

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, 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 to 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 handle large lists: for a list of 20,000 subscribers, it requires less than 6 minutes to send a message to 95% 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 (5.4) of Sympa [<http://www.sympa.org>].

License

Sympa is free software; you may distribute it under the terms of the GNU General Public License Version 2 [<http://www.gnu.org/copyleft/gpl.html>].

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.

Features

Sympa provides all the basic features that any mailing list management software 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. These features are:

- **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) and the online help are internationalized. Translations are gathered in standard PO files;
- **Template-based user interface.** Every web page and service message can be customized through the TT2 template format;
- **MIME support.** Sympa naturally respects MIME in the distribution process, and in addition it 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`);
- **Fine control of authorizations.** The rights to perform controlled actions (such as sending a message, subscribe to a list, etc.) are set using an [authorization scenarios](#). Any listmaster can define new authorization scenarios in order to complement the 20 predefined configurations included in the distribution (Example: forward multipart messages to the list editor, while distributing others without requiring any further authorization);
- **Privileged operations** can be performed by list editors or list owners (or any other user category), as defined in the list `config` file or by the robot administrator, the listmaster, defined in the `/etc/sympa.conf` global configuration file (a listmaster can also be defined for a particular virtual host). Privileged operations include the usual `ADD`, `DELETE` and `REVIEW` commands, which can be authenticated through a unique password or an S/MIME signature;
- **Web interface:** *WWSympa* is a global Web interface to all Sympa functions (including administration). It provides:
 - a classification of lists, along with a search index,
 - an 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),
 - the management of shared documents (download, upload, specific access control for each document),
 - an HTML presentation personalized for each user with the list of his/her current subscriptions, including access to message archives, subscription options, etc

- management tools for list managers (bounce processing, changing of list parameters, moderating incoming messages),
- tools for the robot administrator (list creation, global robot configuration);
To know more, refer to [WWSympa, Sympa's web interface](#).
- **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). To know more, refer to [Sympa and its database](#);
- **Virtual hosting**: a single Sympa installation can provide multiple virtual robots with both email and web interface customization (see [Virtual host](#));
- **LDAP-based mailing lists**: e-mail addresses can be retrieved dynamically from a database accepting SQL queries, or from an LDAP directory. In order to maintain reasonable response times, Sympa retains the data source in an internal cache controlled by a TTL (Time To Live) parameter (see [include-ldap-query](#));
- **LDAP authentication**: via uid and emails stored in LDAP Directories. Alternative email addresses, extracted from a LDAP directory, may be used to "unify" subscriptions (see [Authentication with email address, uid or alternate email address](#));
- **Antivirus scanner**: Sympa extracts attachments from incoming messages and runs a virus scanner on them. Currently working with McAfee/uvscan, Fsecure/fsav, Sophos, AVP, Trend Micro/VirusWall and Clam Antivirus (see [Antivirus](#));
- **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"
- **RSS channel**.
- **Custom user attributes**: you can customize the subscription form by requesting additional informations to future users when they request their subscription. See the doc regarding these [custom attributes](#).

Project directions

Sympa is a very active project: check the [release notes](#). Thus it is not possible to maintain multiple documents about the Sympa project directions anymore. Please refer to the Future Sympa developments page for information about the project directions.

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 able to handle. 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 dates:

- Mar 1999 Internal use of a database (MySQL), definition of list subscriber with external data source (RDBMS or LDAP).
- Oct 1999 Stable version of WWSympa, introduction of authorization scenarios.
- Feb 2000 Web bounce management.
- Apr 2000 Archive 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 plugin.
- Jan 2002 Virtual hosting, LDAP authentication.
- Aug 2003 Automatic bounce management.
- Sep 2003 CAS-based 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, ...
- 2006 full UTF-8 support
- 2007 automatic list creation
- 2008 Web sessions, Custom user attributes

Mailing lists and support

If you wish to contact the authors of Sympa, please use the address sympa-authors@cr.ufr.fr.

There are also a few mailing-lists about Sympa [<http://listes.cru.fr/sympa/lists/informatique/sympa>]:

- `sympa-users(@)cru.fr` general information list
- `sympa-fr(@)cru.fr`, for French-speaking users
- `sympa-announce(@)cru.fr`, Sympa announcements
- `sympa-dev(@)cru.fr`, Sympa developers
- `sympa-pootle(@)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 last name).

You may also refer to the Sympa homepage [<http://www.sympa.org>]; there you will find the latest version [[http://www.sympa.org/distribution/latest version](http://www.sympa.org/distribution/latest%20version)], the FAQ and so on.

Organization

Here is a snapshot of what Sympa looks like once installed on your system. This also illustrates the Sympa philosophy, we guess. Almost all configuration files can be defined for a particular list, for a virtual host or for the entire site, and most of them have a reasonable default value provided by Sympa distribution.

The following reference manual assumes a particular location for all files and directories. Note that binary distributions usually change those locations according to the operating system file organization. When installing Sympa from source kit, `configure` can be called with command options in order to change all default file locations.

- `/home/sympa`
The root directory of Sympa. You will find almost everything related to Sympa under this directory, except logs and main configuration files.
- `/home/sympa/bin`
This directory contains the binaries, including the CGI. It also contains the default authorization scenarios, templates and configuration files as in the distribution. `/home/sympa/bin` may be completely overwritten by the `make install` so you must not customize templates and authorization scenarios under `/home/sympa/bin`.
- `/home/sympa/bin/etc`
Here Sympa stores the default versions of what it will otherwise find in `/home/sympa/etc` (task models, authorization scenarios, templates and configuration files, recognized S/Mime certificates, families).
- `/home/sympa/etc`
This is your site's configuration directory. Consult `/home/sympa/bin/etc` when drawing up your own.
- `/home/sympa/etc/create_list_templates/`
List templates (suggested at list creation time).
- `/home/sympa/etc/scenari/`
This directory will contain your authorization scenarios. If you don't know what the hell an authorization scenario is, refer to [Authorization scenarios](#). Those authorization scenarios are default scenarios but you may look at `/home/sympa/etc/my.domain.org/scenari/` for default scenarios of `my.domain.org` virtual host and `/home/sympa/expl/mylist/scenari` for scenarios specific to a particular list.
- `/home/sympa/etc/data_sources/`
This directory will contain your `.incl` files (see [Data inclusion file](#)). At the moment it only deals with files required by paragraphs `owner_include` and `editor_include` in the config file.
- `/home/sympa/etc/list_task_models/`
This directory will store your own list task models (see [Customizing tasks](#)).
- `/home/sympa/etc/global_task_models/`
Contains your global task models (see [Customizing tasks](#)).
- `/home/sympa/etc/web_tt2/` (used to be `/home/sympa/etc/wws_templates/`)
The web interface (*WWSympa*) is composed of template HTML files parsed by the CGI program. Templates can also be defined for a particular list in `/home/sympa/expl/mylist/web_tt2/` or in `/home/sympa/etc/my.domain.org/web_tt2/`
- `/home/sympa/etc/mail_tt2/` (used to be `/home/sympa/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.
- `/home/sympa/etc/families/`
Contains your family directories (see [Mailing list creation](#)). Family directories can also be created in `/home/sympa/etc/my.domain.org/families/`
- `/home/sympa/etc/my.domain.org`
The directory host define the virtual host `my.domain.org` dedicated to management of all lists of this domain (list description of `my.domain.org` are stored in

/home/sympa/expl/my.domain.org). Those directories for virtual hosts have the same structure as /home/sympa/etc which is the configuration dir of the default robot.

- /home/sympa/expl
Sympa's working directory.
- /home/sympa/expl/mylist
The list directory (refer to [Mailing list definition](#)). Lists stored in this directory belong to the default robot as defined in sympa.conf file, but a list can be stored in /home/sympa/expl/my.domain.org/mylist directory and it is managed by my.domain.org virtual host.
- /home/sympa/expl/X509-user-certs
The directory where Sympa stores all user's certificates.
- /home/sympa/locale
Internationalization directory. It contains message catalogues in the GNU .po format.
- /home/sympa/spool
Sympa uses 9 different spools (see [Spools](#)).
- /home/sympa/src/
Sympa sources.

Programs

- `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 files 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.

Configuration files

- `/etc/sympa.conf`
The main configuration file. See [Sympa.conf parameters](#).
- `/etc/wwsympa.conf`
WWSympa configuration file. See [the description of WWSympa](#).
- `edit_list.conf`
Defines which parameters/files are editable by owners. See [List editing](#).
- `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 [Topics](#).
- [auth.conf](#)
Defines authentication backend organization (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).
- `crawlers_detection.conf`
This file specifies how Sympa detects web crawlers. It is used in order to optimize the Sympa web interface responses and internal mechanisms for crawlers. In this version the file is limited to a list of user agent strings, but in the future it may be enriched with IP addresses. When a crawler is detected, Sympa allows the web client to cache pages so crawlers should not browse old archives every day. In addition, Sympa does not create http sessions for crawlers. This keeps the Sympa session table quite

small.

- `nrcpt_by_domain.conf`

This file is used to limit the number of recipients per SMTP session. Some ISPs trying to block spams reject sessions with too many recipients. In such cases you can set the `nrcpt` `robot.conf` parameter to a lower value but this will affect all SMTP sessions with any remote MTA. This file is used to limit the number of recipients for some specific domains. The file must contain a list of domains followed by the maximum number of recipients per SMTP session. Example:

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

- `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.

- `ldap_alias_manager.conf`

This file defines the parameters for a LDAP directory, when using `ldap_alias_manager.pl` as the mail aliases management script.

Spools

See [Spool related](#) for spool definition in `sympa.conf`.

- `/home/sympa/spool/auth/`
For storing messages until they have been confirmed. Files are created and processed by the `sympa.pl` program.
- `/home/sympa/spool/bounce/`
For storing incoming bouncing messages. Files are created by the `bouncequeue` program (via mail aliases) and processed by the `bounced.pl` daemon.
- `/home/sympa/spool/bounce/bad/`
For storing bouncing messages for which bounce management failed, though an user was identified. Files are moved there by the `bounced.pl` daemon.
- `/home/sympa/spool/bounce/OTHER/`
Stores bouncing messages for which Sympa couldn't determine the original sender. Files are moved there by the `bounced.pl` daemon.
- `/home/sympa/spool/digest/`
For storing message digests before they are sent. Files are created and processed by the `sympa.pl` daemon.
- `/home/sympa/spool/mod/`
For storing unmoderated messages. Files are created by the `sympa.pl` program and processed by either `sympa.pl` or `wwsympa.fcgi`.
- `/home/sympa/spool/msg/`
For storing incoming messages (including commands). Files are created by the `queue` program (via mail aliases) and processed by the `sympa.pl` program.
- `/home/sympa/spool/msg/bad/`
Sympa stores rejected messages in this directory. Files are created by the `sympa.pl` daemon.
- `/home/sympa/spool/distribute/`
For storing messages ready for distribution. This spool is used only if the installation runs 2 `sympa.pl` daemons, one for commands, one for messages.
- `/home/sympa/spool/distribute/bad/`
Sympa stores rejected messages in this directory. Files are created by the `sympa.pl` process dedicated to message distribution.
- `/home/sympa/spool/task/`
For storing all tasks created. Files are created and processed by the `task_manager.pl` daemon.
- `/home/sympa/spool/outgoing/`
`sympa.pl` dumps messages in this spool to await archiving by `archived.pl`. `wwsympa.fcgi` may also create files in this spool.
- `/home/sympa/spool/outgoing/bad/`
For storing messages which couldn't be archived. Files are moved there by the `archived.pl` daemon.
- `/home/sympa/spool/topic/`
For storing topic information files.
- `/home/sympa/spool/tmp/`
For storing temporary information, as `stderr` flux from processes or message parts submitted to the anti-virus

Roles and privileges

You can assign roles to users (identified via their email addresses) at different levels in Sympa; privileges

are associated (or can be associated) to these roles. We list these roles below (from the most powerful to the least), along with the relevant privileges.

(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.

(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 the Sympa web interface. Listmasters also inherit the privileges of list owners (for any list defined in the virtual host), but not the moderator privileges.

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) owner privileges and are also responsible for managing the list owners and editors themselves (through the web interface). With Sympa's 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.

(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.

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.

Subscribers (or list members)

Subscribers are the people who are members of a mailing list; they either subscribed, or got added directly by the listmaster or via a data source (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 are not hard coded in Sympa but expressed via the so-called authorization scenarios (see [Scenarios](#)).

Installing Sympa

Most pieces of Sympa are written in Perl. It also includes a few dedicated programs written in C.

Obtaining Sympa, related links

The Sympa distribution is available from <http://www.sympa.org> [<http://www.sympa.org>]. All important resources are available there:

- sources;
- release notes;
- `.rpm` and `.deb` packages for Linux;
- mailing list about Sympa (see [Mailing lists and support](#));
- contributions;
- ...

Migrating from another software

If you are moving your mailing list software from another software (majordomo, mailman, listserv, lyris, etc) you will need to migrate your data (list configuration, list members, list archives) to the Sympa format. You can benefit from scripts written by other Sympa users and listed on our [contributions page](#).

If you extend/fix the scripts, please submit the new version.

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.

- installing a RDBMS (Oracle, MySQL(version 4.1 minimum), SQLite, Sybase or PostgreSQL) and creating Sympa's Database. This is required for using the web interface for Sympa. Please refer to ([Sympa and its database](#)).
- installing an MTA (Message Transfer Agent): sendmail, postfix, qmail or exim
- installing a web server (Apache being the most commonly used)
- installing libxml 2 [<http://xmlsoft.org/>], required by the LibXML Perl module;
- installing the `gettext-devel` library;
- installing CPAN CPAN (Comprehensive Perl Archive Network) [<http://www.perl.com/CPAN>] modules;
- creation of a dedicated UNIX user.

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.

For remarks regarding problems specific to your OS, please refer to the FAQ.

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 [<ftp://ftp.oleane.net/pub/mirrors/gnu/>].

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 [<ftp://ftp.oleane.net/pub/mirrors/sendmail-ucb/>] or a more recent release. You may also use `postfix` or `qmail`.

Installing Perl and CPAN modules

To be able to use Sympa you must have release 5.8 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 multiplexer [<http://www.perl.com/CPAN/src/latest.tar.gz>]; it will find one for you.

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 release to another, the following list of modules might not be up to date:

- CGI;
- CipherSaber;
- DB_File;
- DBD;
- DBI;
- Digest-MD5;
- Encode;
- FCGI;
- File-Spec;
- IO-stringy;
- libintl-perl;
- libwww-perl;
- MailTools;

- MHonArc;
- MIME-Charset;
- MIME-EncWords;
- MIME-tools;
- MIME-Base64;
- Regexp-Common;
- Template-Toolkit;
- XML-LibXML.

