Sympa, a Mailing List Management Software - Reference manual

Software version: 5.3

Authors: Serge Aumont, Olivier Salaün, Christophe Wolfhugel

Manual release date: November, 6th 2006

sympa.conf parameters part2

sympa.conf parameters part3

Sympa and its database

List creation, edition and removal

Lists Families

List configuration parameters

List definition

Sending/receiving setup

Privileges

Archives

- Archive related
  - archive
  - web_archive
  - access
  - quota
  - max_month
  - archive_crypted_msg

Bounce management

- Bounce related
  - bounce
  - bouncers_level1
  - bouncers_level2
  - welcome_return_path
  - remind_return_path
  - verp_rate

Data sources setup

- Data source related
  - user_data_source
  - ttl
  - include_list
  - include_remote_sympa_list
  - include_sql_query
  - include_ldap_query
  - include_ldap_2level_query
  - include_file
  - include_remote_file

Others

- Command related
  - remind_task
  - expire_task
  - review
- List tuning
  - max_size
  - loop_prevention_regex
  - pictures_feature
  - cookie
  - priority
- Spam protection
  - spam_protection
  - web_archive_spam_protection

Using Sympa commands

About this document ...

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 (5.3a.10) of Sympa.

License

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

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 robot should include. While most Sympa features have their equivalents in other mailing list applications, Sympa is unique in including features in a single software package, including:

- **High speed distribution processing** and **load control**. Sympa can be tuned to allow the system administrator to control the amount of computer resources used. Its optimized algorithm allows:
  - the use of your preferred SMTP engine, e.g. Sendmail, qmail or Postfix,
  - tuning of the maximum number of SMTP child processes,
  - grouping of messages according to recipients' domains, and tuning of the grouping factor,
  - detailed logging;
- **Multilingual** user interface. The full user/admin interface (mail and web) is internationalized. Translations are gathered in a standard PO file;
- **Template-based** user interface. Every web page and service message can be customized 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**);
- The **sending process is controlled** on a per-list basis. The list definition allows a number of different actions for each incoming message. A `private` list is a list where only subscribers can send messages. A list configured using `privateoreditorkey` mode accepts incoming messages from subscribers, but will forward any other message (i.e. sent by non-subscribers) to the editor with a unique secret numeric key that will be used by the editor to *reject* or *distribute* the message. For details about the different sending modes, refer to the send parameter. The sending process configuration (as well as most other list operations) is defined using an **authorization scenario**. Any listmaster can define new authorization scenarios in order to complement the 20 predefined configurations included in the distribution (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**.

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 up to handling. The initial version of Sympa brought authentication, the flexible management of commands, high performances in internal data access, and object oriented code for easy code maintenance.

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

Other 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, ...

Mailing lists and support

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

There are also a few mailing-lists about Sympa:

- `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-translation(@)cru.fr`, Sympa translators

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

> `subscribe` *Listname Firstname Name*

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

You may also refer to the Sympa homepage; there you will find the http://www.sympa.org/distribution/latest version, the FAQ and so on.

Organization

Here is a snapshot of what Sympa looks like once it has settled down 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 raisonnable default value comming from Sympa distribution.

The following referenec manual assume a particular locations for all files and directory. Note that that binary distribution usually change thoses location according to the operating system file organization. When installing Sympa from source kit, *configure* can be called with command option in order to change all default file location.

- `/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 to 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.

Binaries

- `sympa.pl`
  The main daemon; it processes commands and delivers messages. Continuously scans the `msg/` spool.
- `sympa_wizard.pl`
  A wizard to edit `sympa.conf` and `wwsympa.conf`. Maybe it is a good idea to run it at the beginning, but these 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).
- `nrcpt_by_domain`
  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 case 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 particular domains. The file must contain a list of domains followed by the maximum number of recipients per SMTP session. Example:
- `data_structure.version`
  This file is automatically created and maintained by Sympa itself. It contains the current version of your Sympa service and is used to detect upgrades and trigger maintenance procedures such as database structure changes.

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

Spools

See Spool related for spool definition in `sympa.conf`.

- `/home/sympa/spool/auth/`
  For storing messages until they have been confirmed.
- `/home/sympa/spool/bounce/`
  For storing incoming bouncing messages.
- `/home/sympa/spool/digest/`
  For storing message digests before they are sent.
- `/home/sympa/spool/mod/`
  For storing unmoderated messages.
- `/home/sympa/spool/msg/`
  For storing incoming messages (including commands).
- `/home/sympa/spool/msg/bad/`
  Sympa stores rejected messages in this directory
- `/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.
- `/home/sympa/spool/task/`
  For storing all tasks created.
- `/home/sympa/spool/outgoing/`
  `sympa.pl` dumps messages in this spool to await archiving by `archived.pl`.

- `/home/sympa/spool/topic/`
  For storing topic information files.

Roles and privileges

You can assign roles to users (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 less), 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 persons 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).

sympa.pl

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

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

- `– debug` | `-d`
  Sets Sympa in debug mode and keeps it attached to the terminal. Debugging information is output to STDERR, along with standard log information. Each function call is traced. Useful while reporting a bug.
- `service` *process_command* | *process_message* | *process_creation*
  Sets Sympa daemon to process only message distribution (process_message) or only commands (process_command) or list creation requests (process_creation).
- `– config` *`config_file`* | `-f` *`config_file`*
  Forces Sympa to use an alternative configuration file. Default behavior is to use the configuration file as defined in the Makefile ($CONFIG).
- `– mail` | `-m`
  Sympa will log calls to sendmail, including recipients. Useful for keeping track of each mail sent (log files may grow faster though).
- `– lang` *`catalog`* | `-l` *`catalog`*
  Set this option to use a language catalog for Sympa. The corresponding catalog file must be located in `~sympa/locale` directory.
- `– keepcopy` *`recipient_directory`* | `-k` *`recipient_directory`*
  This option tells Sympa to keep a copy of every incoming message, instead of deleting them. *recipient_directory* is the directory to store messages.
- `– create_list – robot` *`robotname`* `– – input_file` *`/path/to/list_file.xml`*
  Create the list described by the xml file, see List creation on command line with sympa.pl.
- `– close_list` *`listname@robot`*
  Closes the list (changing its status to closed), remove aliases (if `sendmail_aliases` parameter was set) and remove subscribers from DB (a dump is created in the list directory to allow restoring the list). When you are in a family context, refer to List family closure.
- `– dump` *`listname`* | *`ALL`*
  Dumps subscribers of a list or all lists. Subscribers are dumped in `subscribers.db.dump`.
- `– import` *`listname`*
  Imports subscribers in the *listname* list. Data are read from STDIN.
- `– lowercase`
  Lowercases e-mail addresses in database.
- `– help` | `-h`
  Print usage of sympa.pl.
- `– make_alias_file`
  Creates an aliases file in `/tmp/` with all list aliases (only list which status is ‘open‘). It uses the list_aliases.tt2 template.

- `– version` | `-v`

  Prints current version of Sympa.
- `– instanciate_family` ***familyname robotname*** `– – input_file` ***/path/to/family_file.xml***

  Instantiates the family *familyname*. See Lists families.
- `– close_family` ***familyname*** `– – robot` ***robotname***

  Closes the *familyname* family. See List family closure.
- `– 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.
- `– 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.
- `– 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.
- `– 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.

INIT script

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

This script starts these 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 of the `–mail` login 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).

