queuemod. | undef

29.2.4 Functions for message notification

send_notify_to_listmaster(), send_notify_to_owner(), send_notify_to_editor(), send_notify_to_user().

These functions are used to notify listmaster, list owner, list editor or user about events.

send_notify_to_listmaster()

Sends a notice to listmaster by parsing "listmaster_notification" template. The template makes a specified or a generic treatement according to variable $param.type (:= $operation parameter). The message is sent by calling List : :send_file() (see 29.2.2, page 267) or List : :send_global_file() (see 29.2.2, page 268) according to the context : global or list context. Available variables for the template are set up by this function, by $param parameter and by List : :send_global_file() or List : :send_file().

IN :

1. `operation` (+) : notification type, corresponds to $type in the template
2. `robot` (+) : robot
3. `param` (+) : ref(HASH) | ref (ARRAY) - values for variable used in the template :
   – *if ref(HASH) then* variables used in the template are keys of this HASH. These following keys are required in the function, according to $operation value :
     – 'listname'(+) *if $operation=('request_list_creation' | 'automatic_bounce_management')*
   – *if ref(ARRAY) then* variables used in template are named as : $param0, $param1, $param2, ...

OUT : 1 | undef

send_notify_to_owner()

Sends a notice to list owner(s) by parsing “listowner_notification” template. The template makes a specified or a generic treatement according to variable $param.type (:= $operation parameter). The message is sent by calling List : :send_file() (see 29.2.2, page 267). Available variables for the template are set up by this function, by $param parameter and by List : :send_file().

IN :

1. `self` (+) : ref(List) - the list for owner notification
2. `operation` (+) : notification type, corresponds to $type in the template
3. `param` (+) : ref(HASH) | ref (ARRAY) - values for variable used in the template :
   - *if ref(HASH) then* variables used in the template are keys of this HASH.
   - *if ref(ARRAY) then* variables used in template are named as : $param0, $param1, $param2, ...

OUT : 1 | undef

send_notify_to_editor()

Sends a notice to list editor(s) by parsing “listeditor_notification” template. The template makes a specified or a generic treatement according to variable $param.type (:= $operation parameter). The message is sent by calling List : :send_file() (see 29.2.2, page 267). Available variables for the template are set up by this function, by $param parameter and by List : :send_file().

IN :

1. `self` (+) : ref(List) - the list for editor notification
2. `operation` (+) : notification type, corresponds to $type in the template
3. `param` (+) : ref(HASH) | ref (ARRAY) - values for variable used in the template :
   - *if ref(HASH) then* variables used in the template are keys of this HASH.
   - *if ref(ARRAY) then* variables used in template are named as : $param0, $param1, $param2, ...

OUT : 1 | undef

send_notify_to_user()

Sends a notice to a user by parsing “user_notification” template. The template makes a specified or a generic treatement according to variable $param.type (:= with $operation parameter). The message is sent by calling List : :send_file() (see 29.2.2, page 267). Available variables for the template are set up by this function, by $param parameter and by List : :send_file().

IN :

1. `self` (+) : ref(List) - the list for owner notification
2. `operation` (+) : notification type, corresponds to $type in the template
3. `user` (+) : user email to notify
4. `param` (+) : ref(HASH) | ref (ARRAY) - values for variable used in the template :
   - *if ref(HASH) then* variables used in the template are keys of this HASH.
   - *if ref(ARRAY) then* variables used in template are named as : $param0, $param1, $param2, ...

OUT : 1 | undef

29.2.5 Functions for topic messages

is_there_msg_topic(), is_available_msg_topic(), get_available_msg_topic(), is_msg_topic_tagging_required, automatic_tag(), compute_topic(), tag_topic(), load_msg_topic_file(), modifying_msg_topic_for_subscribers(), select_subscribers_for_topic().

These functions are used to manages message topics.

N.B. : There is some exception to use some parameters : msg_topic.keywords for list parameters and topics_subscriber for subscribers options in the DB table. These parameters are used as string splitted by ',' but to access to each one, use the function tools : :get_array_from_splitted_string() (see 29.7, page 297) allows to access the enumeration.

is_there_msg_topic()

Tests if some message topic are defined (`msg_topic` list parameter, see **??**, page **??**).

IN : `self` (+) : ref(List)

OUT : 1 - some msg_topic are defined | 0 - no msg_topic

is_available_msg_topic()

Checks for a topic if it is available in the list : look foreach `msg_topic.name` list parameter (see **??**, page **??**).

IN :

1. `self` (+) : ref(List)
2. `topic` (+) : the name of the requested topic

OUT : `topic` *if it is available* | undef

get_available_msg_topic()

Returns an array of available message topics (`msg_topic.name` list parameter, see **??**, page **??**).

IN : `self` (+) : ref(List)

OUT : ref(ARRAY)

is_msg_topic_tagging_required()

Returns if the message must be tagged or not (`msg_topic_tagging` list parameter set to 'required', see **??**, page **??**).