Since release 2, Sympa requires an RDBMS to work properly. It stores user subscriptions and preferences in a database. Sympa is also able to extract user data from 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):

- DBD (DataBase Driver) related to your RDBMS (e.g. Mysql-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);
- If you want to Download Zip files of list's Archives, you'll need to install the Perl Module for Archive Management:
Archive::Zip.
- SOAP-Lite is required if you are running the Sympa SOAP server.
- File-NFSLock' is required to have NFS locking support.

Creating 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 `/home/sympa`.

Creating the database

See [creating a sympa database](#)

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 multilingual messages are stored; and, of course, the `src/` directory for the mail robot and the `wwsympa` directory for the web interface.

Example:

```
-----
# su -c "gzip -dc sympa-5.4.x.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 home directory (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 Red Hat;
- `%-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-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=LOGIN%`, 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 overwrite 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`);

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

`make install` does the installation job. 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 [color parameters](#)).

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 necessary to have Sympa run correctly.

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 [Organization](#).

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. You can use restrictive authorizations if you like, since only programs running with the `sympa` account will need to access them.

Robot aliases

See [Robot aliases](#).

Web setup

See [Web server setup](#)

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* login behavior 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. You can test log feature by using `testlogs.pl`.

If your system is running `syslog-ng`, add these lines to your `syslog-ng.conf` (in some cases, `syslog-ng.conf.in`):

```
destination sympa { file ("/var/log/sympa") ; };
log { source(src); filter(f_sympa); destination(sympa); };
filter f_sympa { facility(local1) and match('sympa'); };
```

and restart `syslog`.

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:

- `—add_list familyname —robot robotname —input_file /path/to/list_file.xml`
Adds the list described in the XML file to the *familyname* family. See: [Adding a list to a list family](#).
- `—close_family familyname —robot robotname`
Closes the *familyname* family. See: [List family closure](#).
- `—close_list listname@robot`
Closes the list (changing its status to closed), removes aliases (if `sendmail_aliases` parameter was set) and removes subscribers from DB (a dump is created in the list directory to allow the list restoration). When you are in a family context, refer to: [List family closure](#).
- `—close_unknown`
When instantiating a family, this option tells Sympa to silently close lists unknown to the family.
- `—config config_file | -f config_file`
Forces Sympa to use an alternative configuration file. The default behavior is to use the configuration file as defined in the Makefile (`$CONFIG`).
- `—create_list —robot robotname —input_file /path/to/list_file.xml`
Creates the list described by the xml file, see: [List creation on command line with sympa.pl](#).
- `—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.
- `—dump listname | ALL`
Dumps subscribers, either of the list *listname* or of all lists. Subscribers are dumped in `subscribers.db.dump`.
- `—foreground`
the process remains attached to the TTY
- `—help | -h`
Prints `sympa.pl` usage.
- `—import listname`
Imports subscribers in the *listname* list. Data are read from STDIN.
- `—instantiate_family familyname —robot robotname —input_file /path/to/family_file.xml`
Instantiates the family *familyname*. See [Lists families](#).
- `—keepcopy recipient_directory | -k recipient_directory`
Tells Sympa to keep a copy of every incoming message instead of deleting them. *recipient_directory* is the directory to store messages.
- `—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.
- `—lowercase`
Lowercases e-mail addresses in database.
- `—mail | -m`
Sympa will log calls to `sendmail`, including recipients. Useful to keep track of each mail sent (log files may grow faster though).
- `—make_alias_file`
Creates an aliases file in `/tmp/` with all list aliases (only lists whose status is 'open'). It uses the

list_aliases.tt2 template.

- `—modify_list familyname —robot robotname —input_file /path/to/list_file.xml`
Modifies the existing family list, with description contained in the XML file. See: [Modifying a family list](#).
- `—quiet`
When instantiating a family, this option tells Sympa to skip output to STDOUT.
- `—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.
- `service process_command | process_message | process_creation`
Sets Sympa daemon to process only message distribution (`process_message`) or only commands (`process_command`) or list creation requests (`process_creation`).
- `—sync_include listaddress`
Triggers an update of list members, useful if the list uses external data sources.
- `—upgrade —from=X —to=Y`
Runs Sympa maintenance script to upgrade from version X to version Y.
- `—version | -v`
Prints current version of Sympa.

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 `wwwsympa.fcgi`.

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

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 immediately whatever a distribution message is in progress. In this case, when `sympa.pl` restarts, messages will be distributed many times.

kill -HUP

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

Upgrading Sympa

Sympa upgrade is a relatively riskless operation, 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).

Upgrading Sympa means that you follow these steps:

1. [retrieve](#) the latest source version of Sympa

1. stop Sympa

1. [install it](#) :

```
./configure ; make ; make install
```

1. run the following command.

```
sympa.pl --upgrade
```

And that' it!

This command will perform all the required DB changes (if running MySQL) and will update the configuration files if required.

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).

For example, note that, starting from Sympa 5.3b.4, the minimum version for MySQL is 4.1.

Sympa is a long-term project, so some major changes may need some extra work. The following list consists of well known changes that require some attention:

- version 5.1 (August 2005) uses XHTML and CSS in web templates;
- version 4.2b3 (August 2004) introduces TT2 template format;
- version 4.0a5 (September 2003) changes `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` lists all changes and of course, all changes that may require some attention from the installer. As mentioned 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 with MySQL), adding the following table (MySQL example):
***** 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 email address in a LDAP directory
<RETURN>
    
```

CPAN modules update

The installation of required and optional Perl modules (CPAN) 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:
1. -----
perl module      from CPAN      STATUS
-----
1. -----
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:
-----
1. -----
perl module from CPAN STATUS
-----
1. -----
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, <STDIN> line 1.
-----
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 ?
-----

```

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 mentioned in the NEWS file (database structure is describe in the `src/etc/script/` directory of distribution).

Output of Sympa logs :

```

-----
Table admin_table created in database sympa
-----
Field 'comment_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field comment_admin added to table admin_table
-----
Field 'date_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field date_admin added to table admin_table
-----
Field 'include_sources_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field include_sources_admin added to table admin_table
-----
Field 'included_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field included_admin added to table admin_table
-----
Field 'info_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field info_admin added to table admin_table
-----
Field 'list_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field list_admin added to table admin_table
-----

```

```

Field 'profile_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field profile_admin added to table admin_table
-----
Field 'reception_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field reception_admin added to table admin_table
-----
Field 'role_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field role_admin added to table admin_table
-----
Field 'subscribed_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field subscribed_admin added to table admin_table
-----
Field 'update_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
Field update_admin added to table admin_table
-----
Field 'user_admin' (table 'admin_table' ; database 'sympa') was NOT found. Attempting to add it...
-----
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
-----

```

Preserving your customizations

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

Running two 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 \
1. --with-confdir=/home/sympa-dev/etc \
1. --with-mandir=/home/sympa-dev/man \
1. --with-initdir=/home/sympa-dev/init \
-----
--with-piddir=/home/sympa-dev/pid
-----
1. --with-lockdir=/home/sympa-dev/lock \
1. --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`.

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 `/home/sympa/expl` directory that contains list config;
- the directory that contains the spools;
- the directory `/etc/sympa.conf` and `wwsympa.conf`. Sympa new installation creates a file

`/etc/sympa.conf` (see `sympa.conf` parameters) and randomly initializes 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 might have problems!

- the web archive.

In some cases, 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` processes on both servers;
1. transfer the database;
1. edit the `/data_structure.version` on the new server; change the version value to reflect the old number;
1. start "`sympa.pl -upgrade`", it will upgrade the database structure according to the hop you do.

Mail aliases

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

SMRSH

If using `sendmail`, maybe it is configured to use the secured shell `smrsh`.

```
# grep smrsh /etc/mail/sendmail.mc
FEATURE(`smrsh', `~/usr/sbin/smrsh')dnl
```

`Smrsh` obliges you to copy all the programs that are called from the mail aliases into the dedicated `/etc/smrsh` directory. Therefore you will have to tell Sympa that binaries should be installed in the `/etc/smrsh` directory. This can be performed via the `--with-bindir` option of `sympa's configure`:

```
./configure --with-bindir=/etc/smrsh
```

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. When receiving the message, the SMTP server 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 to 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: "| /home/sympa/bin/queue sympa@my.domain.org"
listmaster: "| /home/sympa/bin/queue listmaster@my.domain.org"
bounce+*: "| /home/sympa/bin/bouncequeue sympa@my.domain.org"
abuse-feedback-report: "| /home/sympa/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 host](#)).

`sympa-request` should be the address of the robot administrator, i.e. a person who manages 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 active. It is useful if `welcome_return_path` `unique` or `remind_return_path` `unique` or the `verp_rate` parameter is not 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 users can complain about spam. It is mainly used by AOL.

Don not 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

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 [Alias manager](#)), then Sympa will install automatically the following aliases for you.

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

<code>mylist:</code>	<code> /home/sympa/bin/queue mylist@my.domain.org</code>
<code>mylist-request:</code>	<code> /home/sympa/bin/queue mylist-request@my.domain.org</code>
<code>mylist-editor:</code>	<code> /home/sympa/bin/queue mylist-editor@my.domain.org</code>
<code>mylist-owner:</code>	<code> /home/sympa/bin/bouncequeue mylist@my.domain.org</code>
<code>mylist-subscribe:</code>	<code> /home/sympa/bin/queue mylist-subscribe@my.domain.org</code>
<code>mylist-unsubscribe:</code>	<code> /home/sympa/bin/queue mylist-unsubscribe@my.domain.org</code>

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

Similarly, the address `mylist-editor` can be used to contact the list editors, if defined in `/home/sympa/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 [return path suffix](#)). The `bouncequeue` program stores these messages in the `queuebounce` directory. [WWSympa](#) 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!

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. To use a different alias management tool (`ldap_alias_manager.pl` for example), you should edit the `alias_manager` `sympa.conf` parameter (see [alias manager](#)).

The script expects the following arguments :

1. `add | del`
1. `<list name>`
1. `<list domain>`

Example:

```
/home/sympa/bin/alias_manager.pl add mylist cru.fr
```

`/home/sympa/bin/alias_manager.pl` works on the alias file (as defined in `sympa.conf`) through 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.

`/home/sympa/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 (see [list_aliases.tt2](#)).

Note that you do not need alias management if you use MTA functionalities such as Postfix' `virtual_transport`. Then you can disable alias management in Sympa by positioning the `sendmail_aliases` parameter to `none`.

Ludovic Marcotte has written a version of `ldap_alias_manager.pl` that is LDAP enabled. This

script is distributed with Sympa distribution. The script has later been extended by Philippe Baumgart, British Telecom. You can customize the LDAP parameters via the `ldap_alias_manager.conf` file.

Virtual domains

When using virtual domains with `sendmail` or `postfix`, you can not refer to `mylist@my.domain.org` on the right-hand side of a `/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: /home/sympa/bin/queue
sympa@my.domain.org ..... my.domain.org-listA: /home/sympa/bin/queue listA@my.domain.org
```

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

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

Internal mail routing

If your Sympa server does virtual hosting, then it needs to perform mail routing internally. Here is how mail routing is handled :

- incoming messages are spooled with a file name (and a X-Sympa-To SMTP header) that corresponds to the `queue` program parameter defined in the mail aliases ;
- when processed by the `sympa.pl` process, Sympa determines the current virtual host by comparing the domain part of the file name with the `domain` parameter defined in the `sympa.conf` or in the `robot.conf` files of each virtual host.

Therefore you should ensure that the domain used as a parameter to the `queue` program in mail aliases corresponds to the `domain` configuration parameter of the virtual host.

sympa.conf parameters

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

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

keyword value

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

- Presentation
 - Presentation
 - License
 - Features
 - Project directions
 - History
 - Mailing lists and support
- Organization
 - Organization
 - Programs
 - Configuration files
 - Spools
 - Roles and privileges
 - (Super) listmasters
 - (Robot) listmasters
 - Privileged list owners
 - (Basic) list owners
 - Moderators (also called Editors)
 - Subscribers (or list members)
- Installing Sympa
 - Installing Sympa
 - Obtaining Sympa, related links
 - Migrating from another software
 - Prerequisites
 - System requirements
 - Installing Perl and CPAN modules
 - Required CPAN modules

- Creating a UNIX user
- Creating the database
- Compilation and installation
 - Choosing directory locations
- Robot aliases
- Web setup
- Logs
- Running Sympa
 - sympa.pl
 - INIT script
 - Stopping Sympa and signals
- Upgrading Sympa
 - Upgrading Sympa
 - Incompatible changes
 - CPAN modules update
 - Database structure update
 - Preserving your customizations
 - Running two Sympa versions on a single server
 - Moving to another server
- Mail aliases
 - Mail aliases
 - SMRSH
 - Robot aliases
 - List aliases
 - Alias manager
 - Virtual domains
 - Internal mail routing
- sympa.conf parameters index
 - sympa.conf parameters
- sympa.conf parameters part1
 - sympa.conf parameters
 - Site customization
 - domain
 - email
 - listmaster
 - listmaster_email
 - wwsympa_url
 - soap_url
 - spam_protection javascript | at | none
 - web_archive_spam_protection javascript | at | none | cookie
 - color_0, color_1, ..., color_15
 - Obsolete color parameters
 - logo_html_definition
 - main_menu_custom_button
 - css_path
 - css_url
 - static_content_path
 - static_content_url
 - pictures_feature
 - pictures_max_size
 - cookie
 - create_list
 - automatic_list_feature
 - automatic_list_creation
 - automatic_list_removal
 - global_remind
 - allow_subscribe_if_pending
- sympa.conf parameters part2
 - sympa.conf parameters
 - Directories
 - home
 - etc
 - System related
 - syslog
 - log_level
 - log_socket_type
 - pidfile
 - pidfile_creation
 - umask
 - Sending related

- distribution_mode
- maxsmtp
- log_smtp
- use_blacklist
- max_size
- misaddressed_commands
- misaddressed_commands_regexp
- nrcpt
- avg
- alias_manager
- sendmail
- sendmail_args
- sendmail_aliases
- rfc2369_header_fields
- remove_headers
- remove_outgoing_headers
- ignore_x_no_archive_header_feature
- anonymous_headers_fields
- list_check_smtp
- list_check_suffixes
- urlize_min_size
- Bulk mailer
 - pidfile_bulk
 - sympa_packet_priority
 - bulk_fork_threshold
 - bulk_max_count
 - bulk_lazytime
 - bulk_wait_to_fork
- Quotas
 - default_shared_quota
 - default_archive_quota
- Spool related
 - spool
 - queue
 - queuedistribute
 - queuemod
 - queuedigest
 - queueauth
 - queueoutgoing
 - queuetopic
 - queuebounce
 - queuetask
 - queueautomatic
 - tmpdir
 - sleep
 - clean_delay_queue
 - clean_delay_queueoutgoing
 - clean_delay_queuebounce
 - clean_delay_queueother
 - clean_delay_queuemod
 - clean_delay_queueauth
 - clean_delay_queuesubscribe
 - clean_delay_queuetopic
 - clean_delay_queueautomatic
 - clean_delay_tmpdir
- sympa.conf parameters part3
 - sympa.conf parameters
 - Internationalization related
 - localedir
 - supported_lang
 - lang
 - web_recode_to
 - filesystem_encoding
 - Bounce related
 - verp_rate
 - welcome_return_path
 - remind_return_path
 - return_path_suffix
 - expire_bounce_task
 - purge_orphan_bounces_task
 - eval_bouncers_task
 - process_bouncers_task