Incompatible changes

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

Sympa is a long-term project, so some major changes may need some extra work. The following list 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 t
*****   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 I
<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:
------------------------------------------
perl module          from CPAN       STATUS
-----------          ---------       ------
Archive::Zip         Archive-Zip    OK (1.09   >= 1.05)
CGI                  CGI            OK (2.89   >= 2.52)
DB_File              DB_FILE        OK (1.806  >= 1.75)
Digest::MD5          Digest-MD5     OK (2.20   >= 2.00)
FCGI                 FCGI           OK (0.67   >= 0.67)
File::Spec           File-Spec      OK (0.83   >= 0.8)
IO::Scalar           IO-stringy     OK (2.104  >= 1.0)
LWP                  libwww-perl    OK (5.65   >= 1.0)
Locale::TextDomain   libintl-perl   OK (1.10   >= 1.0)
MHonArc::UTF8        MHonArc        version is too old ( < 2.4.6).
>>>>>>> You must update ''MHonArc'' to version '''' <<<<<<.
Setting FTP Passive mode
Description:
Install module MHonArc::UTF8 ? n
MIME::Base64         MIME-Base64    OK (3.05   >= 3.03)
MIME::Tools          MIME-tools     OK (5.411  >= 5.209)
Mail::Internet       MailTools      OK (1.60   >= 1.51)
Regexp::Common       Regexp-Common  OK (2.113  >= 1.0)
Template             Template-ToolkitOK (2.13   >= 1.0)
XML::LibXML          XML-LibXML     OK (1.58   >= 1.0)
```

```
Checking for OPTIONAL modules:
------------------------------------------
perl module          from CPAN       STATUS
-----------          ---------       ------
Bundle::LWP          LWP            OK (1.09   >= 1.09)
Constant subroutine CGI::XHTML_DTD redefined at /usr/lib/perl5/5.8.0/constant.pm line 108, <STDIN> li
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 a
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 \
            --with-confdir=/home/sympa-dev/etc \
            --with-mandir=/home/sympa-dev/man \
            --with-initdir=/home/sympa-dev/init \
              --with-piddir=/home/sympa-dev/pid
            --with-lockdir=/home/sympa-dev/lock \
            --with-sendmail_aliases=/home/sympa-dev/etc/sympa_aliases
```

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

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

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;
2. transfer the database;
3. edit the `/data_structure.version` on the new server; change the version value to reflect the old number;
4. start “sympa.pl -upgrade“, it will upgrade the database structure according 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‘re using, where you store aliases and whether you are managing virtual domains or not.

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 for mailing lists, a special mail alias is reserved for administrative requests, so that Sympa can be permanently accessible to users. The following lines must therefore be added to the `sendmail` alias file (often `/etc/aliases`):

```
sympa: ''| /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 no null for at least one list.

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

Don‘t forget to run `newaliases` after any change to the `/etc/aliases` file!

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

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

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:

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

The address `mylist-request` should correspond to the person responsible for managing `mylist` (the owner). Sympa will forward messages 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* (see presentationWWSympa) 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.

The script expects the following arguments :

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

Example:

```
/home/sympa/bin/alias_manager.pl add mylistcru.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; it needs to be customized with your own LDAP parameters.

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
```

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.

conf-parameters part1

conf-parameters part2

conf-parameters part3

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

Example:

```
domain cru.fr
```

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 sympa on the same host (a sympa test for example).

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

For example, if you put:

```
listmaster listmaster-test
```

you must modify the sympa aliases like this:

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

See Robot aliases for all aliases.

wwsympa_url

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

This is the root URL of *WWSympa*.

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

(Default value: `javascript`)

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

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`.

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; m
```

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 preparsed CSS file. In order to replace dynamic CSS files by these static files, set the `css_url` parameter.

After an upgrade, `sympa.pl` 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 file are made parsing a `web_tt2` file 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. The server admin module includes a CSS administration page that can help you to install static CSS files.

static_content_path

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

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) 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 picture 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 is defined by an authorization scenario (see Authorization scenarios).

Defines who can run a `REMIND *` command.

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 octual 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 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.

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

Example:

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

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.

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: `1`)

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

Example:

```
clean_delay_queue 3
```

clean_delay_queuemod

(Default value: `10`)

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

clean_delay_queueauth

(Default value: `3`)

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

clean_delay_queuesubscribe

(Default value: `10`)

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

clean_delay_queuetopic

(Default value: `7`)

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.

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 are now encoded in utf-8.

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

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

filesystem_encoding

(Default value: `utf-8`)

Example:

```
filesystem_encoding iso-8859-1
```

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

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 that have changed on disk in the meanwhile.

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.

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_optionsmysql_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.

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.

chk_cert_expiration_task

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

crl_update_task

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

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 Clam:

```
  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.

Sympa and its database

Most basic features of Sympa will work without a RDBMS, but WWSympa and bounced require a relational database. Currently you can use one of the following RDBMS: MySQL, SQLite, PostgreSQL, Oracle, Sybase. Interfacing with other RDBMS requires only a few changes in the code, since the API used, DBI (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 (if using `include2` mode);
- list administrative information is stored in the `admin_table` table if using `include2` mode, along with owner and editor options. This table also contains the cache for included owners and editors.

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.

Installing PERL modules

Sympa will use DBI to communicate with the database system and therefore requires the DBD for your database system. DBI and DBD::YourDB (Msql-Mysql-modules for MySQL) are distributed as CPAN modules. Refer to 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 but is used only in include2 mode. It contains owner and editor options (list concerned, administrative role, date of “subscription“, delivery mode, private information, gecos and profile option for owners).

Database creation

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

- MySQL database creation script:

```
## MySQL Database creation script
CREATE DATABASE sympa;
## Connect to DB
\r sympa
CREATE TABLE user_table (
      email_user          varchar (100) NOT NULL,
      gecos_user          varchar (150),
      password_user                 varchar (40),
      cookie_delay_user   int,
      lang_user           varchar (10),
      attributes_user               varchar(255),
      PRIMARY KEY (email_user)
);
CREATE TABLE subscriber_table (
      list_subscriber               varchar (50) NOT NULL,
      user_subscriber               varchar (100) NOT NULL,
      robot_subscriber    varchar (80) NOT NULL,
      date_subscriber               datetime NOT NULL,
      update_subscriber   datetime,
      visibility_subscriber         varchar (20),
      reception_subscribervarchar (20),
      topics_subscriber   varchar (200),
      bounce_subscriber   varchar (35),
      bounce_score_subscriber smallint (6),
      bounce_address_subscriber     varchar (100),
      comment_subscriber  varchar (150),
      subscribed_subscriber         int(1),
      included_subscriber int(1),
      include_sources_subscriber varchar(50),
      PRIMARY KEY (list_subscriber, user_subscriber, robot_subscriber),
      INDEX (user_subscriber,list_subscriber,robot_subscriber)
);
CREATE TABLE admin_table (
      list_admin                    varchar(50) NOT NULL,
      user_admin                    varchar(100) NOT NULL,
      robot_admin                   varchar(80) NOT NULL,
      role_admin                    enum('listmaster','owner','editor') NOT NULL,
      date_admin                    datetime NOT NULL,
      update_admin                  datetime,
      reception_admin     varchar(20),
      comment_admin                 varchar(150),
      subscribed_admin    int(1),
      included_admin      int(1),
      include_sources_admin         varchar(50),
      info_admin                    varchar(150),
      profile_admin                 enum('privileged','normal'),
      PRIMARY KEY (list_admin, user_admin, robot_admin, role_admin),
      INDEX (list_admin, user_admin,robot_admin,role_admin)
);
CREATE TABLE netidmap_table (
        netid_netidmap              varchar (100) NOT NULL,
      serviceid_netidmap                varchar (100) NOT NULL,
      robot_netidmap                  varchar (80) NOT NULL,
        email_netidmap              varchar (100),
        PRIMARY KEY (netid_netidmap, serviceid_netidmap, robot_netidmap)
);
CREATE TABLE logs_table (
      id_logs                       bigint(20) NOT NULL,
      date_logs           int(11) NOT NULL,
      robot_logs          varchar(80),
      list_logs           varchar(50),
      action_logs                   varchar(50) NOT NULL,
      parameters_logs               varchar(100),
      target_email_logs   varchar(100),
      user_email_logs               varchar(100),
      msg_id_logs                   varchar(255),
      status_logs                   varchar(10) NOT NULL,
      error_type_logs               varchar(150),
      client_logs                   varchar(100),
      daemon_logs                   varchar(10) NOT NULL,
      PRIMARY KEY (id_logs)
);
```