IN : `self` (+) : ref(List)

OUT : 1 - the message must be tagged | 0 - the msg can be no tagged

automatic_tag()

Computes topic(s) (with compute_topic() function) and tags the message (with tag_topic() function) if there are some topics defined.

IN :

1. `self` (+) : ref(List)
2. `msg` (+) : ref(MIME : :Entity)- the message to tag
3. `robot` (+) : robot

OUT : list of tagged topic : strings separated by ','. It can be empty. | undef

compute_topic()

Computes topic(s) of the message. If the message is in a thread, topic is got from the previous message else topic is got from applying a regexp on the subject and/or the body of the message (`msg_topic_keywords_apply_on` list parameter, see**??**, page **??**). Regexp is based on `msg_topic.keywords` list parameters (See **??**, page **??**).

IN :

1. `self` (+) : ref(List)
2. `msg` (+) : ref(MIME : :Entity)- the message to tag

OUT : list of computed topic : strings separated by ','. It can be empty.

tag_topic()

Tags the message by creating its topic information file in the `/usr/local/sympa-os/spool/topic/` spool. The file contains the topic list and the method used to tag the message. Here is the format :

```
TOPIC topicname,...
METHOD editor|sender|auto
```

IN :

1. `self` (+) : ref(List)
2. `msg_id` (+) : string - the message ID of the message to tag
3. `topic_list` (+) : the list of topics (strings splitted by ',')
4. `method` (+) : 'auto' |'editor'|'sender' - the method used for tagging

OUT : name of the created topic information file (`directory/listname.msg_id`) | undef

load_msg_topic_file()

Search and load msg topic file corresponding to the message ID (`directory/listname.msg_id`). It returns information contained inside.

IN :

1. `self` (+) : ref(List)
2. `msg_id` (+) : the message ID
3. `robot` (+) : the robot

OUT : undef | ref(HASH), keys are :
– `topic` : list of topics (strings separated by ',')
– `method` : 'auto' |'editor'|'sender' - the method used for tagging
– `msg_id` : message ID of the tagged message
– `filename` : name of the file

modifying_msg_topic_for_subscribers()

Deletes topics of subscriber that does not exist anymore and send a notify to concerned subscribers. (Makes a diff on msg_topic parameter between the list configuration before modification and a new state by calling tools : :diff_on_arrays() function, see 29.7, page 298). This function is used by wwsympa : :do_edit_list().

IN :

1. `self` (+) : ref(List) - the concerned list before modification
2. `new_msg_topic` (+) : ref(ARRAY) - new state of msg_topic parameters

OUT :

1. 1 *if some subscriber topics have been deleted*
2. 0 *else*

select_subscribers_for_topic()

Selects subscribers that are subscribed to one or more topic appearing in the topic list incoming when their reception mode is 'mail', and selects the other subscribers (reception mode different from 'mail'). This function is used by List : :send_msg() function during message diffusion (see 29.2.1, page 266).

IN :

1. `self` (+) : ref(List)
2. `string_topic` (+) : string splitted by ',' - the topic list
3. `subscribers` (+) : ref(ARRAY) - list of subscriber emails

OUT : ARRAY - list of selected subscribers

29.2.6 Scenario evaluation

The following function is used to evaluate scenario file "<action>.<parameter_value>", where <action>action corresponds to a configuration parameter for an action and <parameter_value> corresponds to its value.

request_action()

Return the action to perform for one sender using one authentication method to perform an operation

IN :

1. `operation` (+) : SCALAR - the requested action corresponding to config parameter
2. `auth_method` (+) : 'smtp'|'md5'|'pgp'|'smime'
3. `robot` (+) : robot
4. `context` () : ref(HASH) - contains value to instantiate scenario variables (hash keys)
5. `debug` () : boolean - *if true* adds keys 'condition' and 'auth_method' to the returned hash.

OUT : undef | ref(HASH) with keys :
– action : 'do_it'|'reject'|'request_auth'|'owner'|'editor'|'editorkey'|'listmaster'
– reason : 'value' *if action == 'reject' in scenario and if there is reject(reason='value')* to match a key in mail_tt2/authorization_reject.tt2. This is used in errors reports (see **??**, page **??**)
– tt2 : template name if action == 'reject' in scenario and there is reject(tt2='template_name').
– condition : the checked condition.
– auth_method : the checked auth_method.