- minimum_bouncing_count
- minimum_bouncing_period
- bounce_delay
- default_bounce_level1_rate
- default_bounce_level2_rate
- bounce_email_prefix
- bounce_warn_rate
- bounce_halt_rate
- default_remind_task
- Tuning
 - cache_list_config
 - lock_method
 - sympa_priority
 - request_priority
 - owner_priority
 - default_list_priority
- Database related
 - update_db_field_types
 - db_type
 - db_name
 - db_host
 - db_port
 - db_user
 - db_passwd
 - db_timeout
 - db_options
 - db_env
 - db_additional_subscriber_fields
 - db_additional_user_fields
 - purge_user_table_task
 - purge_tables_task
 - purge_logs_table_task
 - logs_expiration_period
 - purge_session_table_task
 - session_table_ttl
 - purge_challenge_table_task
 - challenge_table_ttl
- Loop prevention
 - loop_command_max
 - loop_command_sampling_delay
 - loop_command_decrease_factor
 - loop_prevention_regex
- S/MIME configuration
 - openssl
 - capath
 - cafile
 - key_passwd
- DKIM
 - DKIM_feature
 - dkim_add_signature_to
 - dkim_signature_apply_on
 - dkim_private_key_path
 - dkim_signer_domain
 - dkim_selector
 - dkim_signer_identity
 - dkim_header_list
- Antivirus plug-in
 - antivirus_path
 - antivirus_args
 - antivirus_notify
- Mailing list definition
 - Mailing list definition
 - Mail aliases
 - List configuration file
 - Examples of configuration files
 - Subscribers file
 - Info file
 - Homepage file
 - Data inclusion file
 - List template files
 - welcome.tt2

- bye.tt2
- removed.tt2
- reject.tt2
- invite.tt2
- remind.tt2
- summary.tt2
- list_aliases.tt2
- Stats file
- List model files
 - remind.annual.task
 - expire.annual.task
- Message header and footer
 - Archive directory
- List creation, edition and removal
 - List creation, editing and removal
 - List creation
 - Data for list creation
 - XML file format
 - List families
 - List creation on command line with sympy.pl
 - Creating and editing mailing lists using the Web
 - List creation on the web interface
 - Who can create lists on the web interface
 - Typical list profile and web interface
 - customize create_list_request.tt2
 - List editing
 - Removing a list
- Lists Families
 - List families
 - Family concept
 - Using family
 - Definition
 - Instantiation
 - Modification
 - Closure
 - Adding a list to a list family
 - Removing a list from a list family
 - Modifying a family list
 - Editing list parameters in a family context
 - Automatic list creation
 - Configuring your MTA
 - Defining the list family
 - Configuring Sympa
- List configuration parameters
 - List configuration parameters
- List configuration / List definition
 - List parameters: definition
 - subject
 - visibility
 - owner
 - owner_include
 - editor
 - editor_include
 - topics
 - host
 - lang
 - family_name
 - latest_instantiation
- List configuration / Sending and receiving setup
- List configuration / Privileges
- List configuration / Archives
 - Archive related
 - archive (OBSOLETE)
 - web_archive
 - archive_crypted_msg
- List configuration / Bounce management
 - Bounce related
 - bounce
 - bouncers_level1

- bouncers_level2
- welcome_return_path
- remind_return_path
- verp_rate
- List configuration / Data sources setup
 - Data source related
 - user_data_source
 - ttl
 - distribution_ttl
 - include_list
 - include_remote_sympa_list
 - include_sql_query
 - include_ldap_query
 - include_ldap_2level_query
 - include_file
 - include_remote_file
- List configuration / Others
 - Command related
 - remind_task
 - expire_task
 - review
 - List tuning
 - max_size
 - loop_prevention_regex
 - pictures_feature
 - cookie
 - custom_vars
 - custom_attribute
 - merge_feature
 - priority
 - Spam protection
 - spam_protection
 - web_archive_spam_protection
- Reception mode
 - Message topics
 - Message topic definition in a list
 - Subscribing to message topics for list subscribers
 - Message tagging
 - Multipart/alternative
- Shared documents
 - Shared documents
 - The three kinds of operations on a document
 - The description file
 - The predefined authorization scenarios
 - Access control
 - Shared document actions
 - Template files
 - d_upload.tt2
 - d_properties.tt2
- Sympa and its database
 - Sympa and its database
 - Prerequisites
 - Installing PERL modules
 - Creating a Sympa DataBase
 - Database structure
 - Database automatic creation and update
 - Database manual creation
 - Setting database privileges
 - Importing subscriber data
 - Importing data from a text file
 - Importing data from subscribers files
 - Extending database table format
 - Sympa logs in the database
 - Sympa configuration
- WWSympa, Sympa's web interface
 - WWSympa, Sympa's web interface
 - Organization
 - Web server setup
 - wwsympa.fcgi access permissions
 - Installing wwsympa.fcgi in your Apache server

- Installing wwsympa.fcgi in nginx
- Installing wwsympa.fcgi in lighttpd
- Using FastCGI
- wwsympa.conf parameters
 - arc_path
 - archive_default_index thrd – mail
 - archived_pidfile
 - bounce_path
 - bounced_pidfile
 - cookie_expire
 - cookie_domain
 - default_home
 - icons_url (obsolete since Sympa 5.4)
 - log_facility
 - mhonarc
 - htmlarea_url
 - password_case sensitive | insensitive
 - title
 - use_fast_cgi 0|1
- Database configuration
- Logging in as listmaster
 - The listmaster web interface
- Suspend my subscription
- Exclude users
- Web archives
 - Web archive
 - Features
 - MHonArc tool
 - Archives structure
 - Configuration parameters
 - Archived.pl daemon
 - Rebuilding web archive
 - Importing archives
- Sympa Internationalization
 - Sympa Internationalization
 - Catalogs and templates
 - Translating Sympa into your language
 - Defining language-specific templates
 - Translating topics titles
 - Handling of charsets
- Sympa RSS channel
 - Sympa RSS channel
 - latest_lists
 - active_lists
 - latest_arc
 - latest_d_read
- Sympa SOAP server
 - Sympa SOAP server
 - Introduction
 - Supported functions
 - Web server setup
 - Until version 5.3
 - Version 5.4 and higher
 - Sympa setup
 - Trust remote applications
 - The WSDL service description
 - Client-side programming
 - Writing a Java client with Axis
 - The test command line SOAP client
- Authentication
 - Authentication
 - S/MIME and HTTPS authentication
 - Authentication with email address, uid or alternate email address
 - Generic SSO authentication
 - CAS-based authentication
 - auth.conf
 - "regexp" and "negative_regexp" : the auth.conf switches
 - Login form
 - "auth.conf" structure
 - user_table paragraph

- Idap paragraph
- generic_sso paragraph
- cas paragraph
- Sharing WWSympa's authentication with other applications
 - Perl example
 - How to do it using PHP ?
 - What about using SOAP to access Sympa sessions
- Provide a Sympa login form in another application
- Setting up a Shibboleth-enabled Sympa server
 - Implementation in Sympa
 - Prerequisites
 - How it works
 - Configuring Sympa
 - Configuring Apache
 - Configuring Shibboleth
 - Declaring your Sympa service in your favourite federation
 - Coping with virtual hosts
 - What if you don't trust provided email addresses?
- Authorization scenarios
 - Authorization scenarios
 - Location of scenario file
 - Scenario structure
 - Scenario title
 - Rules overview
 - Rules definition
 - Named Filters
 - LDAP Named Filters Definition
 - SQL Named Filters Definition
 - Search condition
 - Scenario inclusion
 - Scenario implicit inclusion
 - Blacklist implicit rule
 - Custom Perl package conditions
 - Hiding scenario files
- Virtual hosting
 - Virtual host
 - How to create a virtual host
 - robot.conf
 - Virtual host customization
 - Managing multiple virtual hosts
- Interaction between Sympa and other applications
 - Interaction between Sympa and other applications
 - Soap
 - RSS channel
 - Sharing WWSympa's authentication with other applications
 - Sharing data with other applications
 - Subscriber count
- Message handling
 - Message workflow
 - Does Sympa alter messages?
 - Loop prevention
 - Customizing messages
- Managing list members and list owners
 - Managing list members and list owners
 - Standard definition of list members and owners
 - Dynamic mailing lists
- Customizing Sympa
 - Customizing Sympa/WWSympa
 - Template file format
 - Mail template files
 - helpfile.tt2
 - lists.tt2
 - global_remind.tt2
 - your_infected_msg.tt2
 - message_report.tt2
 - Web template files
 - Sympa colors customization guide
 - CSS files
 - css_path and css_url parameters

- What stylesheet will be used ?
 - Using wwsympa CSS generation process
 - Use custom stylesheets only
- Internationalization
 - Sympa internationalization
 - List internationalization
 - User internationalization
- Topics
- Authorization scenarios
- Custom parameters
- Custom user attributes
 - Custom attributes definition
 - How are the custom attributes values obtained from users?
 - How is it stored?
 - So, what can you do with that feature?
- Loop detection
- Tasks
- Bounce management
 - Bounce management
 - VERP
 - ARF
- Antivirus
 - Antivirus
- Antispam
 - How does Sympa deal with spams ?
 - A spam filter is needed
 - nrctpt_by_domain.conf
- DKIM
 - DKIM features for Sympa
 - Incoming messages
 - Outgoing messages
- Using Sympa with LDAP
 - Using Sympa with LDAP
- Sympa with S/MIME and HTTPS
 - Sympa with S/MIME and HTTPS
 - Signed message distribution
 - Use of S/MIME signatures by Sympa itself
 - Use of S/MIME encryption
 - S/Sympa configuration
 - Installation
 - Managing user certificates
 - Configuration in sympa.conf
 - Configuration to recognize S/MIME signatures
 - distributing encrypted messages
 - Managing certificates with tasks
 - chk_cert_expiration.daily.task model
 - crl_update.daily.task model
- Using Sympa commands
 - Using Sympa commands
 - User commands
 - Owner commands
 - Moderator commands
- About this document ...

[toc](#) · %2009/%08/%28 %11:%Aug

sympa.conf parameters

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

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

keyword value

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

Site customization

domain

This parameter used to be named `host`.

This keyword is **mandatory**. It is the domain name used in the `From:` header of mail sent by the Sympa engine. So the SMTP engine (gmail, sendmail, postfix or whatever) must recognize this domain as a local address. This parameter is also the default domain for the mailing lists and is used for mail routing internally in Sympa if you have defined virtual hosts. Note that a list domain can be changed on a per-list basis (see `host` parameter).

Example:

```
domain lists.my.tld
```

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
```

listmaster

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

Example:

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

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 sympas 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_email listmaster-test
```

you must modify the `sympa` aliases like this:

```
listmaster-test: | /home/sympa/bin/queue listmaster@my.domain.org
```

See [Robot aliases](#) for all aliases.

wwsympa_url

(Default value: `http://your.host/sympa`)

This is the root URL of the Sympa web interface. This parameter is used to construct URLs while sending notification emails to users.

Example:

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

soap_url

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

Example:

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

spam_protection javascript | at | none

(Default value: `javascript`)

There is a need to protect Sympa website against spambot which collect email addresses in public websites. Description of the supported values:

- `javascript`: the address is hidden using a javascript. Users who enable Javascript can see nice mailto addresses where others have nothing.
- `at`: the "@" char is replaced by the string "AT".

- none: no protection against spammers.

web_archive_spam_protection javascript | at | none | cookie

(Default value: cookie)

The same as `spam_protection`, but restricted to the web archive. An additional value is available: `cookie`, which means that users must submit a small form in order to receive a cookie before browsing the web archive. This block all robots, including search engine robots.

color_0, color_1, ..., color_15

They are the color definition parameters for the web interface. These parameters can be overwritten in each virtual host definition. Colors are used in the CSS files and unfortunately they are also in use in some web templates. The sympa admin interface shows all colors in use.

To know the exact role of each `color_x` parameter please consult [the color customization guide](#).

Obsolete color parameters

A few color parameters were used in the past for color definition of the web interface:

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

These parameters are not used in version 5.1 and higher anymore, but still available in `style.css`, `print.css`, `print-preview.css` and `fullPage.css`.

Note: `light_color` is still used for the header color of the New lists presentation array.

logo_html_definition

This parameter allows you to insert in the upper left corner of the page a piece of HTML code, usually to insert a logo in the page. This is a very basic but easy customization. Example:

```
logo_html_definition <a href='http://www.mycompany.com'><img style="float: left; margin-top: 7px; margin-left: 37px;" src='http://logos/mylogo.jpg' alt="my comp
```

main_menu_custom_button

You may modify the main menu content by editing the `menu.tt2` file but you can also edit the following robot parameters in order to add up to 3 button. each button is defined by a title (the text in the button), an URL and optionnaly a target.

- `main_menu_custom_button_1_title`
- `main_menu_custom_button_1_url`
- `main_menu_custom_button_1_target`

Replace digit 1 by 2 or 3 for the second and third custom button.

example :

```
main_menu_custom_button_1_title faq

main_menu_custom_button_1_url http://www.cru.fr/faq/universalistes/index

main_menu_custom_button_1_target help
```

css_path

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

The server admin module includes a CSS administration page. By pushing the "Install static css" button in this page, you create the CSS files in the folder whose path is contained by the `css_url` parameter.

After an upgrade, `sympa.p1` automatically updates the static CSS files with the newly installed `css.tt2`. Therefore, this is not a good place to store customized CSS files.

css_url

By default, CSS files `style.css`, `print.css`, `print-preview.css` and `fullPage.css` are delivered by the Sympa web interface itself using a Sympa action named `css`. URLs look like "http://foo.org/sympa/css/style.css". CSS files are built by parsing a template named `css.tt2`. This allows dynamic definition of colors, and in a near future 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 files somewhere in your website. This way, Sympa will deliver only one page instead of one page and four CSS files at each click. This can be done using the `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 sophisticated new skin by editing these files.