- SQLiteL database creation script:

```
  CREATE TABLE user_table (
        email_user          varchar (100) NOT NULL,
        gecos_user          varchar (150),
        password_user                 varchar (40),
        cookie_delay_user   integer,
        lang_user           varchar (10),
        attributes_user               varchar(255),
        PRIMARY KEY (email_user)
  );
  CREATE TABLE subscriber_table (
        list_subscriber               varchar (50) NOT NULL,
        user_subscriber               varchar (100) NOT NULL,
        robot_subscriber    varchar (80) NOT NULL,
        date_subscriber               timestamp NOT NULL,
        update_subscriber   timestamp,
        visibility_subscriber         varchar (20),
        reception_subscribervarchar (20),
        topics_subscriber       varchar (200),
        bounce_subscriber   varchar (35),
        bounce_address_subscriber     varchar (100),
        comment_subscriber  varchar (150),
        subscribed_subscriber         boolean,
        included_subscriber boolean,
        include_sources_subscriber varchar(50),
        bounce_score_subscriber integer,
        PRIMARY KEY (list_subscriber, user_subscriber, robot_subscriber)
  );
  CREATE INDEX subscriber_idx ON subscriber_table (user_subscriber,list_subscriber,robot_subscriber
  CREATE TABLE admin_table (
        list_admin                    varchar(50) NOT NULL,
        user_admin                    varchar(100) NOT NULL,
        robot_admin                   varchar(80) NOT NULL,
        role_admin                    varchar(15) NOT NULL,
        date_admin                    timestamp NOT NULL,
        update_admin                  timestamp,
        reception_admin     varchar(20),
        comment_admin                 varchar(150),
        subscribed_admin    boolean,
        included_admin      boolean,
        include_sources_admin         varchar(50),
        info_admin                    varchar(150),
        profile_admin                 varchar(15),
        PRIMARY KEY (list_admin, user_admin, robot_admin, role_admin)
  );
  CREATEINDEX admin_idx ON admin_table(list_admin, user_admin, robot_admin, role_admin);
  CREATE TABLE netidmap_table (
          netid_netidmap              varchar (100) NOT NULL,
        serviceid_netidmap      varchar (100) NOT NULL,
        robot_netidmap                  varchar (80) NOT NULL,
          email_netidmap              varchar (100),
          PRIMARY KEY (netid_netidmap, serviceid_netidmap, robot_netidmap)
  );
  CREATEINDEX netidmap_idx ON netidmap_table(netid_netidmap, serviceid_netidmap, robot_netidmap);
  CREATE TABLE logs_table (
        id_logs                       integer NOT NULL,
        date_logs           integer NOT NULL,
        robot_logs          varchar(80),
        list_logs           varchar(50),
        action_logs                   varchar(50) NOT NULL,
        parameters_logs               varchar(100),
        target_email_logs   varchar(100),
        user_email_logs               varchar(100),
        msg_id_logs                   varchar(255),
        status_logs                   varchar(10) NOT NULL,
        error_type_logs               varchar(150),
        client_logs                   varchar(100),
        daemon_logs                   varchar(10) NOT NULL,
        PRIMARY KEY (id_logs)
  );
  CREATEINDEX logs_idx ON logs_table(id_logs);
```

- PostgreSQL database creation script:

```
  -- PostgreSQL Database creation script
  CREATE DATABASE sympa;
  -- Connect to DB
  \connect sympa
  DROP TABLE user_table;
  CREATE TABLE user_table (
        email_user          varchar (100) NOT NULL,
        gecos_user          varchar (150),
        cookie_delay_user       int4,
          password_user               varchar (40),
          lang_user               varchar (10),
        attributes_user               varchar (255),
        CONSTRAINT ind_user PRIMARY KEY (email_user)
  );
  DROP TABLE subscriber_table;
  CREATE TABLE subscriber_table (
        list_subscriber               varchar (50) NOT NULL,
        user_subscriber               varchar (100) NOT NULL,
        robot_subscriber    varchar (80) NOT NULL,
        date_subscriber               timestamp with time zone NOT NULL,
        update_subscriber   timestamp with time zone,
        visibility_subscriber         varchar (20),
        reception_subscribervarchar (20),
        topics_subscriber   varchar (200),
        bounce_subscriber   varchar (35),
        bounce_score_subscriber int4,
        bounce_address_subscriber     varchar (100),
        comment_subscriber  varchar (150),
        subscribed_subscriber         smallint,
        included_subscriber smallint,
        include_sources_subscriber varchar(50),
        CONSTRAINT ind_subscriber PRIMARY KEY (list_subscriber, user_subscriber, robot_subscriber)
  );
```

```
CREATE INDEX subscriber_idx ON subscriber_table (user_subscriber,list_subscriber,robot_subscriber
DROP TABLE admin_table;
CREATE TABLE admin_table (
      list_admin                    varchar(50) NOT NULL,
      user_admin                    varchar(100) NOT NULL,
      robot_admin                   varchar(80) NOT NULL,
      role_admin                    varchar(15) NOT NULL,
      date_admin                    timestamp with time zone NOT NULL,
      update_admin                  timestamp with time zone,
      reception_admin     varchar(20),
      comment_admin                 varchar(150),
      subscribed_admin    smallint,
      included_admin      smallint,
      include_sources_admin         varchar(50),
      info_admin                    varchar(150),
      profile_admin                 varchar(15),
        CONSTRAINT ind_admin PRIMARY KEY (list_admin, user_admin, robot_admin, role_admin)
);
CREATEINDEX admin_idx ON admin_table(list_admin, user_admin,robot_admin, role_admin);
DROP TABLE netidmap_table;
CREATE TABLE netidmap_table (
        netid_netidmap              varchar (100) NOT NULL,
      serviceid_netidmap      varchar (100) NOT NULL,
      robot_netidmap                  varchar (80) NOT NULL,
        email_netidmap              varchar (100),
        CONSTRAINT ind_netidmap PRIMARY KEY (netid_netidmap, serviceid_netidmap, robot_netidmap)
);
CREATEINDEX netidmap_idx ON netidmap_table(netid_netidmap, serviceid_netidmap, robot_netidmap);
DROP TABLE logs_table;
CREATE TABLE logs_table (
      id_logs                       bigint NOT NULL,
      date_logs           int4 NOT NULL,
      robot_logs          varchar (80),
      list_logs           varchar (50),
      action_logs                   varchar (50) NOT NULL,
      parameters_logs               varchar (100),
      target_email_logs   varchar (100),
      user_email_logs               varchar (100),
      msg_id_logs                   varchar (255),
      status_logs                   varchar (10) NOT NULL,
      error_type_logs               varchar (150),
      client_logs                   varchar (100),
      daemon_logs                   varchar (10) NOT NULL,
        CONSTRAINT ind_logs PRIMARY KEY (id_logs)
);
CREATEINDEX logs_idx ON logs_table(id_logs);
```