29.2.7 Structure and access to list configuration parameters

List parameters are representated in the list configuration file, in the list object (`list->{'admin'}`) and on the Web interface. Here are translation and access functions :

		other (3)		
	(1)⟶	↑	(5)⟶	
CONFIG FILE		LIST OBJECT		WEB INTERFACE
	⟵ (2)	(4)	⟵ (6)	

1. Loading file in memory :
   `List::_load_admin_file(),_load_include_admin_user_file(),_load_list_param()`
2. Saving list configuration in file :
   `List::_save_admin_file(),_save_list_param()`
3. Tools to get parameter values :
   `List::get_param_value(),_get_param_value_anywhere(),_get_single_param_value()`
4. Tools to initialize list parameter with defaults :
   `List::_apply_default()`
5. To present list parameters on the web interface :
   `wwsympa::do_edit_list_request(),_prepare_edit_form(),_prepare_data()`
6. To get updates on list parameters from the web interface :
   `wwsympa::do_edit_list(),_check_new_value`

List parameters can be simple or composed in paragraph, they can be unique or multiple and they can singlevalued or multivalued. Here are the different kinds of parameters and an exemple :

parameters		SIMPLE	COMPOSED
SINGLE	singlevalued	(a) *lang*	(b) *archiv.period*
	multivalued	(c) *topics*	(d) *available_user_option.reception*
MULTIPLE	singlevalued	(e) *include_list*	(f) *owner.email*
	multi values	not defined	not defined

Here are these list parameters format in list configuration file in front of perl representation in memory :

	List Configuration FILE	`$list->{'admin'}`
(a)	param value	'scalar'
(b)	param p1 val1 p2 val2	'HASH→scalar'
(c)	param val1,val2,val3	'ARRAY(scalar & split_char)'
(d)	param p1 val11, val12, val13 p2 val21, val22, val23	'HASH→ARRAY(scalar & split_char)'
(e)	param val1 param val2	'ARRAY(scalar)'
(d)	param p1 val11 p2 val12 param p1 val21 p2 val22	'ARRAY(HASH→scalar)'

29.3 sympa.pl

This is the main script ; it runs as a daemon and does the messages/commands processing. It uses these funstions : DoFile(), DoMessage(), DoCommand(), DoSendMessage(), DoForward(), SendDigest(), CleanSpool(), sigterm(), sighup().

Concerning reports about message distribution, function List : :send_file() (see 29.2.2, page 267) or List : :send_global_file() (see 29.2.2, page 268) is called with mail template "message_report". Concernong reports about commands, it is the mail template "command_report".

DoFile()

Handles a received file : function called by the sympa.pl main loop in order to process files contained in the queue spool. The file is encapsulated in a Message object not to alter it. Then the file is read, the header and the body of the message are separeted. Then the adequate function is called whether a command has been received or a message has to be redistributed to a list.

So this function can call various functions :
– sympa : :DoMessage() for message distribution (see 29.3, page 278)
– sympa : :DoCommand() for command processing (see 29.3, page 278)
– sympa : :DoForward() for message forwarding to administrators (see 29.3, page 279)
– sympa : :DoSendMessage() for wwsympa message sending (see 29.3, page 279).
About command process a report can be sent by calling List : :send_global_file() (see 29.2.2, page 268) with template "command_report". For message report it is the template "message_report".

IN : `file`(+) : the file to handle

OUT : $status - result of the called function | undef

DoMessage()

Handles a message sent to a list (Those that can make loop and those containing a command are rejected). This function can call various functions :
– List : :distribute_msg() for distribution (see 29.2.1, page 265)
– List : :send_auth() for authentification or topic tagging by message sender(see 29.2.3, page 270)
– List : :send_to_editor() for moderation or topic tagging by list moderator(see 29.2.3, page 269).
– List : :automatic_tag() for automatic topic tagging (see 29.2.5, page 273).
IN :

1. `which`(+) : 'list_name@domain_name - the concerned list
2. `message`(+) : ref(Message) - sent message
3. `robot`(+) : robot

OUT : 1 *if everything went fine* in order to remove the file from the queue | undef

DoCommand()

Handles a command sent to sympa. The command is parse by calling Commands : :parse() (see 29.4.1, page 281).

IN :

1. `rcpt` : recepient | <listname>-<subscribe|unsubscribe>
2. `robot`(+) : robot
3. `msg`(+) : ref(MIME : :Entity) - message containing the command
4. `file`(+) : file containing the message

OUT : $success - result of Command : :parse() function | undef.

DoSendMessage()

Sends a message pushed in spool by another process (ex : wwsympa.fcgi) by calling function mail : :mail_forward() (see 29.1.1, page 263).

IN :

1. `msg`(+) : ref(MIME : :Entity)
2. `robot`(+) : robot

OUT : 1 | undef

DoForward()

Handles a message sent to <listname>-editor : the list editor, <list>-request : the list owner or the listmaster. The message is forwarded according to $function by calling function mail : :mail_forward() (see 29.1.1, page 263).

IN :

1. `name`(+) : list name *if ($function != 'listmaster')*
2. `function`(+) : 'listmaster' | 'request' | 'editor'
3. `robot`(+) : robot
4. `msg`(+) : ref(MIME : :Entity)

OUT : 1 | undef

SendDigest()

Reads the queuedigest spool and send old digests to the subscribers with the digest option by calling List : :send_msg_digest() function mail : :mail_forward() (see 29.2.1, page 266).