If you want to use the Sympa color parameters inside a static CSS file, you must do two things :

1. define an alias in your Apache configuration that associates the content of the folder whose path is stored in the `css_path` parameter to the URL specified in `css_url`;
1. use the Skins administration page after you changed colors in your configuration file to generate the static CSS files.

static_content_path

(Default value : `{Sympa install directory}/static-content`)

Some content may be delivered by the HTTP server (Apache) without any need to be controlled or parsed by Sympa. It is stored in the directory chosen through the `static_content_dir` parameter. The current Sympa version stores subscribers' pictures in this directory. Later updates will add stylesheets, icons, ... The directory is created by `sympa.pl` when started. This parameter can be defined also in `robot.conf`.

static_content_url

(Default value : `/static-sympa`)

Content stored in the directory specified by parameter `static_content_url` must be served by the 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`.

pictures_feature

(Default value: `off`)

Example:

```
-----
pictures_feature on
-----
```

Subscribers can upload their picture (from the 'Subscriber option' page) to use as an avatar so that reviewing subscribers shows a gallery. This parameter defines the default for corresponding list parameter but it does NOT allow to disable the feature overall. If you want to disable the feature for your entire site, you need to customize the `edit-list.conf` file to deny editing of the corresponding list parameter.

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

pictures_max_size

The maximum size of the uploaded avatar file (bytes).

cookie

This string is used to generate MD5 authentication keys. It allows generated authentication keys to differ from a 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 catastrophe, `sympa.pl` refuses to start if the `cookie` parameter was changed.

Example:

```
-----
cookie gh869jku5
-----
```

create_list

(Default value: `public_listmaster`)

The `create_list` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

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

Example:

```
-----
create_list intranet
-----
```

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 [Automatic list creation](#)).

automatic_list_creation

(Default value: none)

The `automatic_list_creation` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

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

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 creation, if they contain no list member (see [Automatic list creation](#)).

global_remind

(Default value: listmaster)

The `global_remind` parameter refers to an authorization scenario (see [Authorization scenarios](#)).

Defines who can run a `REMIND * command`.

allow_subscribe_if_pending

(Default value: on)

If set to "off", it is forbidden to add subscribers (through `wwsympa`) to a list whose status is different from "open".

sympa.conf parameters

Directories

home

(Default value: /home/sympa/expl)

The directory whose subdirectories correspond to the different lists.

Example: `home /home/sympa/expl`

etc

(Default value: /home/sympa/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;
- `list_task_models` for local list task models.

Example:

```
etc /home/sympa/etc
```

System related

syslog

(Default value: LOCAL1)

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

Example:

```
syslog LOCAL2
```

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
```

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.

pidfile

(Default value: /home/sympa/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 does not exist.

Example:

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

pidfile_creation

(Default value: /home/sympa/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 does not exist.

Example:

```
pidfile_creation /var/run/sympa-creation.pid
```

umask

(Default value: 027)

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

Example:

```
umask 007
```

Sending related

distribution_mode

(Default value: single)

Use this parameter to determine whether your installation runs only one `sympa.pl` daemon that processes both messages to distribute and commands (`single`), or if `sympa.pl` will fork to run two separate processes, one dedicated to message distribution and one dedicated to commands and message pre-processing (`fork`). The second choice makes a better priority processing for message distribution and faster command response, but it requires a bit more computer resources.

Example:

```
distribution_mode fork
```

maxsmtp

(Default value: 20)

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

Example:

```
maxsmtp 500
```

log_smtp

(Default value: `off`)

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

Example:

```
log_smtp on
```

use_blacklist

(Default value: `send,create_list`)

Sympa provides a blacklist feature available for list editors and owners. The `use_blacklist` parameter defines which operations use the blacklist. Search in blacklist is mainly useful for the `send` service (distribution of a message to the subscribers). You may use blacklist for other operations such as review, archive, etc., but be aware that those web services need fast response and blacklist may require some resources.

If you do not want blacklist at all, define `use_blacklist` to `none` so that the user interface to manage blacklist will disappear from the web interface.

max_size

(Default value: `5 Mb`)

Maximum size (in bytes) 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
```

misaddressed_commands

(Default value: `reject`)

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

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](#).

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 recipients for a particular domain. Check the `nrcpt_by_domain` configuration file (see [nrcpt_by_domain](#)).

avg

(Default value: `10`)

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

alias_manager

(Default value: `/home/sympa/bin/alias_manager.pl`)

The absolute path to the script that will add/remove mail aliases, see [Mail aliases](#).

Example:

```
alias_manager /home/sympa/bin/ldap_alias_manager.pl
```

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
```

sendmail_args

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

Arguments passed to the SMTP message transfer agent.

sendmail_aliases

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

Path of the alias file that contains all list related aliases. It is recommended to create a specific alias file so that Sympa never overwrites 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).

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.

remove_headers

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

This is the list of SMTP headers fields that Sympa should remove from incoming messages. Use it, for example, to ensure some privacy for your users by discarding anonymous options. An equivalent parameter can be set in list configuration files. The removal of these header fields is applied before Sympa adds his own header fields (`rfc2369_header_fields` and `custom_header`).

Example:

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

remove_outgoing_headers

(Default value: `none`)

You can define a comma-separated list of SMTP header fields that you wish Sympa to remove from outgoing headers. An equivalent parameter can be set in list configuration files. The removal happens after Sympa's own header fields are added ; therefore, it is a convenient way to remove Sympa's own header fields (like `X-Loop` or `X-No-Archive`) if you wish.

Example:

```
remove_outgoing_headers X-no-archive"
```

ignore_x_no_archive_header_feature

(Default value: `off`)

Sympa's default behavior is to skip archiving of incoming messages that have an `X-no-archive` SMTP header filed set. The `ignore_x_no_archive_header_feature` parameter allows to change this behavior.

Example:

```
ignore_x_no_archive_header_feature on
```

anonymous_headers_fields

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

This parameter defines the list of SMTP header fields that should be removed when a mailing list is setup in anonymous mode (see [anonymous_sender](#)).

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 are creating 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.

list_check_suffixes

(Default value: request, owner, unsubscribe)

This parameter is a comma-separated list of admin suffixes you are using for Sympa aliases, i.e. mylist-request, mylist-owner, etc. This parameter is used with the list_check_smtp parameter. It is also used to check list names at list creation time.

urlize_min_size

(Default value: 10240)

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

Bulk mailer

pidfile_bulk

Default: <default_pid_dir>/bulk.pid

The location in which the bulk.pl pidfile is created.

sympa_packet_priority

Default: 5

The default priority set to a packet to be sent by the bulk.

bulk_fork_threshold

Default: 1

The minimum number of packets in database before the bulk forks to increase sending rate.

bulk_max_count

Default: 3

The max number of bulks that will run on the same server.

bulk_lazytime

Default: 600

The number of seconds a slave bulk will remain running without processing a message before it spontaneously dies.

bulk_wait_to_fork

Default: 10

The number of seconds a master bulk waits between two packets number checks.

Quotas

default_shared_quota

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

default_archive_quota

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

Spool related

spool

(Default value: /home/sympa/spool)

The parent directory which contains all the other spools.

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:

```
-----  
/home/sympa/spool/msg  
-----
```

queuedistribute

(Default value: `/home/sympa/spool/distribute`)

This parameter is optional and retained solely for backward compatibility.

queuemod

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

This parameter is optional and retained solely for backward compatibility.

queuedigest

This parameter is optional and retained solely for backward compatibility.

queueauth

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

This parameter is optional and retained solely for backward compatibility.

queueoutgoing

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

This parameter is optional and retained solely for backward compatibility.

queuetopic

(Default value: `/home/sympa/spool/topic`)

This parameter is optional and retained solely for backward compatibility.

queuebounce

(Default value: `/home/sympa/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.

queuetask

(Default value: `/home/sympa/spool/task`)

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

queueautomatic

(Default value: `none`)

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

Example:

```
-----  
/home/sympa/spool/msg  
-----
```

tmpdir

(Default value: `/home/sympa/spool/tmp`)

Temporary directory used by OpenSSL and antiviruses.

sleep

(Default value: 5)

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

clean_delay_queue

(Default value: 7)

Retention period (in days) for "bad" messages in the messages spool (as specified by [queue](#)). Sympa keeps messages rejected for various reasons (badly formatted, looping, etc.) in this directory. This configuration variable controls the number of days these messages are kept.

Example:

```
clean_delay_queue 3
```

clean_delay_queueoutgoing

(Default value: 7)

Retention period (in days) for "bad" messages in the outgoing spool (as specified by [queueoutgoing](#)). Sympa keeps messages rejected for various reasons (unable to create archive directory, to copy file, etc.) in this directory. This configuration variable controls the number of days these messages are kept.

Example:

```
clean_delay_queueoutgoing 3
```

clean_delay_queuebounce

(Default value: 7)

Retention period (in days) for "bad" messages in the bounce spool (as specified by [queuebounce](#)). Sympa keeps messages rejected for various reasons (unknown original sender, unknown feedback type) in this directory. This configuration variable controls the number of days these messages are kept.

Example:

```
clean_delay_queuebounce 3
```

clean_delay_queueother

(Default value: 30)

Retention period (in days) for messages in the bounce/OTHER spool (as specified by [queuebounce](#)). Sympa keeps messages rejected for various reasons in this directory. This configuration variable controls the number of days these messages are kept.

Example:

```
clean_delay_queueother 7
```

clean_delay_queuemod

(Default value: 30)

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.

clean_delay_queueauth

(Default value: 30)

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

clean_delay_queuesubscribe

(Default value: 30)

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

clean_delay_queuetopic

(Default value: 30)

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

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 with `BAD`. This configuration variable controls the number of days these messages are kept.

clean_delay_tmpdir

(Default value: 7)

Retention period (in days) for files put in the tmp dir (as specified by `tmpdir`). This configuration variable controls the number of days these files are kept.

sympa.conf parameters

Internationalization related

localedir

(Default value: `/home/sympa/locale`)

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

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 catalogs but it can be narrowed by the listmaster.

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.

web_recode_to

(OBSOLETE)

All web pages now use UTF-8 charset

filesystem_encoding

OBSOLETE

Now all files (including configuration files, templates, authorization scenarios,...) must use UTF-8 charset.

Bounce related

verp_rate

(Default value: 0%)

See `VERP` for more information on VERP in Sympa.

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

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

welcome_return_path

(Default value: `owner`)

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

remind_return_path

(Default value: owner)

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

`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).

`expire_bounce_task`

(Default value: daily)

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

`purge_orphan_bounces_task`

(Default value: Monthly)

This parameter tells what task will be used by `task_manager.pl` to perform bounce cleaning. This task deletes bounce archive for unsubscribed users.

`eval_bouncers_task`

(Default value: daily)

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

`process_bouncers_task`

(Default value: monthly)

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

`minimum_bouncing_count`

(Default value: 10)

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

`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.

`bounce_delay`

(Default value: 0)

Another parameter for the bounce-score evaluation: this one represents the average time (in days) for a bounce to come back to the Sympa server after a post was send to a list. Usually bounces are delivered on the same day as the original message.

`default_bounce_level1_rate`

(Default value: 45)

This is the default value for `bouncerslevel1_rate` entry (see [bouncers_level1](#)).

`default_bounce_level2_rate`

(Default value: 75)

This is the default value for `bouncerslevel2_rate` entry (see [bouncers_level2](#)).

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 starts 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 encodes the subscriber address. This parameter is useful if you want to run more than one Sympa on the same host (a test Sympa 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 [Robot aliases](#) for all aliases.

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.

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.

default_remind_task

(Default value: 2month)

This parameter defines the default `remind_task` list parameter.

Tuning

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 of parsing all config files. The drawback of this method is that the list config cache can live for a long time (not recreated when the Sympa processes restart); the Sympa processes could still use authorization scenario rules or default for list parameters (set in `sympa.conf`) that have changed on disk in the meantime. You can work this problem out by frequently running a `sympa.pl --reload_list_config` using the crontab. In the long term, Sympa should update `config.bin` files via the `task_manager`.

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

lock_method

Format: flock | nfs (Default value: flock)

This parameter will tell Sympa how it should perform locks when required (updating DB, updating config file,...). The default method uses the standard `flock` function. Another option is to use NFS locking ; it requires that you install `File::NFSLock` perl module first.

sympa_priority

(Default value: 1)

Priority applied to Sympa commands while running the spool.

Available since release 2.3.1.

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.

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.

default_list_priority

(Default value: 5)

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

Available since release 2.3.1.

Database related

The following parameters are needed when using a RDBMS, but are otherwise not required.

update_db_field_types

Format:

```
update_db_field_types auto | disabled
```

(Default value: `auto`)

This parameter defines whether Sympa automatically updates database structure to match the expected datafield types. This feature is only available with MySQL. Note however that since version 5.3b.5, Sympa will not alter DB fields that have a bigger size (if `update_db_fields_types` is set to `auto`).

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.

db_name

(Default value: `sympa`)

Name of the database containing user information. If you are using SQLite, then this parameter is the DB file name.

db_host

Database host name.

db_port

Database port.

db_user

User with read access to the database.

db_passwd

Password for `db_user`.

db_timeout

This parameter is used for SQLite only.

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.

db_env

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

Example for Oracle:

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

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
```

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
```

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.

purge_tables_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 `bulkspool_table` table that don't have any relationships to a packet in the `bulkmailer_table`.


purge_logs_table_task

(Default value: `daily`)

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 `logs_table` table.

logs_expiration_period

(Default value: `3`)

Datas in `logs_table` table are removed when they are older than `logs_expiration_period`. The parameter value is interpreted as a number of month. ( it should be replaced by stand duration expression).

purge_session_table_task

(Default value: `daily`)

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 `session_table` table.

session_table_ttl

(Default value: `3d`)

Session duration is controled by `sympa_session` cookie validity attribute, but it is needed for security reason to control this delay on the server side. The default time to leave for sessions. Session with unactivity period longer than this parameters are removed. If this parameter is very long then the `session_table` may become very large, mainly because most crawler robots do not manage cookies.

Duration values format are easy to read, the format is a string without spaces including y for years, m for months d for days, h for hours min for minutes and sec for secondes.

purge_challenge_table_task

(Default value: `daily`)

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 `challenge_table` table. (This table is used to store information about email authentication challenges).

challenge_table_ttl

(Default value: `5d`)

Challenge sent by email are store until they are used, but Sympa may remove challenges that are *too old* both for security reasons and to keep table in a reasonable size. This parameter is used to specify what does mean a *too old challenge*.

Loop prevention

The following define your loop prevention policy for commands (see [Loop detection](#)).

loop_command_max

(Default value: `200`)

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

loop_command_sampling_delay

(Default value: `3600`)

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

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.

loop_prevention_regex

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

This regular expression is applied to message sender addresses. 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.

S/MIME configuration

Sympa can optionally check and use S/MIME signatures for security purposes. In this case, the three first following parameters must be set by the listmaster (see [Configuration in sympa.conf](#)). The two others are optional.

openssl

The path for the OpenSSL binary file.

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 of this form to them (`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 the `mod_ssl` module for Apache.

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`.

key_passwd

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

Antivirus plug-in

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

antivirus_path

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

Example:

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

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
```

Example with AVP:

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

Example with Sophos:

```
antivirus_path /usr/local/bin/sweep
```

```
antivirus_args -nc -nb -ss -archive
```

Example with Clamav:

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

antivirus_notify

sender | nobody

(Default value: sender)

This parameter defines whether Sympa should notify the email sender when a virus has been detected.