- Sybase database creation script:

```
/* Sybase Database creation script 2.5.2 */
/* Thierry Charles <tcharles@electron-libre.com> */
/* 15/06/01 : extend password_user */
/* sympa database must have been created */
/* eg: create database sympa on your_device_data=10 log on your_device_log=4 */
use sympa
go
create table user_table
(
    email_user             varchar(100)           not null,
    gecos_user             varchar(150)           null    ,
    password_user          varchar(40)            null    ,
    cookie_delay_user      numeric                null    ,
    lang_user              varchar(10)            null    ,
    attributes_user          varchar(255)           null    ,
    constraint ind_user primary key (email_user)
)
go
create index email_user_fk on user_table (email_user)
go
create table subscriber_table
(
    list_subscriber        varchar(50)            not null,
    user_subscriber        varchar(100)           not null,
    robot_subscriber       varchar(80)           not null,
    date_subscriber        datetime               not null,
    update_subscriber      datetime               null,
    visibility_subscriber  varchar(20)            null    ,
    reception_subscriber   varchar(20)            null    ,
    topics_subscriber        varchar(200)           null,
    bounce_subscriber      varchar(35)            null    ,
    bounce_score_subscriber numeric               null    ,
    comment_subscriber     varchar(150)           null    ,
    subscribed_subscriber  numeric              null    ,
    included_subscriber    numeric                null    ,
    include_sources_subscriber varchar(50)        null    ,
    constraint ind_subscriber primary key (list_subscriber, user_subscriber, robot_subscriber)
)
go
create index list_subscriber_fk on subscriber_table (list_subscriber)
go
create index user_subscriber_fk on subscriber_table (user_subscriber)
go
create index robot_subscriber_fk on subscriber_table (robot_subscriber)
go
create table admin_table
(
      list_admin                    varchar(50)       not null,
      user_admin                    varchar(100)          not null,
      robot_admin                   varchar(80)           not null,
      role_admin                    varchar(15)       not null,
      date_admin                    datetime    not null,
      update_admin                  datetime          null,
      reception_admin     varchar(20)       null,
      comment_admin                 varchar(150)          null,
      subscribed_admin    numeric           null,
      included_admin      numeric           null,
      include_sources_admin         varchar(50)       null,
      info_admin                    varchar(150)      null,
      profile_admin                 varchar(15)       null,
       constraint ind_admin primary key (list_admin, user_admin,robot_admin,role_admin)
)
go
create index list_admin_fk on admin_table (list_admin)
go
create index user_admin_fk on admin_table (user_admin)
go
create index robot_admin_fk on admin_table (robot_admin)
go
create index role_admin_fk on admin_table (role_admin)
go
create table netidmap_table
(
        netid_netidmap             varchar (100) NOT NULL,
      serviceid_netidmap     varchar (100) NOT NULL,
      robot_netidmap                 varchar (80) NOT NULL,
        email_netidmap             varchar (100),
        constraint ind_netidmap primary key (netid_netidmap, serviceid_netidmap, robot_netidmap)
)
go
create index netid_netidmap_fk on netidmap_table (netid_netidmap)
go
create index serviceid_netidmap_fk on netidmap_table (serviceid_netidmap)
go
create index robot_netidmap_fk on netidmap_table (robot_netidmap)
go
CREATE TABLE logs_table (
      id_logs                       numeric NOT NULL,
      date_logs           numeric NOT NULL,
      robot_logs          varchar(80),
      list_logs           varchar(50),
      action_logs                   varchar(50) NOT NULL,
      parameters_logs               varchar(100),
      target_email_logs   varchar(100),
      user_email_logs               varchar(100),
      msg_id_logs                   varchar(255),
      status_logs                   varchar(10) NOT NULL,
      error_type_logs               varchar(150),
      client_logs                   varchar(100),
      daemon_logs                   varchar(10) NOT NULL,
      constraint ind_logs primary key (id_logs)
)
go
create index id_logs_fk on logs_table (id_logs)
go
```

- Oracle database creation script:

```
## Oracle Database creation script
## Fabien Marquois <fmarquoi@univ-lr.fr>
/Bases/oracle/product/7.3.4.1/bin/sqlplus loginsystem/passwdoracle <<-!
 create user SYMPA identified by SYMPA default tablespace TABLESP
temporary tablespace TEMP;
 grant create session to SYMPA;
 grant create table to SYMPA;
 grant create synonym to SYMPA;
 grant create view to SYMPA;
 grant execute any procedure to SYMPA;
 grant select any table to SYMPA;
 grant select any sequence to SYMPA;
 grant resource to SYMPA;
!
/Bases/oracle/product/7.3.4.1/bin/sqlplus SYMPA/SYMPA <<-!
CREATE TABLE user_table (
        email_user              varchar2(100) NOT NULL,
        gecos_user              varchar2(150),
        password_user           varchar2(40),
        cookie_delay_user       number,
        lang_user               varchar2(10),
      attributes_user               varchar2(500),
        CONSTRAINT ind_user PRIMARY KEY (email_user)
);
CREATE TABLE subscriber_table (
        list_subscriber         varchar2(50) NOT NULL,
        user_subscriber         varchar2(100) NOT NULL,
        robot_subscriber         varchar2(80) NOT NULL,
        date_subscriber         date NOT NULL,
      update_subscriber   date,
        visibility_subscriber   varchar2(20),
        reception_subscriber    varchar2(20),
      topics_subscriber   varchar2(200),
      bounce_subscriber       varchar2 (35),
      bounce_score_subscriber number,
      bounce_address_subscriber       varchar2 (100),
      comment_subscriber      varchar2 (150),
      subscribed_subscriber         number NULL    constraint  cons_subscribed_subscriber CHECK 
      included_subscriber number NULL    constraint  cons_included_subscriber CHECK (included_sub
      include_sources_subscriber varchar2(50),
        CONSTRAINT ind_subscriber PRIMARY KEY (list_subscriber,user_subscriber,robot_subscriber)
);
CREATE TABLE admin_table (
      list_admin                    varchar2(50) NOT NULL,
      user_admin                    varchar2(100) NOT NULL,
      robot_admin                   varchar2(80) NOT NULL,
      role_admin                    varchar2(20) NOT NULL,
      date_admin                    date NOT NULL,
      update_admin                  date,
      reception_admin     varchar2(20),
      comment_admin                 varchar2(150),
      subscribed_admin    number NULL    constraint  cons_subscribed_admin CHECK (subscribed_admi
      included_admin      number NULL    constraint  cons_included_admin CHECK (included_admin in
      include_sources_admin         varchar2(50),
      info_admin                    varchar2(150),
      profile_admin                 varchar2(20),
       CONSTRAINT ind_admin PRIMARY KEY (list_admin,user_admin,robot_admin,role_admin)
);
CREATE TABLE netidmap_table (
        netid_netidmap              varchar2 (100) NOT NULL,
      serviceid_netidmap      varchar2 (100) NOT NULL,
      robot_netidmap                  varchar2 (80) NOT NULL,
        email_netidmap              varchar2 (100),
        CONSTRAINT ind_netidmap PRIMARY KEY (netid_netidmap, serviceid_netidmap, robot_netidmap)
);
CREATE TABLE logs_table (
      id_logs                       number NOT NULL,
      date_logs           number NOT NULL,
      robot_logs          varchar2 (80),
      list_logs           varchar2 (50),
      action_logs                   varchar2 (50) NOT NULL,
      parameters_logs               varchar2 (100),
      target_email_logs   varchar2 (100),
      user_email_logs               varchar2 (100),
      msg_id_logs                   varchar2 (255),
      status_logs                   varchar2 (10) NOT NULL,
      error_type_logs               varchar2 (150),
      client_logs                   varchar2 (100),
      daemon_logs                   varchar2 (10) NOT NULL,
      CONSTRAINT ind_admin PRIMARY KEY (id_logs)
);
!
```

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

Example:

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

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 subscribes 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 then 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.