IN : - **OUT** : - | undef

CleanSpool()

Cleans old files from spool $spool_dir older than $clean_delay.

IN :

1. `spool_dir`(+) : the spool directory
2. `clean_delay`(+) : the delay in days

OUT : 1

sigterm()

This function is called when a signal -TERM is received by sympa.pl. It just changes the value of $signal loop variable in order to stop sympa.pl after endding its message distribution if in progress. (see 4.3, page 37)

IN : - **OUT** : -

sighup()

This function is called when a signal -HUP is received by sympa.pl. It changes the value of $signal loop variable and switchs of the ”–mail” (see 4.3, page 37) logging option and continues current task.

IN : - **OUT** : -

29.4 Commands.pm

This module does the mail commands processing.

29.4.1 Commands processing

parse(), add(), del(), subscribe(), signoff(), invite(), last(), index(), getfile(), confirm(), set(), distribute(), reject(), modindex(), review(), verify(), remind(), info(), stats(), help(), lists(), which(), finished().

parse()

Parses the command line and calls the adequate subroutine (following functions) with the arguments of the command. This function is called by sympa : :DoCommand() (see 29.3, page 278).

IN :

1. `sender`(+) : the command sender
2. `robot`(+) : robot
3. `i`(+) : command line
4. `sign_mod` : 'smime' | undef

OUT : 'unknown_cmd' | $status - command process result

add()

Adds a user to a list (requested by another user), and can send acknowledgements. New subscriber can be notified by sending template 'welcome'.

IN :

1. `what`(+) : command parameters : listname, email and comments eventually
2. `robot`(+) : robot
3. `sign_mod` : 'smime' | undef - authentification

OUT : 'unknown_list' | 'wrong_auth' | 'not_allowed' | 1 | undef

del()

Removes a user to a list (requested by another user), and can send acknowledgements. Unsubscriber can be notified by sending template 'removed'.

IN :

1. `what`(+) : command parameters : listname and email
2. `robot`(+) : robot
3. `sign_mod` : 'smime' | undef - authentification

OUT : 'unknown_list' | 'wrong_auth' | 'not_allowed' | 1

subscribe()

Subscribes a user to a list. New subscriber can be notified by sending him template 'welcome'.

IN :

1. `what(+)` : command parameters : listname and comments eventually
2. `robot(+)` : robot
3. `sign_mod` : 'smime' | undef - authentification

OUT : 'unknown_list' | 'wrong_auth' | 'not_allowed' | 1 | undef

signoff()

Unsubscribes a user from a list. He can be notified by sending him template 'bye'.

IN :

1. `which(+)` : command parameters : listname and email
2. `robot(+)` : robot
3. `sign_mod` : 'smime' | undef - authentification

OUT : 'syntax_error' | 'unknown_list' | 'wrong_auth' | 'not_allowed' | 1 | undef

invite()

Invites someone to subscribe to a list by sending him the template 'invite'.

IN :

1. `what(+)` : command parameters : listname, email and comments
2. `robot(+)` : robot
3. `sign_mod` : 'smime' | undef - authentification

OUT : 'unknown_list' | 'wrong_auth' | 'not_allowed' | 1 | undef

last()

Sends back the last archive file by calling List : :archive_send() function (see 29.2.3, page 268).

IN :

1. `which(+)` : listname
2. `robot(+)` : robot

OUT : 'unknown_list' | 'no_archive' | 'not_allowed' | 1

index()

Sends the list of archived files of a list.

IN :

1. `which`(+) : listname
2. `robot`(+) : robot

OUT : 'unknown_list' | 'no_archive' | 'not_allowed' | 1

getfile()

Sends back the requested archive file by calling List : :archive_send() function (see 29.2.3, page 268).

IN :

1. `which`(+) : commands parameters : listname and filename(archive file)
2. `robot`(+) : robot

OUT : 'unknown_list' | 'no_archive' | 'not_allowed' | 1

confirm()

Confirms the authentification of a message for its distribution on a list by calling function List : :distribute_msg() for distribution (see 29.2.1, page 265) or by calling List : :send_to_editor() for moderation (see **??**, page **??**).

IN :

1. `what`(+) : authentification key (command parameter)
2. `robot`(+) : robot

OUT : 'wrong_auth' | 'msg_not_found' | 1 | undef

set()

Changes subscription options (reception or visibility)

IN :

1. `what`(+) : command parameters : listname and reception mode (digest|digestplain|nomail|normal...) or visibility mode(conceal|noconceal).
2. `robot`(+) : robot

OUT : 'syntax_error' | 'unknown_list' | 'not_allowed' | 'failed' | 1

distribute()

Distributes the broadcast of a validated moderated message.

IN :

1. `what`(+) : command parameters : listname and authentification key
2. `robot`(+) : robot

OUT : 'unknown_list' | 'msg_not_found' | 1 | undef

reject()

Refuses and deletes a moderated message. Rejected message sender can be notified by sending him template 'reject'.