Mailing list definition

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

Mail aliases

See list aliases section, [Mail aliases](#).

List configuration file

The configuration file for the `mylist` list is named `/home/sympa/expl/my.domain.org/mylist/config` (or `/home/sympa/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 [cache_list_config](#)), a binary version of the config (`/home/sympa/expl/my.domain.org/mylist/config.bin`) is maintained to allow a faster restart of daemons (this is especially useful 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, [List configuration parameters](#).

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
```

Subscribers file

Be careful: since version 3.3.6 of Sympa, a RDBMS is required for internal data storage. Flat files should not be use anymore except for testing purpose. Sympa will not use these files if the list is configured with `include_database` or `user_data_source`.

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

- `email_address`
Email address of the subscriber.
- `gecos_data`
Information about the subscriber (last name, first name, etc.) This parameter is optional at subscription time.
- `reception | nomail | digest | summary | notice | txt | html | urlize | not_me`
Special delivery modes which the subscriber may select. Special modes can be either `nomail`, `digest`, `summary`, `notice`, `txt`, `html`, `urlize` and `not_me`. In normal delivery mode, the delivery attribute for a subscriber is not displayed. In this mode, subscription to message topics is available. See the [SET LISTNAME SUMMARY](#) command, the [SET LISTNAME NOMAIL](#) command and the [digest](#) parameter.
- `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 CONCEAL](#) command for details.

Info file

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

Homepage file

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

Data inclusion file

Every file has the `.incl` extension. Moreover, these files must be declared in paragraphs `owner_include` or `editor_include` in the list configuration file (without the `.incl` extension) (see [List configuration parameters](#)). These files can be template files.

Sympa looks for them in the following order:

1. `/home/sympa/expl/mylist/data_sources/<file>.incl;`
1. `/home/sympa/etc/data_sources/<file>.incl;`
1. `/home/sympa/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 email addresses using an SQL or LDAP query, or by including other mailing lists.

A data inclusion file is made of paragraphs separated by blank lines and introduced by a keyword. Valid paragraphs are `include_file`, `include_remote_file`, `include_list`, `include_remote_sympa_list`, `include_sql_query`, `include_ldap_2level_query` and `include_ldap_query`. They are described in the [List configuration parameters](#) chapter.

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

Example:

- in the list configuration file `/home/sympa/expl/mylist/config` :

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

- in `/home/sympa/etc/data_sources/myfile.incl`:

```
include_sql_query
db_type [1949eb08aram.0 %]
host sqlserv.admin.univ-[1949eb08aram.1 %].fr
user [1949eb08aram.2 %]
passwd [1949eb08aram.3 %]
db_name [1949eb08aram.4 %]
```

```
sql_query SELECT DISTINCT email FROM [1949eb08aram.5 %]
```

- once it has been parsed with provided parameters, the inclusion directives would look like this:

```
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
```

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 [Template file format](#).

Sympa looks for these files in the following order:

1. /home/sympa/expl/mylist/mail_tt2/<file>.tt2;
1. /home/sympa/etc/mail_tt2/<file>.tt2;
1. /home/sympa/bin/etc/mail_tt2/<file>.tt2.

If the file starts with a From: line, it is considered 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:

- [`onf.email %`]: Sympa email address local part;
- [`onf.domain %`]: Sympa's robot domain name;
- [`onf.sympa %`]: Sympa's complete email address;
- [`onf.wwsympa_url %`]: *WWSympa*'s root URL;
- [`onf.listmaster %`]: listmasters' email addresses;
- [`Ost.name %`]: list name;
- [`Ost.host %`]: list hostname (default is Sympa robot domain name);
- [`Ost.lang %`]: list language;
- [`Ost.subject %`]: list subject;
- [`Ost.owner %`]: list owners table hash;
- [`Oser.email %`]: user email address;
- [`Oser.gecos %`]: user geccos field (usually his/her name);
- [`Oser.password %`]: user password;
- [`Oser.lang %`]: user language;
- [`0.000000e+00xecution_date %`]: the date when the scenario is executed.

You may also dynamically include a file from a template using the [`% INSERT %`] directive.

Example:

```
Dear [Oser.email %],

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

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

The owners of [ Ost.name %] are:
[ 0.0000000E+00EACH ow = list.owner %]
  [Ow.value.gecos %] <[Ow.value.email %]>
[ 0.000000E+00ND %]
```

welcome.tt2

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

bye.tt2

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

removed.tt2

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

reject.tt2

Sympa will send a reject message to the senders of messages rejected by the list editors. If they prefix their REJECT with the keyword QUIET, the reject message will not be sent.

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 %]: email of the person who sent the INVITE command;
- [0r1 %]: the mailto: URL to subscribe to the list.

remind.tt2

This file contains a message sent to each subscriber when one of the list owners sends the REMIND command.

summary.tt2

Template for summaries (reception mode close to digest), see the SET LISTNAME SUMMARY command.

list_aliases.tt2

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

Note that this template is not a service messages, therefore it is not located in a mail_tt2/ subdirectory.

Stats file

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

- number of messages sent, used to generate X-sequence headers;
- number of messages X number of recipients;
- number of bytes X number of messages;
- number of bytes X number of messages X number of recipients;
- number of subscribers;
- last update date (epoch format) of the subscribers cache in DB, used by lists in include2 mode only.

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 [Tasks](#).

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.

expire.annual.task

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

Message header and footer

You may create the /home/sympa/expl/mylist/message.header and /home/sympa/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 is 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 algorithm:

```

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

```

Archive directory

The `/home/sympa/expl/mylist/archives/` directory contains the messages archived for lists which are archived; see [archive](#). The files are named in accordance with the archiving frequency defined by the `archive` parameter.

List creation, editing and removal

The list creation can be done in two ways, according to listmaster needs:

- family instanciation, to create and manage a 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 [Automatic list creation](#)).
- command line creation of individual list with `sympa.pl` or on the web interface according to privileges defined by listmasters. In this case, lists are free from their creation model.

Management of mailing lists by list owners is usually done through the web interface: when a list is created, whatever its status (`pending` or `open`), the owners can use *WWSympa* administration features to modify list parameters, 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.

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 email) to be replaced by Sympa at list creation time. It is easy to create new list templates by modifying existing ones.

Please note that contributions to the distribution are welcome to complete the set of existing templates...



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.

in addition to 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 through an XML file.

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 information;

- 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>
  <moderator>
    <email>user@domain.org</email>
  </moderator>
  <topic>Computing</topic>
  <sql>
    <type>Oracle</type>
    <host>sqlserv.admin.univ-x.fr</host>
    <port>1521</port>
    <user>stdutilisateur</user>
    <pwd>monsecret</pwd>
    <name>les_etudiants</name>
    <env>ORACLE_HOME=[oracle_path]</env>
    <query>SELECT DISTINCT email FROM etudiant</query>
  </sql>
</list>
```

Then edit List Creation Template – example:
`/[sympahome]/bin/etc/create_list_templates/discussion_list/config.tt2`

```
subject [ubject %]

status [tatus %]

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

[ 0.000000E+00ND %]
visibility noconceal

send privateoreditorkey

Web_archive
access public
```



```

subscribe open_notify

shared_doc
  d_edit [hared_edit %]
  d_read [hared_read %]

lang [% language %]

[ 0.000000EACH o = owner %]
owner
  email [0.email %]
  profile privileged
  [% IF o.gecos %]
  geccos [0.gecos %]
  [ 0.000000E+00ND %]

[ 0.000000E+00ND %]
[% IF moderator %]
  [ 0.000000EACH m = moderator %]
editor
  email [% m.email %]

  [ 0.000000E+00ND %]
[ 0.000000E+00ND %]

[% IF sql %]
include_sql_query
  db_type [ql.type %]
  db_port [ql.port %]
  host [ql.host %]
  user [ql.user %]
  passwd [ql.pwd %]
  db_name [ql.name %]
  db_env [ql.env %]
  sql_query [ql.query %]

[ 0.000000E+00ND %]

default_user_options
  reception urlize|mail|digest

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 does not exclude 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, the 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 the `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>`

List families

See chapter [Lists families](#).

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 [XML file format](#).

By default, the status of the list created 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 `/home/sympa/etc/create_list_templates` or in `/home/sympa/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 `/home/sympa/etc/create_list_templates` and in the `/home/sympa/bin/etc/create_list_templates` directory, then the local profile will be used by *WWSympa*.

Creating and editing mailing lists 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.

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 lists are then validated by the listmaster.

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

Who can create lists on the web interface

This is defined by the `create_list sympa.conf` parameter. This parameter refers to a `create_list` authorization scenario. It will determine whether the `create list` button is displayed and whether list creation requires a listmaster confirmation.

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

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

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 email and the list subject together with a short description. But in this case, you do not need any XML file. Concerning these typical list profiles, they are described before, see [Typical list profile \(list template creation\)](#). You can check available profiles. 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 `/home/sympa/bin/etc` directory, and you may create your own customized version in `/home/sympa/etc`). This file controls which of the available list templates are to be displayed. Example:

```
-----
## This sample hides the public_anonymous create_list template
public_anonymous hidden
defaults read
-----
```

customize create_list_request.tt2

The list creation form is in a template named `create_list_request.tt2`. You may modify this template in order to some other input that will be used to modify the created list. Any new input variable will be caught by `wwwsympa.fcgi` and available in the `tt2` hash [`custom_input %`] when using the list template to create the list.

exemple :

```
## This is an html part added in create_list-request.tt2
<input type="text" name="custom_input.ldap_group" />
```

In `config.tt2` you may use

```
...
include_ldap_query
host ldap.foo.edu
suffix ou=accounts,dc... filter (&(isMemberOf=[%
custom_input.ldap_group %]))
...
```

For more details on `tt2` customization, templates path etc please go to the [web template files section](#)

List editing

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

Each line is a set of 3 field.

```
<Parameter> <Population> <Privilege>
<Population>: <listmaster|privileged_owner|owner>
<Privilege>: <write|read|hidden>
```

Parameter can be any list config parameter or the name of a template (thus controlling the edition of the template through the *customize* web admin feature. You can refer to a subentry of a structured list parameter using the `'.'` as a separator (examples: **owner.email** or **web_archive.quota**). **default** is a reserved parameter name that means *any other parameter*.

There is no hierarchical relationship between populations in this configuration file. You need to explicitly list populations.

For example, `listmaster` will not match rules referring 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 the listmaster can edit the `profile` `privileged owner` attribute.
- Owners can edit their own attributes (except `profile` and `email`).
- The person creating a new list becomes its privileged owner.
- Privileged owners can edit any `gecos/reception/info` attribute of any owner.
- Privileged owners can edit owners' email addresses (but not privileged owners' email addresses).

Sympa aims at defining 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' emails, but they cannot grant the responsibility of list management to others without referring to the listmaster.

Concerning list editing in a family context, see [editing list parameters in a family context](#).

Removing a list

You can remove (close) a list either from the command line or by using the web interface.

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

```
sympa.pl --close_list=mylist@mydomain
```

Privileged owners can remove a mailing list through the list administration 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.

List families

A list can have from three up to dozens of parameters. 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 groups of lists. Moreover, families allow listmasters to delegate a part of configuration list to owners, in a controlled way according to family properties. Distribution will provide defaults families.

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;
- a degree of independence between the lists and the family: list parameters editing rights and constraints on these parameters can be *free* (no constraint), *controlled* (a set of available values 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 operations performed on a family:

- definition: definition of the list creation template, the degree of independence and family customizations;
- instantiation: list 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 and has consequences on every list;
- closure: closure of each list;
- adding a list to a family;
- closing a family list;
- modifying a family list.

Using family

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 the following 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 email) 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 [List template files](#) and [Template file format](#).

Example:

```

subject [ubject %]

status [tatus %]

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

[ 0.000000E+00ND %]
visibility noconceal

send privateoreditorkey

web_archive
  access public

subscribe open_notify

shared_doc
  d_edit [hared_edit %]
  d_read [hared_read %]

lang [% language %]

[ 0.000000E+00REACH o = owner %]
owner
  email [0.email %]
  profile privileged
  [% IF o.gecos -%]
  geccos [0.gecos %]
  [ 0.000000E+00ND %]

[ 0.000000E+00ND %]
[% IF moderator %]
  [ 0.000000E+00REACH m = moderator %]
editor
  email [% m.email %]

  [ 0.000000E+00ND %]
[ 0.000000E+00ND %]

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

```

```
[ 0.000000E+00ND %]
ttl 360
```

param_constraint.conf

This file is mandatory. It defines constraints on parameters. There are three kinds 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 [msg-topic](#)), `owner_include.source_parameter` (see [owner include](#)) and `editor_include.source_parameter` (see [editor include](#)). About `digest` parameter (see [digest](#)), only 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 [List editing](#). 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).

Note that by default, the `family_name` parameter is not writable, you should not change this editing right.

customizable files

Families provide a new level of customization for scenarios (see [Authorization scenarios](#)), templates for service messages (see [Site template files](#)) and templates for web pages (see [Web template files](#)). Sympa looks for these files in the following level order: list, family, robot, server site or distribution.

Example of custom hierarchy:

```
/home/sympa/etc/families/myfamily/mail_tt2/
/home/sympa/etc/families/myfamily/mail_tt2/bye.tt2
/home/sympa/etc/families/myfamily/mail_tt2/welcome.tt2
```

Instantiation

Instantiation allows to generate lists. You must provide an XML file made of list descriptions, the root element being `family` and which is only composed of `list` elements. List elements are described in section [XML file format](#). Each list is described by the set of values for affectation list parameters.

Here is a sample command to instantiate a family:

```
sympa.pl --instantiate_family my_family --robot samplerobot --input_file /path/to/my_file.xml
```

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.

`-close_unknown` option can be added to automatically close undefined lists during a new instantiation

`-quiet` option can be added to skip the report printed to STDOUT

Example:

```
<?xml version="1.0" ?>
<family>
  <list>
    <listname>listel</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>foo@cru.fr</email>
  <gecos>C.R.U.</gecos>
</owner>
<owner multiple="1">
  <email>foo@emns.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>foo@cru.fr</email>
    <gecos>C.R.U.</gecos>
  </owner>
  <owner multiple="1">
    <email>foo@emns.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 with the previous instantiation, there are three cases:

- list creation: new lists described by the new instantiation;
- list modification: lists already existing but possibly changed because of changed parameters values in the XML file or because of changed family properties;
- list removal: lists no more described by the new instantiation. In this case, the listmaster must validate 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 all parameters modified (through the web interface), indicated 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 instantiation, etc.), the list is set in status `error_config` and listmasters are notified. Then they will have to perform any necessary action in order to put the list in use.
- For each list closure in family context, the list is set in status `family_closed` and owners are notified.
- For each overwritten list customization, owners are notified.