Management of the include cache

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

Sympa used to manage the cache of such *included* subscribers in a DB File (`include` mode), but now stores subscribers, editors and owners in the database (`include2` mode). These changes brought the following advantages:

- Sympa processes are smaller when dealing with big mailing lists (in include mode).
- Cache update is now performed regularly by a dedicated process, the task manager.
- Mixed lists (included + subscribed users) can now be created.
- Sympa can now provide delivery options for *included* members.
- Bounce information can be managed for *included* members.
- Sympa keeps track of the data sources of a member (available on the web REVIEW page).
- *Included* members can also subscribe to the list. It allows them to remain in the list in case they might not be included anymore.

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 configuration

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

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

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

All your lists are now configured to use the database, unless you set 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 performance you can configure *WWSympa* to use FastCGI; the CGI will be persistent in memory.
All data will be accessed through the CGI, including web archives. This is required to allow the authentication scheme to be applied systematically.

Authentication is based on passwords stored in the database table `user_table`; if the appropriate `Crypt::CipherSaber` is installed, 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;
2. evaluate user‘s permissions for the requested action;
3. process the requested action;
4. set up variables resulting from the action;
5. parse the HTML template files.

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:

- 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;
- 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
```

- Dedicated Apache server: run a dedicated Apache server with `sympa.sympa` as uid.gid (the Apache default is `apache.apache`);
- Apache suExec: use an Apache virtual host with `sympa.sympa` as uid.gid; Apache needs to be compiled with suexec. Be aware that the Apache suexec usually define a lowest UID/GID allowed to be a target user for suEXEC. For most systems, including binaries distribution of Apache, the default value `100` is common. So Sympa UID (and Sympa GID) must be higher than 100 or suexec must be tuned in order to allow lower UID/GID. Check http://httpd.apache.org/docs/suexec.html#install for details. The User and Group directive have to be set before the FastCgiServer directive is encountered;
- C wrapper: otherwise, you can overcome restrictions on the execution of suid scripts by using a short C program, owned by Sympa and with the suid bit set, to start `wwsympa.fcgi`. Here is an example (with no guarantee attached):

```
  #include <unistd.h>
  #define WWSYMPA ''/home/sympa/bin/wwsympa.fcgi''
  int main(int argn, char **argv, char **envp) {
      argv[0] = WWSYMPA;
      execve(WWSYMPA,argv,envp);
  }
```

Installing wwsympa.fcgi in your Apache server

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

```
   Example: <code>Alias /static-sympa /home/sympa/static_content</code>
```

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

```
   Example: <code>ScriptAlias /sympa /home/sympa/bin/wwsympa.fcgi</code>
```

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

Example:

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

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

If you are using `sudo` (see wwsympa.fcgi access permissions), then replace `wwsympa.fcgi` calls with `wwsympa_sudo_wrapper.pl`.

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

Using FastCGI

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

WWSympa can also work without FastCGI, depending on the `use_fast_cgi` parameter (see use_fast_cgi - 1).

To run *WWSympa* with FastCGI, you need to install:

- `mod_fastcgi`: the Apache module that provides FastCGI features;
- `FCGI`: the Perl module used by *WWSympa*.

wwsympa.conf parameters

arc_path

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

archive_default_index thrd - mail

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

archived_pidfile

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

bounce_path

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

bounced_pidfile

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

cookie_expire

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

cookie_domain

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

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

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

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

default_home

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

icons_url

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

log_facility

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

mhonarc

(Default value: `/usr/bin/mhonarc`)
Path to the (superb) MhOnArc program. Required for the HTML archive.

htmlarea_url

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

password_case sensitive | insensitive

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

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

title

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

use_fast_cgi 0|1

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

MhOnArc

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

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

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

- `/home/sympa/expl/mylist/mhonarc-ressources` in order to create a specific archive look for a particular list;
- or `/home/sympa/etc/mhonarc-ressources`.

Archiving daemon

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

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

   ```
   web_archive
   access public|private|owner|listmaster|closed
   quota 10000
   ```

   If `web_archive` is defined for a list, every message distributed by this list is copied to `/home/sympa/spool/outgoing/` (there is no need to create nonexistent subscribers to receive copies of messages). In this example, disk quota (expressed in Kbytes) for the archive is limited to 10 Mb;
3. start `archived.pl`. Sympa and Apache;
4. check *WWSympa* logs, or alternatively, start `archived.pl` in debug mode (-d).
5. If you change MhOnArc resources and wish to rebuild the entire archive using the new look defined for MhOnArc, simply create an empty file named `.rebuild.mylist@myhost` in `/home/sympa/spool/outgoing`, and make sure that the owner of this file is Sympa.
   Example:

   ```
   su sympa -c "touch /home/sympa/spool/outgoing/.rebuild.sympa-fr@cru.fr"
   ```

   You can also rebuild the web archive from the admin page of the list.
   Furthermore, if you want to get the list archive, you can do it via the `List-admin menu→ Archive Management`

Database configuration

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

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

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

Logging in as listmaster

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

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

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

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

Sympa Internationalization

Catalogs and templates

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

Documentation and resources about software translations : http://translate.sourceforge.net/doc

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

Examples:

```
<code>[%|loc%]User Email[% END %]</code>
```

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

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

Translating Sympa GUI in your language

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

Defining language-specific templates

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

Example :

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

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

Web templates can also make use of the `locale` variable to make templates multi-lingual:

Example :

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

Translating topics titles

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

Handling of encodings

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

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

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

Sympa RSS channel

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

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

Example: `https://my.server/wws/rss`

The access control of RSS queries proceed on the same way as *WWSympa* actions referred to. Sympa provides the following RSS features:

- the latest created lists on a robot (`latest_lists`);
- the most active lists on a robot(`active_lists`);
- the latest messages of a list (`active_arc`);
- the latest shared documents of a list (`latest_d_read`).

latest_lists

This provides the latest created lists.

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

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

Parameters:

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

active_lists

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

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

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

Parameters:

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

latest_arc

This provides the latest messages of a list.

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

Parameters:

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

latest_d_read

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

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

Parameters:

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

Sympa SOAP server

Introduction

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

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

- The client sends credentials and then requests a service providing a cookie with id `sympa-user`.
- The client authenticates the end user providing the `sympa-user` HTTP cookie. This can be used in order to share an authenticated session between Sympa and other applications running on the same server as *WWSympa*. The SOAP method used is `getUserEmailByCookieRequest`.

- The client provides user email and password and requests a service in a single SOAP access using the `authenticateAndRun` SOAP service.
- The client is trusted by Sympa as a proxy application and is authorized to set some variables that will be used by Sympa during the authorization scenario evaluation. Trusted applications have own password there, and the variables they can set are listed in a configuration file named `trusted_applications.conf`. See Trust remote applications.

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

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

Web server setup

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

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

```
FastCgiServer /home/sympa/bin/sympa_soap_server.fcgi -processes 1
ScriptAlias /sympasoap /home/sympa/bin/sympa_soap_server.fcgi

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

Sympa setup

The only mandatory parameter you need to set in the `sympa.conf`/`robot.conf` files is the `soap_url`, that defines the URL of the SOAP service corresponding to the ScriptAlias you have previously set up in the Apache configuration.