IN :

1. `what`(+) : command parameters : listname and authentification key
2. `robot`(+) : robot

OUT : 'unknown_list' | 'wrong_auth' | 1 | undef

modindex()

Sends a list of current messages to moderate of a list (look into spool queuemod) by using template 'modindex'.

IN :

1. `name`(+) : listname
2. `robot`(+) : robot

OUT : 'unknown_list' | 'not_allowed' | 'no_file' | 1

review()

Sends the list of subscribers of a list to the requester by using template 'review'.

IN :

1. `listname`(+) : list name
2. `robot`(+) : robot
3. `sign_mod` : 'smime' | undef - authentification

OUT : 'unknown_list' | wrong_auth | no_subscribers | 'not_allowed' | 1 | undef

verify()

Verifies an S/MIME signature.

IN :

1. `listname`(+) : list name
2. `robot`(+) : robot
3. `sign_mod` : 'smime' | undef - authentification

OUT : 1

remind()

Sends a personal reminder to each subscriber of a list or of every list (if $which = *) using template 'remind' or 'global_remind'.

IN :

1. `which`(+) : * | listname
2. `robot`(+) : robot
3. `sign_mod` : 'smime' | undef - authentification

OUT : 'syntax_error' | 'unknown_list' | 'wrong_auth' | 'not_allowed' | 1 | undef

info()

Sends the list information file to the requester by using template 'info_report'.

IN :

1. `listname`(+) : name of concerned list
2. `robot`(+) : robot
3. `sign_mod` : 'smime' | undef - authentification

OUT : 'unknown_list' | 'wrong_auth' | 'not_allowed' | 1 | undef

stats()

Sends the statistics about a list using template 'stats_report'.

IN :

1. `listname`(+) : list name

2. `robot`(+) : robot
3. `sign_mod` : 'smime' | undef - authentification

OUT : 'unknown_list' | 'not_allowed' | 1 | undef

help()

Sends the help file for the software by using template 'helpfile'.

IN :

1. : ?
2. `robot`(+) : robot

OUT : 1 | undef

lists()

Sends back the list of public lists on this node by using template 'lists'.

IN :

1. : ?
2. `robot`(+) : robot

OUT : 1 | undef

which()

Sends back the list of lists that sender is subscribed to. If he is owner or editor, managed lists are noticed. Message is sent by using template 'which'.

IN :

1. : ?
2. `robot`(+) : robot

OUT : 1

finished()

Called when 'quit' command is found. It sends a notification to sender : no process will be done after this line.

IN : -

OUT : 1

29.4.2 tools for command processing

get_auth_method()

get_auth_method()

Called by processing command functions to return the authentification method and to check the key if it is 'md5' method.

IN :

1. `cmd`(+) : requesting command
2. `email`(+) : used to compute auth if needed in command
3. `error`(+) : ref(HASH) - keys are :
   – `type` : $type for "message_report" template parsing
   – `data` : ref(HASH) for "message_report" template parsing
   – `msg` : for do_log()
4. `sign_mod`(+) : 'smime' - smime authentification | undef - smtp or md5 authentification
5. `list` : ref(List) | undef - in a list context or not

OUT : 'smime' | 'md5' | 'smtp' - authentification method if checking not failed | undef

29.5 wwsympa.fcgi

This script provides the web interface to *Sympa*.

do_subscribe(), do_signoff(), do_add(), do_del(), do_change_email(), do_reject(), do_send_mail(), do_sendpasswd(), do_remind(), do_set(), do_send_me(), do_request_topic(), do_tag_topic_by_sender().

do_subscribe()

Subscribes a user to a list. New subscriber can be notified by sending him template 'welcome'.

- **IN** : -
- **OUT** : 'subrequest' | 'login' | 'info' | $in.previous_action | undef

do_signoff()

Unsubscribes a user from a list. The unsubscriber can be notified by sending him template 'bye'.
- **IN** : -
- **OUT** : 'sigrequest' | 'login' | 'info' | undef

do_add()

Adds a user to a list (requested by another user) and can send acknowledgements. New subscriber can be notified by sending him template 'welcome'.
- **IN** : -
- **OUT** : 'loginrequest' | ($in.previous_action || 'review') | undef

do_del()

Removes a user from a list (requested by another user) and can send acknowledgements. Unsubscriber can be notified by sending template 'removed'.
- **IN** : -
- **OUT** : 'loginrequest' | ($in.previous_action || 'review') | undef

do_change_email()

Changes a user's email address in *Sympa* environment. Password can be send to user by sending template 'sendpasswd'.
- **IN** : -
- **OUT** : '1' | 'pref' | undef

do_reject()

Refuses and deletes moderated messages. Rejected message senders are notified by sending them template 'reject'.
- **IN** : -
- **OUT** : 'loginrequest' | 'modindex' | undef

do_distribute()