Modification

To modify a family, you have to edit family files manually. The modification will be effective while the next instantiation.

WARNING: The family modification must be done just before an instantiation. Otherwise, alive lists would not respect new family properties and they would be set in status `error_config` immediately.

Closure

Closes every list (installed under the indicated robot) of this family: 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 restoration of the list).

Here is a sample command to close a family:

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

Adding a list to a list family

Adds a list to the family without instantiating the whole 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 [List creation on command line with sympa.pl](#).

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
```

Removing a list from a list family

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
```

Modifying a family 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 [List creation on command line with sympa.pl](#).

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

```
sympa.pl --modify_list my_family --robot samplerobot --input_file /path/to/my_file.xml
```

Editing list parameters in a family context

According to file `edit_list.conf`, editing rights are controlled. See [List editing](#). But in a family context, constraints parameters are added to editing right as it is summarized in this array:

array

Note: in order to preserve list customization for instantiation, every parameter modified (through the web interface) is indicated in the `config_changes` file.

Automatic list creation

You can benefit from the family concept to let Sympa automatically create lists for you. Let us assume that you want to open a list according to specified criteria (age, geographical location, ...) 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 will not be created yet. The mailing list creation is triggered when Sympa receives a message addressed to this list.

To enable automatic list creation, you will 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.

Configuring your MTA

The familyqueue solution (with postfix)

To do so, you have to configure your MTA for it to add a custom header field to messages. The easiest way is to customize your aliases manager, so that mails for automatic lists are not 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 family name (whereas the `queue` program only expects the list address).

Now 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 we need to write to all CTOs who are fifty years old, we 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. We'll also need to tell the MTA to accept mail for addresses that do not yet exist since by default postfix will reject mail for unknown local users.

```

/etc/postfix/main.cf
...
transport_maps = regexp:/etc/postfix/transport_regexp
local_recipient_maps = pcre:/etc/postfix/local_recipient_regexp unix:passwd.byname $alias_maps

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

/etc/postfix/local_recipient_regexp
/^.*-owner@lists\.domain\.com$/      1
/^auto-.*@lists\.domain\.com$/      1

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

```

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

The sympa-milter solution (with sendmail)

If you don't use postfix or don't want to dig in postfix alias management, you have an alternative solution for automatic listes management: sympa-milter.

This program is a contribution by Jose-Marcio Martins da Cruz [mailto:Jose-Marcio.Martins@ensmp.fr].

What it does is checking all incoming mails and, if it recognizes a message to an automatic list, adds the relevant headers in it and places it in Sympa's automatic spool. It replaces familyqueue.

For all the doc, we assume you're using sendmail.

This is the procedure to make it work:

Install sympa-milter

You can download the latest version at the following address: <http://j-chkmail.ensmp.fr/sympa-milter/> [<http://j-chkmail.ensmp.fr/sympa-milter/>].

Once you have the archive, decompress it: `tar xzvf sympa-milter-0.6.tgz`.

Then install the program:

```
# cd sympa-milter-0.6/
# ./configure
# make
# make install
```

The default install directory is `/usr/local/sympa-milter/` (you can change this value with the `—prefix` configure option).

The install process also adds a launcher into `/etc/init.d/`, named `sympa-milter`. You'll need to setup links to it under `/etc/rc3.d`. If you're using Fedora like Linux distributions, you can use `/sbin/chkconfig` to setup these links.

```
/sbin/chkconfig sympa-milter on
```

You must then set up the configuration file, `sympa-milter.conf`. You will find a sample configuration file inside `/usr/local/sympa-milter/etc` directory. This file contains two sections whose border are XML-like tags. Inside a section, a parameter is defined on a single line by the sequence:

```
parameters_name parameter_value
```

- the general section, between the `<general>` and `</general>` tags is used to define, well general parameters, related to the program execution. It contains the following items:
 - `log_level` (positive integer value): the amount of logs generated by sympa-milter;
 - `log_facility` (string): the syslog facility in which the program will log;
 - `log_severity` (string: yes/no): If you enable this, `syslog` will include a string like `[ID 000000 local6.info]` in each log line, allowing you to identify the log level and facility.
 - `socket` (string): the socket used by the application; must be the same as the one defined in your MTA;
 - `spool_dir` (string): the absolute path to the `automatic` [<http://www.sympa.org/wiki/manual/organization#spools>] spool in which messages should be placed;
 - `pid_file` (string): the absolute path to the pid file (default = `/usr/local/sympa-milter/var/sympa-milter.pid`);
 - `run_as_user` (string) the user the uid under which to execute sympa-milter (default = `sympa`, but changeable by a `configure` script option); this must be the same as the one running sympa;
 - `run_as_group` the group the gid under which to execute sympa-milter (default = `sympa`, but changeable by a `configure` script option); this must be the same as the one running sympa;
- the family definition section, between the `<families>` and `</families>` tags is used to define the regular expressions which will allow sympa-milter to catch list creation messages. This section can contain an unlimited number of identically built lines, following this syntax:

```
family      recipient_regular_expression
```

You should use "plussed aliases" (at least with sendmail) to identify user existence more easily.

Here is an example of `sympa-milter.conf`, filled-up with default values :

```
#
# Section general
#
<general>
log_level      10
log_facility   local6
```

```

log_severity      yes

socket            inet:2030@localhost

spool_dir         /usr/local/sympa-milter/var

pid_file          /usr/local/sympa-milter/var/sympa-milter.pid

run_as_user       sympa
run_as_group      sympa
</general>
#
# Section families
#
<families>
# Syntax :
#   family      recipient regular expression
#
joe               ^joe+.*@one.domain.com
toto              ^bob+toto@other.domain.com
best              ^best.*@another.domain.com
</families>

```

Note: It is probably better to make all your regular expression start with "^". This way, bouncing messages won't be caught by sympa-milter and normally processed.

You can use any regular expression to define the addresses used by your family.

Set up your MTA

What you must do to make all the thingy to work is:

- setting up your MTA to use sympa-milter:

```

O InputMailFilters=sympa-milter
Xsympa-milter, S=inet:2030@localhost, T=C:2m;S:20s;R:20s;E:5m

```

- defining aliases to prevent sendmail from howling that a user (corresponding to your automatic list) doesn't exist. If all your automatic lists start with "auto", for example you can write:

```

auto : /dev/null

```

or

```

auto : "some_file"

```

Reload your MTA config. All set!

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 (nemes.size != 2 || nemes.0 != 'auto');
tokens = nemes.1.split('\.');
```

```

THROW autofamily "SYNTAX ERROR: wrong listname syntax" IF (tokens.size != 2 || ! occupations.${tokens.0} || tokens.1 < 20 || tokens.1 > 99 );
age = tokens.1 div 10;
%]

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

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

include_ldap_query
attrs mail
filter (&(objectClass=inetOrgPerson)(employeeType=[0occupations.${tokens.0}.abbr %])(personAge=[% age %]*))
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.

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 `/home/sympa/spool/msg` spool).

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

```

/home/sympa/etc/sympa.conf
...
automatic_list_feature on
automatic_list_creation public
queueautomatic /home/sympa/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 not use the list-related variables;
- you can only use the `smtp` and `smime` authentication methods in scenario rules (you cannot request the `md5` challenge). Moreover, only the `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.

List configuration parameters

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

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

Configuration parameters must be separated by blank lines and **BLANK LINES ONLY!**

Using the web interface the following categories are used to organize the large number of parameters :

- List definition;
- Sending/receiving setup;
- Privileges;

- [Archives](#);
- [Bounce management](#);
- [Data sources setup](#);
- [Others](#).

List parameters: definition

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.

visibility

(Default value: `conceal`)

The `visibility` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

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

- `visibility conceal` (view [\[http://www.sympa.org/distribution/current/src/etc/scenari/visibility.conceal\]](http://www.sympa.org/distribution/current/src/etc/scenari/visibility.conceal))
- `visibility intranet` (view [\[http://www.sympa.org/distribution/current/src/etc/scenari/visibility.intranet\]](http://www.sympa.org/distribution/current/src/etc/scenari/visibility.intranet))
- `visibility noconceal` (view [\[http://www.sympa.org/distribution/current/src/etc/scenari/visibility.noconceal\]](http://www.sympa.org/distribution/current/src/etc/scenari/visibility.noconceal))
- `visibility secret` (view [\[http://www.sympa.org/distribution/current/src/etc/scenari/visibility.secret\]](http://www.sympa.org/distribution/current/src/etc/scenari/visibility.secret))

owner

The `config` file contains one `owner` paragraph per owner. It concerns static owner definition. For dynamic definition, see [owner include](#).

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` and `DELETE` commands (see [Owner commands](#)) on behalf of other users.

When the [subscribe parameter](#) 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 emails. Useful to define an owner with multiple email addresses: they are all recognized when Sympa receives mail, but thanks to `reception nomail`, not all of these addresses need to receive administrative email from Sympa;
- `visibility conceal / noconceal` \\Define if the list owner should be listed on the list web page.
- `gecos data`
Public information about the owner;
- `info data`
Available since release 2.3. Private information about 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.

owner_include

The `config` file contains one `owner_include` paragraph per data inclusion file (see [Data inclusion](#))

file. It concerns dynamic owner definition: inclusion of external data. For static owner definition and more information about owners see [par-owner](#).

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 an enumeration of the values that will be affected to the `param` array used in the template file (see [Data inclusion file](#)). This parameter is not mandatory.
- `reception nomail`
Optional attribute for owner(s) who does not wish to receive emails.
- `visibility conceal / noconceal` \\Define if the included owners should be listed on the list web page.
- `profile privileged | normal`
Profile of the owner(s).

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 [editor include](#).

Example:

```
editor
email Pierre.Paul@myuniversity.edu
gecos Pierre paul (Computer center director)
```

Only the editor of a list is authorized to send messages to the list when the `send` 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](#), even when several moderators are defined.

editor_include

The `config` file contains one `editor_include` paragraph per data inclusion file (see [Data inclusion file](#)). It concerns dynamic editor definition: inclusion of external data. For static editor definition and more information about moderation see [editor](#).

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](#), even when several moderators are defined.

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 the `msg_topic` parameter used to tag emails.

host

(Default value: domain robot parameter)

`host fully-qualified-domain-name`

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

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 [Internationalization](#) for available languages.

family_name

This parameter indicates the name of the family that the list belongs to.

Example:

```
family_name my_family
```

latest_instantiation

This parameter indicates the date of the latest instantiation.

Example:

```
latest_instantiation
email joe.bar@cru.fr
date 27 jui 2004 at 09:04:38
date_epoch 1090911878
```

send

(Default value: `private`)

The `send` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

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

- `send closed` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.closed>])
- `send editorkey` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.editorkey>])
- `send editorkeyonly` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.editorkeyonly>])
- `send editorkeyonlyauth` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.editorkeyonlyauth>])
- `send intranet` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.intranet>])
- `send intranetorprivate` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.intranetorprivate>])
- `send newsletter` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.newsletter>])
- `send newsletterkeyonly` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.newsletterkeyonly>])
- `send private` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.private>])
- `send private_smime` (view [http://www.sympa.org/distribution/current/src/etc/scenari/send.private_smime])
- `send privateandeditorkey` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.privateandeditorkey>])
- `send privateandnomultipartoreditorkey` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.privateandnomultipartoreditorkey>])
- `send privatekey` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.privatekey>])
- `send privatekeyandeditorkeyonly` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.privatekeyandeditorkeyonly>])
- `send privateoreditorkey` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.privateoreditorkey>])
- `send privateorpublickey` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.privateorpublickey>])
- `send public` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/send.public>])

- `send_public_nobcc` (view
[http://www.sympa.org/distribution/current/src/etc/scenari/send_public_nobcc])
- `send_publickey` (view [http://www.sympa.org/distribution/current/src/etc/scenari/send_publickey])
- `send_publicnoattachment` (view
[http://www.sympa.org/distribution/current/src/etc/scenari/send_publicnoattachment])
- `send_publicnomultipart` (view
[http://www.sympa.org/distribution/current/src/etc/scenari/send_publicnomultipart])

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 numeric 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 (i.e. around midnight), Sympa may not be able to process it in time. Therefore do not set a digest time later than 23:00.

N.B.: In family context, `digest` can be constrained only on days.

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.

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,txt,html,urlize,not_me`

modelist is a list of modes (`mail, notice, digest, summary, nomail, txt,html, urlize, not_me, topics`), separated by commas. Only these modes will be allowed for the subscribers of the list. If a subscriber has a delivery mode other than those specified in that 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
-----
```

default_user_options

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

This profile only applies for newly subscribed/included list members. It means that if you change the value of `default_user_options` in a list, it will not apply to the existing list members.

- `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
-----
```

msg_topic

The `msg_topic` parameter starts a paragraph to define a message topic used to tag a message. For each message topic, you have to define a new paragraph (see [Message topics](#)).

Example:

```
-----
msg_topic
name os
keywords linux,mac-os,nt,xp
title Operating System
-----
```

Parameters `msg_topic.name` and `msg_topic.title` are mandatory. `msg_topic.title` is used on the web interface (other is not allowed for the `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 not mandatory.

msg_topic_keywords_apply_on

The `msg_topic_keywords_apply_on` parameter defines which part of the message is used to perform automatic tagging (see [Message topics](#)).

Example:

```
-----
msg_topic_key_apply_on subject
-----
```

Its values can be: `subject`, `body` and `subject_and_body`.

msg_topic_tagging

The `msg_topic_tagging` parameter indicates if tagging is optional or required for a list. (See [Message topics](#))

Example:

```
-----
msg_topic_tagging optional
-----
```

Its values can be "optional", "required_moderator" or "required_sender". When topic is required, a tagging request is sent to the list moderator or to the message sender depending of this parameter value.

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 email 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 indicates the email address to be 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`, the Reply-To: SMTP header field will be overwritten.

Example:

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

anonymous_sender

`anonymous_sender value`

If this parameter is set for a list, all messages distributed through the list are made 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:).

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
-----
```

rfc2369_header_fields

`rfc2369_header_fields help,archive` (Default value: `rfc2369_header_fields` `sympa.conf` parameter)

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

remove_headers

(Default value: `remove_headers` `sympa.conf` parameter)

You can define the list of SMTP header fields that should be removed from incoming messages. Check the [equivalent sympa.conf parameter documentation](#) for further details.

remove_outgoing_headers

(Default value: `remove_outgoing_headers` `sympa.conf` parameter)

You can define the list of SMTP header fields that should be removed before Sympa distributes a message to list members. Check the [equivalent sympa.conf parameter documentation](#) for further details.