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

Trust remote applications

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

This configuration file `trusted_applications.conf` can be created in the robot `etc/` subdirectory or in the `/home/sympa/etc` directory depending on the scope you want for it (the source package include a sample of file `trusted_applications.conf` in the `soap` directory). This file is made of paragraphs separated by empty lines and stating with keyword `trusted_application`. A sample `trusted_applications.conf` file is provided with Sympa sources. Each paragraph defines a remote trusted application with keyword/value pairs:

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

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

```
/home/sympa/bin/sympa_soap_client.pl --soap_url=http://my.server/sympasoap --service=createList --t

/home/sympa/bin/sympa_soap_client.pl --soap_url=http://myserver/sympasoap --service=add --trusted_a
```

Available services are:

- info *list*;
- which;
- lists;
- review *list*;
- amI *function*;
- subscribe *list*;
- signoff *list*;
- add *list email*;
- del *list email*;
- createList *list*;
- closeList *list*;
- login *email password*;
- casLogin *proxyTicket*;
- checkCookie.

The WSDL service description

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

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

<!-- types part -->

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

        <complexType name=''ArrayOfLists''>
                <complexContent>
                        <restriction base=''SOAP-ENC:Array''>
```

```
                                        <attribute ref=''SOAP-ENC:arrayType'' wsdl:arrayType=''tns:li
                              </restriction>
                    </complexContent>
          </complexType>

          <complexType name=''ArrayOfString''>
                    <complexContent>
                              <restriction base=''SOAP-ENC:Array''>
                                        <attribute ref=''SOAP-ENC:arrayType'' wsdl:arrayType=''strinc
                              </restriction>
                    </complexContent>
          </complexType>

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

  <!-- message part -->

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

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

  <message name=''complexWhichRequest''>
  </message>

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

  <message name=''whichRequest''>
  </message>

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

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

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

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

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

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

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

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

```
</message>

<message name=''addRequest''>
        <part name=''list'' type=''xsd:string''/>
        <part name=''email'' type=''xsd:string''/>
        <part name=''gecos'' type=''xsd:string''  xsd:minOccurs=''0''/>
        <part name=''quiet'' type=''xsd:boolean''  xsd:minOccurs=''0''/>
</message>

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

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

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

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

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

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

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

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

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

<message name=''loginResponse''>
        <part name=''return'' type=''xsd:string''/>
</message>

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

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

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

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

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

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

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

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

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

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

<message name=''complexListsRequest''>
</message>

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

<message name=''checkCookieRequest''>
</message>

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

<!-- portType part -->

<portType name=''SympaPort''>
        <operation name=''info''>
                  <input message=''tns:infoRequest'' />
                  <output message=''tns:infoResponse'' />
        </operation>
        <operation name=''complexWhich''>
                  <input message=''tns:complexWhichRequest'' />
                  <output message=''tns:complexWhichResponse'' />
        </operation>
        <operation name=''which''>
                  <input message=''tns:whichRequest'' />
                  <output message=''tns:whichResponse'' />
        </operation>
        <operation name=''amI''>
                  <input message=''tns:amIRequest'' />
                  <output message=''tns:amIResponse'' />
        </operation>
        <operation name=''add''>
                  <input message=''tns:addRequest'' />
                  <output message=''tns:addResponse'' />
        </operation>
        <operation name=''del''>
                  <input message=''tns:delRequest'' />
                  <output message=''tns:delResponse'' />
        </operation>
        <operation name=''createList''>
                  <input message=''tns:createListRequest'' />
                  <output message=''tns:createListResponse'' />
        </operation>
        <operation name=''closeList''>
                  <input message=''tns:closeListRequest'' />
                  <output message=''tns:closeListResponse'' />
        </operation>
        <operation name=''review''>
                  <input message=''tns:reviewRequest'' />
                  <output message=''tns:reviewResponse'' />
        </operation>
        <operation name=''subscribe''>
                  <input message=''tns:subscribeRequest'' />
                  <output message=''tns:subscribeResponse'' />
        </operation>
        <operation name=''signoff''>
                <input message=''tns:signoffRequest'' />
                <output message=''tns:signoffResponse'' />
        </operation>
        <operation name=''login''>
                <input message=''tns:loginRequest'' />
                <output message=''tns:loginResponse'' />
        </operation>
        <operation name=''casLogin''>
                <input message=''tns:casLoginRequest'' />
                <output message=''tns:casLoginResponse'' />
        </operation>
        <operation name=''getUserEmailByCookie''>
                <input message=''tns:getUserEmailByCookieRequest'' />
                <output message=''tns:getUserEmailByCookieResponse'' />
        </operation>
        <operation name=''authenticateAndRun''>
                <input message=''tns:authenticateAndRunRequest'' />
                <output message=''tns:authenticateAndRunResponse'' />
        </operation>
        <operation name=''authenticateRemoteAppAndRun''>
                <input message=''tns:authenticateRemoteAppAndRunRequest'' />
                <output message=''tns:authenticateRemoteAppAndRunResponse'' />
        </operation>
        <operation name=''lists''>
                <input message=''tns:listsRequest'' />
                <output message=''tns:listsResponse'' />
        </operation>
        <operation name=''complexLists''>
                <input message=''tns:complexListsRequest'' />
                <output message=''tns:complexListsResponse'' />
        </operation>
        <operation name=''checkCookie''>
                <input message=''tns:checkCookieRequest'' />
                <output message=''tns:checkCookieResponse'' />
        </operation>
</portType>

<!-- Binding part -->

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

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

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

```
                        <input>
                                <soap:body use=''encoded''
                                        namespace=''urn:sympasoap''
                                        encodingStyle=''http://schemas.xmlsoap.org/soap/enc
                        </input>
                        <output>
                                <soap:body use=''encoded''
                                        namespace=''urn:sympasoap''
                                        encodingStyle=''http://schemas.xmlsoap.org/soap/enc
                        </output>
        </operation>
</binding>

<!-- service part -->

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

</definitions>
```

Client-side programming

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

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

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

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

Writing a Java client with Axis

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

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

- axis.jar;
- saaj.jar;
- commons-discovery.jar;
- commons-logging.jar;
- xercesImpl.jar;
- jaxrpc.jar;
- xml-apis.jar;
- jaas.jar;

- wsdl4j.jar;
- soap.jar.

Next, you have to generate client Java class files from the sympa WSDL URL. Use the following command:

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

For example:

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

Exemple of screen output during generation of Java files:

```

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

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

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

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

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

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

```

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

Authentication

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

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

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

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

S/MIME and HTTPS authentication

Chapter Use of S/MIME signature by Sympa itself deals with Sympa and S/MIME signature. Sympa uses the `OpenSSL` library to work on S/MIME messages, you need to configure some related Sympa parameters: S/X509 Sympa configuration.

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

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

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