Distributes moderated messages by sending a command DISTRIBUTE to sympa.pl. For it, it calls mail : :mail_file() (see 29.1.1, page 262). As it is in a Web context, the message will be set in spool. In a context of message topic, tags the message by calling to function List : :tag_topic() (see 29.2.5, page 274).
– **IN** : -
– **OUT** : 'loginrequest' | 'modindex' | undef

do_modindex()

Allows a moderator to moderate a list of messages and documents and/or tag message in message topic context.
– **IN** : -
– **OUT** : 'loginrequest' | 'admin' | 1 | undef

do_viewmod()

Allows a moderator to moderate a message and/or tag message in message topic context.
– **IN** : -
– **OUT** : 'loginrequest' | 1 | undef

do_send_mail()

Sends a message to a list by the Web interface. It uses mail : :mail_file() (see 29.1.1, page 262) to do it. As it is in a Web context, the message will be set in spool.
– **IN** : -
– **OUT** : 'loginrequest' | 'info' | undef

do_sendpasswd()

Sends a message to a user, containing his password, by sending him template 'send-passwd' list by the Web interface.
– **IN** : -
– **OUT** : 'loginrequest' | 'info' | undef

do_request_topic()

Allows a sender to tag his mail in message topic context.
- **IN** : -
- **OUT** : 'loginrequest' | 1 | undef

do_tag_topic_by_sender()

Tags a message by its sender by calling List : :tag_topic() and allows its diffusion by sending a command CONFIRM to sympa.pl.
- **IN** : -
- **OUT** : 'loginrequest' | 'info' | undef

do_remind()

Sends a command remind to sympa.pl by calling mail : :mail_file() (see 29.1.1, page 262). As it is in a Web context, the message will be set in spool.
- **IN** : -
- **OUT** : 'loginrequest' | 'admin' | undef

do_set()

Changes subscription options (reception or visibility)
- **IN** : -
- **OUT** : 'loginrequest' | 'info' | undef

do_send_me()

Sends a web archive message to a requesting user It calls mail : :mail_forward() to do it (see 29.1.1, page 263). As it is in a Web context, the message will be set in spool.
- **IN** : -
- **OUT** : 'arc' | 1 | undef

29.6 report.pm

This module provides various tools for notification and error reports in every Sympa interface (mail diffusion, mail command and web command).

For a requested service, there are four kinds of reports to users :

- **success notification**
  when the action does not involve any specific mail report or else, the user is notified of the well done of the processus.
- **non authorization** `(auth)`
  a user is not allowed to perform an action, Sympa provides reason of rejecting. The template used to provides this information is `mail_tt2/authorization_reject.tt2`. It contains a list of reasons, indexed by keywords that are mentioned in reject action scenario (see 14.1, page 144)
- **user error** `(user)`
  a error caused by the user, the user is informed about the error reason
- **internal server error** `(intern)`
  an error independent from the user, the user is succintly informed about the error reason but a mail with more information is sent to listmaster using template `mail_tt2/listmaster_notification.tt2`(If it is not necessary, keyword used is `'intern_quiet'`.

For other reports than non authorizations templates used depends on the interface :

- message diffusion : `mail_tt2/message_report.tt2`
- mail commands : `mail_tt2/command_report.tt2`
- web commands : `web_tt2/notice.tt2` for positive notifications and `web_tt2/error.tt2` for rejects.

29.6.1 Message diffusion

These reports use template `mail_tt2/message_report.tt2` and there are two functions : `reject_report_msg()` and `notice_report_msg()`.

reject_report_msg()

Sends a notification to the user about an error rejecting his requested message diffusion.

IN :

1. `type`(+) : 'intern'|'intern_quiet'|'user'|'auth' - the error type
2. `error` : SCALAR - depends on $type :
   - 'intern' : string error sent to listmaster
   - 'user' : $entry in `message_report.tt2`
   - 'auth' : $reason in `authorization_reject.tt2`
3. `user`(+) : SCALAR - the user to notify
4. `param` : ref(HASH) - for variable instantiation `message_report.tt2` (key `msgid`(+) is required *if type == 'intern'*)
5. `robot` : SCALAR - robot
6. `msg_string` : SCALAR - rejected message
7. `list` : ref(List) - in a list context

OUT : 1 | undef

notice_report_msg()

Sends a notification to the user about a success about his requested message diffusion.

IN :

1. `entry`(+) : $entry in `message_report.tt2`
2. `user`(+) : SCALAR - the user to notify
3. `param` : ref(HASH) - for variable instantiation `message_report.tt2`
4. `robot`(+) : SCALAR - robot
5. `list` : ref(List) - in a list context

OUT : 1 | undef

29.6.2 Mail commands

A mail can contains many commands. Errors and notices are stored in module global arrays before sending (intern_error_cmd, user_error_cmd, global_error_cmd, auth_reject_cmd, notice_cmd). Moreover used errors here we can have global errors on mail containing commands, so there is a function for that. These reports use template `mail_tt2/command_report.tt2` and there are many functions :

init_report_cmd()

Inits global arrays for mail command reports.