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 the `[0st.sequence%]` or `[0st.name%]` variables that will get instantiated.

Example:

```
-----
custom_subject sympa-users
-----
```

Other example:

```
-----
custom_subject newsletter num [0st.sequence%]
-----
```

footer_type

`footer_type mime | append` (Default value: `mime`)

This parameter is optional. List owners may decide to add message headers or footers to messages sent through 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. `/home/sympa/exp1/mylist/message.footer.mime` will be ignored. Headers/footers may be appended to text/plain messages only.

info

The scenario definition of who can view the info page of a list.

- `info open` (default)
- `info private`

subscribe

(Default value: `open`)

The `subscribe` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

The `subscribe` parameter defines the rules for subscribing to the list. Predefined authorization

scenarios are:

- `subscribe auth` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.auth>]);
- `subscribe auth_notify` (view [http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.auth_notify]);
- `subscribe auth_owner` (view [http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.auth_owner]);
- `subscribe closed` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.closed>]);
- `subscribe intranet` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.intranet>]);
- `subscribe intranetorowner` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.intranetorowner>]);
- `subscribe open` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.open>]);
- `subscribe open_notify` (view [http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.open_notify]);
- `subscribe open_quiet` (view [http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.open_quiet]);
- `subscribe owner` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.owner>]);
- `subscribe smime` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.smime>]);
- `subscribe smimeorowner` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/subscribe.smimeorowner>]).

unsubscribe

(Default value: `open`)

The `unsubscribe` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

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` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/unsubscribe.auth>]);
- `unsubscribe auth_notify` (view [http://www.sympa.org/distribution/current/src/etc/scenari/unsubscribe.auth_notify]);
- `unsubscribe closed` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/unsubscribe.closed>]);
- `unsubscribe open` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/unsubscribe.open>]);
- `unsubscribe open_notify` (view [http://www.sympa.org/distribution/current/src/etc/scenari/unsubscribe.open_notify]);
- `unsubscribe owner` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/unsubscribe.owner>]).

add

(Default value: `owner`)

`add` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

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

- `add auth` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/add.auth>]);
- `add closed` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/add.closed>]);
- `add owner` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/add.owner>]);
- `add owner_notify` (view [http://www.sympa.org/distribution/current/src/etc/scenari/add.owner_notify]).

del

(Default value: `owner`)

The `del` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

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

- `del auth` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/del.auth>]);
- `del closed` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/del.closed>]);

- `del owner` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/del.owner>]);
- `del owner_notify` (view [http://www.sympa.org/distribution/current/src/etc/scenari/del.owner_notify]).

invite

(Default value: `owner`)

The `invite` command is used to invite someone to subscribe. It should be preferred to the `add` command in most cases. This parameter defines who can use it. The privilege uses scenario specification.

review

(Default value: `owner`)

`review` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

This parameter specifies who can use `REVIEW` (see [User commands](#)), administrative requests.

Predefined authorization scenarios are:

- `review closed` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.closed>]);
- `review intranet` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.intranet>]);
- `review listmaster` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.listmaster>]);
- `review owner` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.owner>]);
- `review private` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.private>]);
- `review public` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.public>]).

remind

(Default value: `owner`)

The `remind` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

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

- `remind listmaster` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/remind.listmaster>]);
- `remind owner` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/remind.owner>]).

shared_doc

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

d_read

(Default value: `private`)

The `d_read` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

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

Predefined authorization scenarios are:

- `d_read owner` (view [http://www.sympa.org/distribution/current/src/etc/scenari/d_read.owner]);
- `d_read private` (view [http://www.sympa.org/distribution/current/src/etc/scenari/d_read.private]);
- `d_read p` (view [http://www.sympa.org/distribution/current/src/etc/scenari/d_read.p]);
- `d_read public` (view [http://www.sympa.org/distribution/current/src/etc/scenari/d_read.public]).

d_edit

(Default value: `owner`)

The `d_edit` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

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` (view [http://www.sympa.org/distribution/current/src/etc/scenari/d_edit.editor]);
- `d_edit owner` (view [http://www.sympa.org/distribution/current/src/etc/scenari/d_edit.owner]);
- `d_edit private` (view [http://www.sympa.org/distribution/current/src/etc/scenari/d_edit.private]);
- `d_edit p` (view [http://www.sympa.org/distribution/current/src/etc/scenari/d_edit.p]);

- `d_edit public` (view [http://www.sympa.org/distribution/current/src/etc/scenari/d_edit.public]).

Example:

```
shared_doc
d_read      public
d_edit      private
```

quota

`quota number-of-Kbytes`

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

Archive related

Sympa show archive both by email and web interface. In versions prior to 5.2, archives were duplicated. Mail archives were stored in the `/home/sympa/expl/mylist/archives/` directory.

Web archives are accessed through the web interface (with access control), they are stored in a directory defined in "wvsympa.conf" ([parameter arc_path](#)). Version 5.2 and later use only this archive repository.

archive (OBSOLETE)

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 (`/home/sympa/expl/mylist/archives/`), used to store these documents.

`access private | public | owner | closed`

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

web_archive

If the `config` file contains a `web_archive` paragraph, Sympa will copy all messages distributed via the list to the `queueoutgoing` spool. It is intended to be used with *WWSympa's* 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
```

web_archive.access

The `access_web_archive` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

Predefined authorization scenarios are:

- `access closed` (view [http://www.sympa.org/distribution/current/src/etc/scenari/access_web_archive.closed]);
- `access intranet` (view [http://www.sympa.org/distribution/current/src/etc/scenari/access_web_archive.intranet]);
- `access listmaster` (view [http://www.sympa.org/distribution/current/src/etc/scenari/access_web_archive.listmaster]);
- `access owner` (view [http://www.sympa.org/distribution/current/src/etc/scenari/access_web_archive.owner]);
- `access private` (view [http://www.sympa.org/distribution/current/src/etc/scenari/access_web_archive.private]);
- `access public` (view [[http://www.sympa.org/distribution/current/src/etc/scenari/access_web_archive.public](#)]).

[http://www.sympa.org/distribution/current/src/etc/scenari/access_web_archive.public).

web_archive.quota

quota *number-of-Kbytes*

This parameter specifies the disk quota for the list's web archive, in kilobytes. This parameter's default is the `default_archive_quota` `sympa.conf` parameter. If quota is exceeded, messages are no more archived and list owners are notified. When the archive reaches 95%, list owners are warnt.

web_archive.max_month

"max_month" parameter specify the maximum number of archives packet created. Old month are removed when new month is created.

archive_crypted_msg

(Default value: `cleartext`)

archive_crypted_msg `cleartext` | `decrypted`

This parameter defines Sympa's behavior when archiving S/MIME encrypted messages. If set to `cleartext`, the original encrypted form of the message will be archived; if set to `decrypted`, a decrypted message will be archived. Note that this applies to both mail and web archives, and also to digests.

Bounce related

bounce

This paragraph defines bounce management parameters (you may also read the [section that describe how Sympa deal with bounces](#)) :

- `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: `daily`)
Name of the task template used to remove old bounces. Useful to remove bounces for a subscriber email if some messages are distributed without receiving new bounces. In this case, the subscriber email seems to be OK again. Active if `task_manager.pl` is running.

Example:

```
## Owners are warnt with 100ouncing addresses
## message distribution is halted with 200ouncing rate
bounce
warn_rate 10
halt_rate 20
```

bouncers_level1

- `rate`
(Default value: `default_bounce_level1_rate` `sympa.conf` parameter)
Each bouncing user has a score (from 0 to 100). This parameter defines the lower score for a user to be a level 1 bouncing user. For example, with default values, users with a score between 45 and 75 are level 1 bouncers.
- `action` `remove_bouncers` | `notify_bouncers` | `none`
(Default value: `notify_bouncers`)
This parameter defines which task is automatically applied on level 1 bouncing users: for example, automatically notify all level 1 bouncers.
- `notification` `none` | `owner` | `listmaster`
(Default value: `owner`)
When an automatic task is performed on level 1 bouncers, a notification email can be sent to listowners or listmasters. This email contains the addresses of the users concerned and the name of the action perform.

bouncers_level2

- **rate**
(Default value: `default_bounce_level2_rate` sympa.conf parameter)
Each bouncing user has a score (from 0 to 100). This parameter defines 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** `remove_bouncers` | `notify_bouncers` | `none`
(Default value: `remove_bouncers`)
This parameter defines which task is automatically applied on level 2 bouncing users: for example, automatically notify all level 2 bouncers.
- **notification** `none` | `owner` | `listmaster`
(Default value: `owner`)
When an automatic task is performed on level 2 bouncers, a notification email can be sent to listowners or listmasters. This email contains the addresses of the users concerned and the name of the action performed.

Example:

```
## All bouncing addresses with a score between 75 and 100
## will be unsubscribed, and listmaster will receive an email

Bouncers level 2

rate:75 Points

action: remove\_bouncers

Notification: Listmaster
```

welcome_return_path

`welcome_return_path` `unique` | `owner`
(Default value: `welcome_return_path_robot` parameter)
If set to `unique`, the welcome message is sent using a unique return path in order to remove the subscriber immediately in case of bounce. See the `welcome_return_path` sympa.conf parameter.

remind_return_path

`remind_return_path` `unique` | `owner`
(Default value: `remind_return_path_robot` parameter)
Same as `welcome_return_path`, but applied to remind messages. See the `remind_return_path` sympa.conf parameter.

verp_rate

(Default value: `verp_rate_host` parameter)
See [VERP](#) for more information on VERP in Sympa.

When `verp_rate` is null, VERP is not used; if `verp_rate` is `100 0.000000E+00RP` is always in use.

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

Data source related

user_data_source

(Default value: `include2`, if using an RDBMS)

`user_data_source` `file` | `database` | `include` | `include2`

Starting with Sympa 5.3.x `include` is interpreted as `include2`. Since Sympa 5.4.x `include2` is the only supported value for this parameter.

Background : *In the former days Sympa did not use a RDBMS and subscribers informations were stored in flat `subscribers` files. We then introduced the ability to include members defined in an external data source and also the optional use of a RDBMS to store subscribers. Therefore we created the `'user_data_source'` parameter. We ended up merging the `'database'` and `'include'` features with `'include2'`. The goal was then to give up the `'user_data_source'` parameter and we have almost reached this goal. Starting with Sympa 5.3.x the `'include'` mode is considered a synonym for `'include2'` and more recently we have removed the `'file'` and `'database'` modes in the development version of Sympa. This means that in Sympa 5.4 the only supported mode will be `'include2'`. Note that migration process has been automated.*

Sympa allows the mailing list manager to choose how Sympa loads subscriber and administrative 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 introduced to allow 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 ease 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 instead of text files. It will improve performance and solve possible conflicts between Sympa and *WWSympa*. Please refer to [Sympa and its database](#).

- `user_data_source` include

Here, subscribers are not defined *extensively* (enumeration of their email addresses) but *intensively* (definition of criteria subscribers must satisfy). Includes can be performed by extracting email 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 maintained in a DB File but in the main database instead. The behavior of the cache is detailed in the database chapter (see [Management of the include cache](#)). This is the only mode that runs the database for administrative data in the database.

ttl

(Default value: 3600)

`ttl delay_in_seconds`

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

distribution_ttl

`distribution_ttl delay_in_seconds`

Before some actions it is useful to make sure that the user's list is up-to-date. To avoid to execute synchronization any time these actions are performed, this parameter defines the delay since the last synchronization after which the user's list will be updated before performing the action.

The actions for which this parameter is checked are :

- list members review
- message distribution

include_list

`include_list listname`

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 would result in an infinite loop.

Example:

```
-----
include_list local-list
-----
```

Other example:

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

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 a 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 to an infinite loop making cross includes.

For this operation, one Sympa site acts as a server while the other acts as a client. On the server side, the only setting needed is to give permission to the remote Sympa to review the list. This is controlled 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 offer an easy and secure client authentication, HTTPS is the only protocol currently supported. An additional parameter is needed: the name of the certificate (and the private key) to be used:

- **cert list**

The certificate to be used is the list certificate (the certificate subject distinguished name email is the list address). The 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 looks like `sympa@my.domain` and files are located in the virtual host `etc` directory if a virtual host is used; otherwise, they are located in `/home/sympa/etc`.

include_sql_query

include_sql_query

It is used to start 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 email 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 connection. 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 data source. It will be used within the REVIEW page to indicate from which datasource each list member comes (useful when having multiple data sources).

- **f_dir** */var/csvdir*

This parameter is optional. It is only used when accessing a CSV data source. When connecting to a CSV data source, 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
```

include_ldap_query

include_ldap_query

This paragraph defines parameters for a LDAP query returning a list of subscribers. 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 useful 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*, the LDAPS protocol is used.
- **ssl_version** *sslv2/sslv3/tls* (Default value: *sslv3*)
If using SSL, this parameter defines whether SSL or TLS is used.
- **ssl_ciphers** *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 allows all ciphers, even those that do not 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 email address(es) in the object returned.
- **select** *first | all* (Default value: *first*)
Defines whether to use only the first address, or all the addresses, in case 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
```

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 email 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 useful 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 includes 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*, the LDAPS protocol is used.
- **ssl_version** *sslv2/sslv3/tls* (Default value: *sslv3*)
If using SSL, this parameter defines whether SSL or TLS is used.
- **ssl_ciphers** *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::LDAP for ciphers is *ALL*, which allows all ciphers, even those that do not 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 object returned, 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 email address(es) in the objects returned 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

```

include_file

`include_file path_to_file`

The file should contain one email 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

```

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. 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. It is only used if HTTP basic authentication is required to access the remote file.
- `passwd user_passwd`
This entry is optional. It is 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

```

Command related

remind_task

(Default value: no default value)

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

Example:

```

remind_task annual

```

expire_task

(Default value: no default value)

This parameter states which model is used to create an `expire` task. An `expire` task regularly checks the subscription (or subscription renewal) date of subscribers and asks them to renew their subscription. If they do not, they are deleted.

Example:

```

expire_task annual

```

review

(Default value: `owner`)

The `review` parameter is defined by an authorization scenario (see [Authorization scenarios](#)).

This parameter specifies who can use the `REVIEW` command (see [User commands](#)), administrative requests.

Predefined authorization scenarios are:

- `review closed` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.closed>]);
- `review intranet` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.intranet>]);
- `review listmaster` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.listmaster>]);
- `review owner` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.owner>]);
- `review private` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.private>]);
- `review public` (view [<http://www.sympa.org/distribution/current/src/etc/scenari/review.public>]).

List tuning

max_size

(Default value: `max_size robot` parameter)

`max_size number-of-bytes`

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

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 message sender addresses. 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.

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 the review page.

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 (see [owner](#)).

Example:

```
-----
cookie secret22
-----
```

custom_vars

(Default value: empty)

You can create an unlimited number of custom parameters to be used with authorization scenarios, web and mail templates.