```
SSLOptions +ExportCertData
```

Authentication with email address, uid or alternate email address

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

This way users can indifferently authenticate with their `ldap_uid`, their `alternate_email` or their canonic email stored in the LDAP directory.

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

Example: a person is described by

```
                 Dn:cn=Fabrice Rafart,
                 ou=Siege ,
                 o=MyCompany,
                 c=FR Objectclass:
                 person Cn: Fabrice Rafart
                 Title: Network Responsible
                 O: Siege
                 Or: Data processing
                 Telephonenumber: 01-00-00-00-00
                 Facsimiletelephonenumber:01-00-00-00-00
                 L:Paris
                 Country: France
                           uid: frafart
                           mail: Fabrice.Rafart@MyCompany.fr
                 alternate_email: frafart@MyCompany.fr
                 alternate:rafart@MyCompany.fr
```

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

Generic SSO authentication

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

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

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

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

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

CAS-based authentication

CAS is the Yale University SSO software. Sympa can use the CAS authentication service.

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

auth.conf

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

The `/home/sympa/etc/auth.conf` is organized in paragraphs. Each paragraph describes an authentication service with all parameters required to perform an authentication using this service. Sympa's current version can perform authentication through LDAP directories, using an external Single Sign-On Service (like CAS or Shibboleth), or using the internal `user_table` table.

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

The second form in the login page contains the list of CAS servers so that users can choose explicitely their CAS service.

Each paragraph starts with one of the keyword `cas`, `ldap` or `user_table`.

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

```

paragraphs
keyword value

paragraphs
keyword value
```

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

Empty lines are also considered as comments and are ignored at the beginning. After the first paragraph, they are considered as paragraph separators. There should only be one directive per line, but their order in the paragraph is of no importance.

Example:

```

  #Configuration file auth.conf for the LDAP authentification
```

```
#Description of parameters for each directory

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

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

## The email address is not provided by the user home institution
generic_sso
        service_name               Shibboleth Federation
        service_id                 myfederation
        http_header_prefix         HTTP_SHIB
        netid_http_header          HTTP_SHIB_EMAIL_ADDRESS
        internal_email_by_netid    1
        force_email_verify         1

ldap
        regexp                                  univ-rennes1\.fr
        host                                    ldap.univ-rennes1.fr:389
        timeout                                 30
        suffix                                  dc=univ-rennes1,dc=fr
        get_dn_by_uid_filter          (uid=[sender])
        get_dn_by_email_filter                  (|(mail=[sender])(mailalternateaddress=[sender]))
        email_attribute                         mail
        alternative_email_attribute   mailalternateaddress,ur1mail
        scope                                   sub
        use_ssl                         1
        ssl_version                     sslv3
        ssl_ciphers                     MEDIUM:HIGH

ldap
        host                                    ldap.univ-nancy2.fr:392,ldap1.univ-nancy2.fr:392,lc
        timeout                                 20
        bind_dn                         cn=sympa,ou=people,dc=cru,dc=fr
        bind_password                   sympaPASSWD
        suffix                                  dc=univ-nancy2,dc=fr
        get_dn_by_uid_filter          (uid=[sender])
        get_dn_by_email_filter                            (|(mail=[sender])(n2atraliasmail=[sender]
        alternative_email_attribute   n2atrmaildrop
        email_attribute                         mail
        scope                                   sub
        authentication_info_url         http://sso.univ-nancy2.fr/

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

user_table paragraph

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

ldap paragraph

- `regexp` and `negative_regexp`

  Same as in the `user_table` paragraph: if an email address is provided (this does not apply to an uid), then the regular expression will be applied to find out if the LDAP directory can be used to authenticate a subset of users.

- `host`
  This keyword is **mandatory**. It is the domain name used in order to bind to the directory and then to extract information. You must mention the port number after the server name. Server replication is supported by listing several servers separated by commas.
  Example:

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

- `timeout`
  It corresponds to the timelimit in the Search fonction. A timelimit that restricts the maximum time (in seconds) allowed for a search. A value of 0 (the default) means that no timelimit will be requested.
- `suffix`
  The root of the DIT (Directory Information Tree). The DN that is the base object entry relative to which the search is to be performed.
  Example: `dc=university,dc=fr`
- `bind_dn`
  If anonymous bind is not allowed on the LDAP server, a DN and password can be used.
- `bind_password`
  This password is used, combined with the `bind_dn` above.
- `get_dn_by_uid_filter`
  Defines the search filter corresponding to the `ldap_uid`. (RFC 2254 compliant). If you want to apply the filter on the user, use the variable ' [sender] ‘. It will work with every type of authentication (uid, `alternate_email`, ...).
  Example:

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

- `get_dn_by_email_filter`
  Defines the search filter corresponding to the email addresses (canonic and alternative - this is RFC 2254 compliant). If you want to apply the filter on the user, use the variable ' [sender] ‘. It will work with every type of authentication (`uid`, `alternate_email`..).
  Example: a person is described by

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

```
alternate:rafart@MaSociete.fr
```

The filters can be:

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

- `email_attribute`
  The name of the attribute for the canonic email in your directory: for instance `mail`, `canonic_email`, `canonic_address`, ... In the previous example, the canonic email is `mail`.
- `alternative_email_attribute`
  The name of the attribute for the alternate email in your directory: for instance `alternate_email`, `mailalternateaddress`, ... You make a list of these attributes separated by commas.

With this list, Sympa creates a cookie which contains various information: whether the user is authenticated via LDAP or not, his alternate email. Storing the alternate email is interesting when you want to canonify your preferences and subscriptions, that is to say you want to use a unique address in `user_table` and `subscriber_table`, which is the canonic email.

- `scope` (Default value: `sub`)
  By default, the search is performed on the whole tree below the specified base object. This may be changed by specifying a scope:
  - `base`: search only the base object,
  - `one`: search the entries immediately below the base object,
  - `sub`: search the whole tree below the base object. This is the default.
- `authentication_info_url`
  Defines the URL of a document describing LDAP password management. When hitting Sympa's *Send me a password* button, LDAP users will be redirected to this URL.
- `use_ssl`
  If set to `1`, connection to the LDAP server will use SSL (LDAPS).
- `ssl_version`
  This defines the version of the SSL/TLS protocol to use. Defaults of Net::LDAPS to `sslv2/3`, other possible values are `sslv2`, `sslv3`, and `tlsv1`.
- ssl_ciphers

Specify which subset of cipher suites are permissible for this connection, using the standard OpenSSL string format. The default value of Net::LDAPS for ciphers is `ALL`, which permits all ciphers, even those that do not encrypt!

generic_sso paragraph

- `service_name`
  This is the SSO service name that will be offered to the user in the login banner menu.
- `service_id`
  This service ID is used as a parameter by Sympa to refer to the SSO service (instead of the service name).

A corresponding URL on the local web server should be protected by the SSO system; this URL would look like `http://yourhost.yourdomain/sympa/sso_login/inqueue` if the `service_id` is `inqueue`.

- `http_header_prefix`
  Sympa gets user attributes from environment variables coming from the web server. These variables are then stored in the `user_table` DB table for later use in authorization scenarios (in structure). Only environment variables starting with the defined prefix will be kept.
- `email_http_header`
  This parameter defines the environment variable that will contain the authenticated user's email address.