IN : -

OUT : -

is_there_any_report_cmd()

Looks for some mail command reports in one of global arrays.

IN : -

OUT : 1 *if there are some reports to send*

global_report_cmd()

Concerns global reports of mail commands. There are many uses cases :

1. **internal server error** for a differed sending at the end of the mail processing :
   - `global_report_cmd('intern',$error,$data,$sender,$robot)`
   - `global_report_cmd('intern_quiet',$error,$data)` : the listmaster won't be noticied
2. **internal server error** for sending every reports directly (by calling `send_report_cmd()`) :
   - `global_report_cmd('intern',$error,$data,$sender,$robot,1)`
   - `global_report_cmd('intern_quiet',$error,$data,$sender,$robot,1)` : the listmaster won't be noticied
3. **user error** for a differed sending at the end of the mail processing :
   `global_report_cmd('user',$error,$data`
4. **user error** for sending every reports directly (by calling `send_report_cmd()`) :
   `global_report_cmd('user',$error,$data,$sender,$robot,1)`

IN :

1. `type`(+) : 'intern'|'intern_quiet'|'user'
2. `error` : SCALAR - depends on $type :
   - 'intern' : string error sent to listmaster
   - 'user' : $glob.entry in `command_report.tt2`
3. `data` : ref(HASH) - for variable instantiation in `command_report.tt2`
4. `sender` : SCALAR - the user to notify
5. `robot` : SCALAR - robot
6. `now` : BOOLEAN - send reports now *if true*

OUT : 1 | undef

reject_report_cmd()

Concerns reject reports of mail commands. These informations are sent at the end of the mail processing. There are many uses cases :

1. **internal server error** :
   - `reject_report_cmd('intern',$error,$data,$cmd,$sender,$robot)`
   - `reject_report_cmd('intern_quiet',$error,$data,$cmd)` : the listmaster won't be noticied
2. **user error** :
   `reject_report_cmd('user',$error,$data,$cmd)`
3. **non authorization** :
   `reject_report_cmd('auth',$error,$data,$cmd)`

IN :

1. `type(+)` : 'intern'|'intern_quiet'|'user'|'auth'
2. `error` : SCALAR - depends on $type :
   - 'intern' : string error sent to listmaster
   - 'user' : $u_err.entry in `command_report.tt2`
   - 'auth' : $reason in `authorization_reject.tt2`
3. `data` : ref(HASH) - for variable instantiation in `command_report.tt2`
4. `cmd` : SCALAR - the rejected command, $xx.cmd in `command_report.tt2`
5. `sender` : SCALAR - the user to notify
6. `robot` : SCALAR - robot

OUT : 1 | undef

notice_report_cmd()

Concerns positive notices of mail commands. These informations are sent at the end of the mail processing.

IN :

1. `entry` : $notice.entry in `command_report.tt2`
2. `data` : ref(HASH) - for variable instantiation in `command_report.tt2`
3. `cmd` : SCALAR - the rejected command, $xx.cmd in `command_report.tt2`

OUT : 1 | undef

send_report_cmd()

Sends the template `command_report.tt2` to $sender with global arrays and then calls to `init_report_command.tt2` function. (It is used by sympa.pl at the end of mail process if there are some reports in gloal arrays)

IN :

1. `sender(+)` : SCALAR - the user to notify
2. `robot(+)` : SCALAR - robot

OUT : 1

29.6.3 Web commands

It can have many errors and notices so they are stored in module global arrays before html sending. (intern_error_web, user_error_web, auth_reject_web, notice_web). These reports use `web_tt2/notice.tt2` template for notices and `web_tt2/error.tt2` template for rejects.

init_report_web()

Inits global arrays for web command reports.

IN : -

OUT : -

is_there_any_reject_report_web()

Looks for some rejected web command reports in one of global arrays for reject.

IN : -

OUT : 1 *if there are some reject reports to send (not notice)*

get_intern_error_web()

Return array of web intern error

IN : -

OUT : ref(ARRAY) - clone of intern_error_web

get_user_error_web()

Return array of web user error

IN : -

OUT : ref(ARRAY) - clone of user_error_web

get_auth_reject_web()

Return array of web authorisation reject

IN : -

OUT : ref(ARRAY) - clone of auth_reject_web

get_notice_web()

Return array of web notice

IN : -

OUT : ref(ARRAY) - clone of notice_web

reject_report_web()

Concerning reject reports of web commands, there are many uses cases :

1. **internal server error** :
   - `reject_report_web('intern',$error,$data,$action,$list,$user,$robot)`
   - `reject_report_web('intern_quiet',$error,$data,$action,$list)` : the listmaster won't be noticied
2. **user error** :
   `reject_report_web('user',$error,$data,$action, $list)`
3. **non authorization** :
   `reject_report_web('auth',$error,$data,$action, $list)`

IN :