- `name`: the name of the custom parameter. Can be any character string;
- `value`: the value given to this custom parameter. Can be any scalar value.

Example:

```
-----
custom_vars
name    sisterList
value   math-teachers
-----
```

See [this parameter usage in Sympa](#).

custom_attribute

(Default value: empty)

This parameter allows the creation of custom user attributes. These attributes values are stored in a XML fragment. See the description of the [user custom attributes functionality](#).

This XML fragment has the form:

```
<?xml version="1.0" encoding="UTF-8" ?>
<custom_attributes>
  <custom_attribute id="accr">
    <value> ultra-violet</value>
  </custom_attribute>
  <custom_attribute id="pt">
    <value>0</value>
  </custom_attribute>
</custom_attributes>
```

- **id**: the value of the `id` attribute in the `custom_attribute` element;
- **name**: the label used for this attribute in the subscription form;
- **comment**: a text displayed in the subscription form to give the users any additional informations that you would find relevant about this attribute (a description of the values to fill in, for example);
- **type** (`string` | `text` | `integer` | `enum`): the type of data this attribute contains. The subscription form will control that the data filled by the user belong to this category. `enum` means that the value will be selected amongst a set allowed values you must define.
- **enum_values**: if `type` has the value `enum`, you must specify at least one allowed value. The web subscription form will contain a selection list with the values you specified.
- **optional** (`required` | `optional`): use "required" if this attribute must be filled by the user, "optional" if it is not mandatory.

merge_feature

(Default value: off)

`merge_feature on / off`

Customizing messages. If set to 'on', allows the subscribers to send messages with custom users attributes.

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.

Spam protection

spam_protection

(Default value: javascript)

There is a need to protect the Sympa website against spambot which collect email addresses in public websites. Various methods are available into Sympa and you can choose from the `spam_protection` and `web_archive_spam_protection` parameters. Possible value are:

- **javascript**: the address is hidden using a Javascript. Users who enable Javascript can see nice mailto addresses where others have nothing.
- **at**: the '@' char is replaced by the string 'AT'.
- **none**: no protection against spammers.

web_archive_spam_protection

(Default value: javascript)

The same as `spam_protection`, but restricted to the web archive. An additional value is available: `cookie`, which means that users must submit a small form in order to receive a cookie before browsing the archive. This blocks all robots, even those from search engines.

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.

Message topic definition in a list

Available message topics are defined by list parameters. For each 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. Otherwise, to use automatic tagging, you should define keywords (see [msg_topic](#)). To define which part of the message is used for automatic tagging, you have to define the `msg_topic_keywords_apply_on` list parameter (see [msg_topic_keywords_apply_on](#)). Tagging a message can be optional or required, depending on the [msg_topic_tagging_list_parameter](#).

Subscribing to message topics for list subscribers

This feature is only available with the `normal` delivery mode. Subscribers can select a message topic to receive messages tagged with this topic. To receive messages that were not tagged, users can subscribe to the topic `other`. The message topics selected by a subscriber are stored in the Sympa database (`subscriber_table` table).

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 resulting 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 [Authorization scenarios](#)). If message tagging is required and if it was not yet performed, Sympa will ask the list moderator.

Tagging a message will create a topic information file in the `/home/sympa/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 email filters.

Multipart/alternative

If available, list members can select the `TXT` or `HTML` reception modes. In these modes, the list member will receive the selected version of a message if the message's content-type is multipart/alternative.

Shared documents

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

WWSympa's shared web features are fairly rudimentary. It is not our aim to provide a sophisticated tool for web publishing, such as those provided by products like *Rearsite*. It is nevertheless very useful to be able to define privileges 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 why 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 document web space, each document is linked to a set of specific control information: its access rights.

A list's shared documents are stored in the `/home/sympa/expl/mylist/shared` directory. This directory is either created via the `Create shared` web admin feature, or at list creation time, if the list `shared_doc` parameter is set. If you don't want the document repository to be created, you should remove the `shared_doc` parameter from the corresponding `create_list_templates/xx/config.tt2` files.

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

The three kinds of operations on a document

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

- the read operation;
- the edit operation;
- the control operation.

The read operation

If applied to a directory, it opens it and lists its contents (only the sub-documents the user is authorized to "see").

If applied to a file, it downloads it, and in the case of a viewable file (*text/plain*, *text/html*, or image), displays it.

The edit operation

It allows:

- subdirectory creation;
- file uploading;
- file unzipping;
- description of a document (title and basic information);
- online editing of a text file;
- document (file or directory) deletion. Directories can be deleted only if they are 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 online, describe it, replace or remove it.

The control operation

The control operation is directly linked to the notion of access rights. If we want shared documents to be secure, we have to control the access to them. Not everybody must be authorized to perform every operation on 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 over a document. Another possible control action on a document is therefore specifying who owns it.

The description file

The information (title, owner, access rights...) related 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`.

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      <email 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 that 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

```

The predefined authorization scenarios

The public scenario

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

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.

The scenario owner

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

The scenario editor

The scenario `editor` is for a moderated shared space for editing. Every subscriber of the list is allowed to edit 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 notified and when the document is installed, the author is notified too. In case of reject, the editor can notify the author or not.

Access control

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

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

Listmaster and privileged owners

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

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 the users authorized to edit a list's configuration may change access rights on its `shared` directory.

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, he/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 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>)
```

Shared document actions

The shared web feature has called for some new actions.

- **action D_ADMIN**
Creates the shared space, closes 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 the document is a folder, it lists all the subdocuments that can be read. If it is a file, it displays it if it is viewable, else downloads it to disk. If the document to be read contains a file named `index` or `index.htm`, and if the user has no permissions other than read on all subdocuments contained, 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**
Unzips 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 can be performed on the document.

Template files

The following template files have been created for the shared document web space.

`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 into it. If the directory is editable without moderation, it displays buttons to create a new subdirectory 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.

`d_editfile.tt2`

The page used to edit a file. If for a text file, allows it to be edited online. This page also enables another file to be substituted in its place.

`d_control.tt2`

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

`d_upload.tt2`

The page to upload a file is only used when the name of the file already exists.

`d_properties.tt2`

This page is used to edit the description file and to rename it.

Sympa and its database

Sympa requires a RDBMS to run. Currently you can use one of the following RDBMS: MySQL (version 4.1 minimum), SQLite, PostgreSQL, Oracle, Sybase. Interfacing with other RDBMS requires only a few changes in the code, since the API used, DBI [<http://www.symbolstone.org/technology/perl/DBI/>] (DataBase Interface), has DBD (DataBase Drivers) for many RDBMS.

Sympa stores three kinds of information in the database, each in one table:

- user preferences and passwords are stored in the `user_table` table;
- list subscription information is stored in the `subscriber_table` table, along with subscription options. This table also contains the cache for included users ;
- list administrative information is stored in the `admin_table` table, along with owner and editor options. This table also contains the cache for included owners and editors.
- logs events are stored in the `logs_table` ; list owners can browse the events for their list through the web interface.

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 the RDBMS you chose! Check the "DBI" Module Availability [<http://www.symbolstone.org/technology/perl/DBI/>].

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 (`Mysql-MySQL-modules` for MySQL) are distributed as CPAN modules. Refer to [Installing PERL and CPAN modules](#) for installation details of these modules.

Creating a Sympa DataBase

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 between information relating to a person (email, real name, password) and his/her subscription options (list concerned, date of subscription, delivery mode, 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 email.

The table concerning owners and editors, the `admin_table`, is built on the same model as the `subscriber_table`. It contains owner and editor options (list concerned, administrative role, date of "subscription", delivery mode, private information, gecoss and profile option for owners).

Database automatic creation and update

At startup, the `sympa.pl` process will check if the database (configured in `sympa.conf`) is available and if it has the expected structure. If not, `sympa.pl` process will create the database or update its structure for you. **Note however that this feature is available with mysql only**, integration for other RDBMS is less complete. Note that this automated process requires that the `mysql root` does not use a password ; if it does, disable it before you run `sympa.pl` for the first time.

The automatic procedure will also grant privileges to the `db_user` you've declared in `sympa.conf`.

Database manual 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,
  gecoss_user         varchar (150),
  password_user       varchar (40),
  last_login_date_user int(11),
  last_login_host_user varchar(60),
  wrong_login_count_user int(11),
  cookie_delay_user   int,
  lang_user           varchar (10),
  attributes_user     text,
  data_user           text,
  PRIMARY KEY (email_user)
);

CREATE TABLE subscriber_table (
  list_subscriber     varchar (50) NOT NULL,
  user_subscriber     varchar (100) NOT NULL,
  custom_attribute_subscriber text,
```

- SQLite database creation script:

```
CREATE TABLE user_table (
  email_user          text NOT NULL,
  gecoss_user         text,
  password_user       text,
  last_login_date_user integer,
  last_login_host_user text,
  wrong_login_count_user integer,
  cookie_delay_user   integer,
  lang_user           text,
  attributes_user     text,
  data_user           text,
  PRIMARY KEY (email_user)
);

CREATE TABLE subscriber_table (
  list_subscriber     text NOT NULL,
  user_subscriber     text NOT NULL,
  custom_attribute_subscriber text,
  robot_subscriber    text NOT NULL,
  date_subscriber     timestamp NOT NULL,
  update_subscriber   timestamp,
  visibility_subscriber text,
  reception_subscriber text,
  topics_subscriber   text,
```

- 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),
    last_login_date_user int4,
    last_login_host_user varchar (60),
    wrong_login_count_user int4,
    lang_user           varchar (10),
    attributes_user     varchar (255),
    data_user           varchar (255),
    CONSTRAINT ind_user PRIMARY KEY (email_user)
);

DROP TABLE subscriber_table;
CREATE TABLE subscriber_table (
    list_subscriber     varchar (50) NOT NULL,
```

- 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   ,
    last_login_date_user numeric,
    last_login_host_user varchar (60),
    wrong_login_count_user numeric,
    cookie_delay_user   numeric              null   ,
    lang_user           varchar(10)           null   ,
    attributes_user     varchar(255)         null   ,
    data_user           varchar(255)         null   ,
    constraint ind_user primary key (email_user)
)
go
```

- Oracle database creation script:

```
## Oracle Database creation script
## Fabien Marquois <fmarquois@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),
    last_login_date_user number,
    last_login_host_user varchar2(60),
    wrong_login_count_user number,
    cookie_delay_user   number,
```

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

Example:

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

Setting database privileges

We strongly recommend that you restrict access to the 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;
```

Importing subscriber data

Importing data from a text file

You can import subscription data into the database from a text file having one entry per line: the first field is an email address, the second (optional) field is the free form name. Fields are space-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 [sympa.pl](#)).

Importing data from subscribers files

If a mailing list was previously set up to store subscribers into a `subscribers` file (the default mode in versions older than 2.2b), you can load subscriber 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`; subscriber data will be loaded into the database at the same time.

If the `subscribers` file is large, 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.

Extending database table format

You can easily add other fields to the three tables, they will not disturb Sympa because it lists explicitly 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 [db_additional_subscriber_fields](#) and [db_additional_user_fields](#)).

Sympa logs in the database

The `logs_table` database table has been introduced with release 5.3 of Sympa. This DB table gathers some kind of logs/journals that Sympa want to keep track of. Only events that changes Sympa's state are logged ; this includes member subscription/removal, message distribution/moderation, bounces handling, user authentication on the web interface. The content of the database can then be searched by list owners ; privacy is enforced to prevent a list owner to access information from other lists.

The `logs_table` table is purged to prevent DB size to diverge. The retention period of log entries is defined by the `logs_expiration_period` parameter

Below is a description of each field of the `logs_table` table :

- `id_logs`: primary key for the table.
- `date_logs`: epoch date representing the time when the action was performed.
- `robot_logs`: the robot the action relates to. It may be empty if no specific virtual host is concerned.
- `list_logs`: the list the action refers to. It may be undefined if the action does not refer to a specific list (like authentication related actions).
- `action_logs`: an identifier for the action performed. Each action belongs to a group of actions (authentication related, subscription related, etc). The list of actions and the group they relate to is defined in the `Log.pm` perl module.
- `parameters_logs`: parameters of the action. It is an optional comma-separated list of parameters.
- `user_email_logs`: this field tells who is performing the action, if authenticated.
- `target_email_logs`: this parameter represents the email address that is mainly concerned by the action ; it may be empty. If a list owner adds a new member to his list, the `target_email_logs` field will contain the new member email address.
- `client_logs`: IP address of the user performing the action, may be empty if action is performed via the

mail interface.

- `msg_id_logs`: Message-ID of the message, if the action applies to a message.
- `status_logs`: the status of the action ; it will contain one of the following values : `success`, `error`.
- `error_type_logs`: If the action failed, this field an error identifier representing the error.
- `daemon_logs`: tells which process has performed the action. It may be one of `bounced`, `sympa`, `wwsympa`

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'll need to tell Sympa where its database is located through the related `sympa.conf` parameters : `db_type`, `db_name`, `db_host`, `db_user`, `db_passwd`.

If you are interfacing Sympa with an Oracle database, note that :

1. the `db_name` corresponds to the Oracle SID.
1. you'll need to set the `ORACLE_HOME` environment variable through the `db_env` `sympa.conf` parameter

All your lists are now configured to use the database, unless you set the 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), in case the database were to become inaccessible.

WWSympa, Sympa's web interface

WWSympa is Sympa's web interface.

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 [Template file format](#)). This makes internationalization of pages, as well as per-site customization, easier.

The code consists of one single PERL CGI script, `wwsympa.fcgi`. To enhance performances 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, passwords 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 email 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;
1. evaluate user's permissions for the requested action;
1. process the requested action;
1. set up variables resulting from the action;
1. parse the HTML template files.

Web server setup

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.

Default behaviour

Until version 5.3: SetuidPerl

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

Starting version 5.4: C wrapper

The C wrapper presented in the preceding section will be automatically built starting version 5.4.

The `wwsympa.fcgi` is wrapped in a small C script, `wwsympa-wrapper.fcgi`, in order to avoid to use the - insecure and no longer maintained - `SetuidPerl` mode.

Alternatives (all versions)

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: /home/sympa/bin/wwsympa.fcgi
-----
```

You should also check that the `requiretty` and `env_reset` flags are not set in the `sudoers` configuration file :

```
-----
#Defaults    requiretty
-----
```

```
-----
#Defaults    env_reset
-----
```

With `requiretty` set, `sudo` would only run when the user is logged in to a real tty; with `env_reset` set, most of your environment variables would be ignored... including your server name, the URL requested, etc.

Dedicated Apache server

Run a dedicated Apache server with `sympa.sympa` as `uid.gid` (the Apache default is `apache.apache`);

Apache suExec

manual/fulltest.txt · Last modified: 2009/08/27 15:23 (external edit)