1. `type`(+) : 'intern'|'intern_quiet'|'user'|'auth'
2. `error`(+) : SCALAR - depends on $type :
   - 'intern' : $error in `listmaster_notification.tt2` and possibly $i_err.msg in `error.tt2`
   - 'intern_quiet' : possibly $i_err.msg in `error.tt2`
   - 'user' : $u_err.msg in `error.tt2`
   - 'auth' : $reason in `authorization_reject.tt2`
3. `data` : ref(HASH) - for variable instantiation in `notice.tt2`
4. `action`(+) : SCALAR - the rejected actin, $xx.action in `error.tt2`, $action in `listmaster_notification.tt2`
5. `list` : " | ref(List)
6. `user` : SCALAR - the user for listmaster notification
7. `robot` : SCALAR - robot for listmaster notification

OUT : 1 | undef

notice_report_web()

Concerns positive notices of web commands.

IN :

1. `msg` : $notice.msg in `notice.tt2`
2. `data` : ref(HASH) - for variable instantiation in `notice.tt2`
3. `action` : SCALAR - the noticed command, $notice.cmd in `notice.tt2`

OUT : 1 | undef

29.7 tools.pl

This module provides various tools for Sympa.

checkcommand()

Checks for no command in the body of the message. If there are some command in it, it returns true and sends a message to $sender by calling List : :send_global_file() (see 29.2.2, page 268) with mail template "message_report".

IN :

1. `msg`(+) : ref(MIME : :Entity) - the message to check
2. `sender`(+) : the message sender
3. `robot`(+) : robot

OUT :
– 1 *if there are some command in the message*
– 0 *else*

get_array_from_splitted_string()

Return an array made from a string splitted by ','. It removes spaces.

IN : `string`(+) : string to split

OUT : ref(ARRAY) -

diff_on_arrays()

Makes set operation on arrays seen as set (with no double) :

IN :

1. `A`(+) : ref(ARRAY) - set
2. `B`(+) : ref(ARRAY) - set

OUT : ref(HASH) with keys :
– deleted : A $\setminus$ B
– added : B $\setminus$ A
– intersection : A $\cap$ B
– union : A $\cup$ B

clean_msg_id()

Cleans a msg_id to use it without '
n', '
s', ¡ and ¿.

IN : `msg_id`(+) : the message id

OUT : the clean msg_id

clean_email()

Lower-case it and remove leading and trailing spaces.

IN : `msg_id`(+) : the email

OUT : the clean email

make_tt2_include_path()

Make an array of include path for tt2 parsing

IN :

1. `robot`(+) : SCALAR - the robotset
2. `dir` : SCALAR - directory ending each path
3. `lang` : SCALAR - for lang directories

4. `list` : ref(List) - for list directory

OUT : ref(ARRAY) - include tt2 path, respecting path priorities.

29.8 Message.pm

This module provides objects to encapsulate file message in order to prevent it from its alteration for using signatures.

new()

Creates an object Message and initialize it :
– `msg` : ref(MIME : :Entity)
– `altered` *if the message is altered*
– `filename` : the file containing the message
– `size` : the message size
– `sender` : the first email address, in the 'From' field
– `decoded_subject` : the 'Subject' field decoded by MIME : :Words : :decode_mimewords
– `subject_charset` : the charset used to encode the 'Subject' field
– `rcpt` : the 'X-Sympa-To' field
– `list` : ref(List) *if it is a message no addressed to Sympa or a listmaster*
– `topic` : the 'X-Sympa-Topic' field.
– *in a 'openssl' context - decrypt message* :
  – `smime_crypted` : 'smime_crypted' *if it is in a 'openssl' context*
  – `orig_msg` : ref(MIME : :Entity) - crypted message
  – `msg` : ref(MIME : :Entity) - decrypted message (see tools : :smime_decrypt())
  – `msg_as_string` : string - decrypted message (see tools : :smime_decrypt())
– *in a 'openssl' context - check signature* :
  – `protected` : 1 *if the message should not be altered*
  – `smime_signed` : 1 *if the message is signed*
  – `smime_subject` : ref(HASH)*if the message is signed* - information on the signer see tools : :smime_parse_cert().

IN :

1. `pkg`(+) : Message
2. `file`(+) : the message file

OUT : ref(Message) | undef

dump()

Dump the message object in the file descriptor $output

IN :

1. `self`(+) : ref(Message)
2. `output`(+) : file descriptor

OUT : '1'

add_topic()

Adds the message topic in the Message object (`topic'` and adds the 'X-Sympa-Topic' field in the ref(MIME : :Entity) `msg'`.

IN :

1. `self`(+) : ref(Message)
2. `topic`(+) : string splitted by ',' - list of topic

OUT : '1'

get_topic()

Returns the topic(s) of the message

IN : `self`(+) : ref(Message)

OUT : " *if no message topic* — string splitted by ',' *if message topic*

N.B. :
– (+) : required parameter, value must not be empty
– | : "or" for parameters value
– $: reference to code parameters or variables
– *condition for parameter*

Index