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

- `internal_email_by_netid`
  If set to `1`, this parameter makes Sympa use its `netidmap` table to associate NetIDs to user email addresses.
- `netid_http_header`
  This parameter defines the environment variable that will contain the user's identifier. This netid will then be associated with an email address provided by the user.
- `force_email_verify`
  If set to `1`, this parameter makes Sympa check the user's email address. If the email address was not provided by the authentication module, then the user is requested to provide a valid email address.

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

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

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

- `ldap_email_attribute`
  The attribute name to be used as user canonical email. In the current version of Sympa, only the first value returned by the LDAP server is used.

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

cas paragraph

- `auth_service_name`
  The friendly user service name as shown by Sympa in the login page.
- `host` (OBSOLETE)
  This parameter has been replaced by **base_url** parameter
- `base_url`
  The base URL of the CAS server.
- `non_blocking_redirection`
  This parameter only concerns the first access to Sympa services by a user, it activates or not the non blocking redirection to the related CAS server to check automatically if the user as been previously authenticated with this CAS server. Possible values are `on` and `off`, default is `on`. The redirection to CAS is used with the CGI parameter `gateway=1` that specifies to CAS server to always redirect the user to the original URL, but just check if the user is logged. If active, the SSO service is effective and transparent, but in case the CAS server is out of order, the access to Sympa services is impossible.
- `login_uri` (OBSOLETE)
  This parameter has been replaced by the `login_path` parameter.
- `login_path` (OPTIONAL)
  The login service path.
- `check_uri` (OBSOLETE)
  This parameter has been replaced by the `service_validate_path` parameter.
- `service_validate_path` (OPTIONAL)
  The ticket validation service path.
- `logout_uri` (OBSOLETE)
  This parameter has been replaced by the `logout_path` parameter.
- `logout_path` (OPTIONAL)
  The logout service path.
- `proxy_path` (OPTIONAL)
  The proxy service path, only used by the Sympa SOAP server.
- `proxy_validate_path` (OPTIONAL)
  The proxy validate service path, only used by the Sympa SOAP server.
- `ldap_host`
  The LDAP host Sympa will connect to fetch user email when user uid is return by CAS service. The `ldap_host` includes the port number and it may be a comma separated list of redondant hosts.

- `ldap_bind_dn`
  The DN used to bind to this server. Anonymous bind is used if this parameter is not defined.
- `ldap_bind_password`
  The password used unless anonymous bind is used.
- `ldap_suffix`
  The LDAP suffix used when searching user email.
- `ldap_scope`
  The scope used when searching user email. Possible values are `sub`, `base` and `one`.
- `ldap_get_email_by_uid_filter`
  The filter used to perform the email search.
- `ldap_email_attribute`
  The attribute name to be used as user canonical email. In the current version of Sympa, only the first value returned by the LDAP server is used.
- `ldap_timeout`
  The time out for the search.
- `ldap_use_ssl`
  If set to `1`, connection to the LDAP server will use SSL (LDAPS).
- `ldap_ssl_version`
  This defines the version of the SSL/TLS protocol to use. Defaults of Net::LDAPS to `sslv2/3`, other possible values are `sslv2`, `sslv3`, and `tlsv1`.
- `ldap_ssl_ciphers`
  Specifies which subset of cipher suites are permissible for this connection, using the OpenSSL string format. The default value of Net::LDAPS for ciphers is `ALL`, which permits all ciphers, even those that do not encrypt!

Sharing WWSympa's authentication with other applications

If you are not using a web Single Sign On system, you might want to make other web applications collaborate with Sympa and share the same authentication system. Sympa uses HTTP cookies to carry users' authentication information from page to page. This cookie contains no information about privileges. To make your application work with Sympa, you have two possibilities:

- Delegating authentication operations to *WWSympa*
  If you want to avoid spending a lot of time programming a CGI to do Login, Logout and Remindpassword, you can copy *WWSympa*'s login page to your application, and then make use of the cookie information within your application. The cookie format is:

  ```
  sympauser=<user_email>:<checksum>
  ```

  where `<user_email>` is the user's complete e-mail address, and `<checksum>` represents the 8 last bytes of the MD5 checksum of the `<user_email>`+Sympa `cookie` configuration parameter. Your application needs to know what the `cookie` parameter is, so it can check the HTTP cookie validity; this is a secret shared between *WWSympa* and your application. *WWSympa*'s `loginrequest` page can be called to return to the referer URL when an action is performed. Here is a sample HTML anchor:

```
<A HREF=''/sympa/loginrequest/referer''>Login page</A>
```

You can also have your own HTML page submitting data to `wwsympa.fcgi` CGI. If you do so, you can set the `referer` variable to another URI. You can also set the `failure_referer` to make *WWSympa* redirect the client to a different URI if login fails.

- Using *WWSympa*'s HTTP cookie format within your authentication module
  To cooperate with *WWSympa*, you simply need to adopt its HTTP cookie format and share the secret it uses to generate MD5 checksums, i.e. the `cookie` configuration parameter. In this way, *WWSympa* will accept users authenticated through your application without further authentication.

Provide a Sympa login form in another application

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

```

<FORM ACTION=''http://listes.cru.fr/sympa'' method=''post''>
      <input type=''hidden'' name=''previous_action'' value=''arc'' />
      Access web archives of list
      <select name=''previous_list''>
      <option value=''sympa-users'' >sympa-users</option>
      </select><br/>
      <input type=''hidden'' name=''action'' value=''login'' />
      <label for=''email''>email address:
      <input type=''text'' name=''email'' id=''email'' size=''18'' value='''' /></label><br />
      <label for=''passwd'' >password:
      <input type=''password'' name=''passwd'' id=''passwd'' size=''8'' /></label> <br/>
      <input class=''MainMenuLinks'' type=''submit'' name=''action_login'' value="Login and access
</FORM>
```

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

Authorization scenarios

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

Location of scenario file

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

When customizing scenario for your own site, robot or list, don‘t modify .../sympa/bin/scenari content or next Sympa update will overwrite it (you must never modify anything in .../sympa/bin/ unless your are patching Sympa). You can modify Sympa behavior if you create new scenario which name is the same as one of the scenario included in the distribution but with a location related to target site, robot or list. You can also add a new scenario ; it will automatically add an accepted value for the related parameter.

When modifying a existing scenario you need to restart Sympa or touch list config file before Sympa use it.

Scenario structure

Each authorization scenario rule contains:

- a condition: the condition is evaluated by Sympa. It can use variables such as `sender` for the sender‘s email, `list` for the list name, etc.
- an authentication method. The authentication method can be `smtp`, `md5` or `smime`. The rule is applied by Sympa if both the condition and authentication method match the runtime context. `smtp` is used if Sympa uses the SMTP `From:` header , `md5` is used if a unique MD5 key as been returned by the requestor to validate the message, and `smime` is used for signed messages (see configuration to recognize S/MIME signatures);
- a returned atomic action that will be executed by Sympa if the rule matches.

Example:

```
del.auth

title.us deletion performed only by list owners, need authentication
title.fr suppression r\'eserv\'ee au propri\'etaire avec authentification
title.es eliminacin reservada slo para el propietario, necesita autentificacin

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

Rules specifications

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

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

<condition> ::= [!] <condition
                | true ()
                | all ()
                | equal (<var>, <var>)
                | match (<var>, /perl_regexp/)
                  | search (<named_filter_file>)
                | is_subscriber (<listname>, <var>)
                | is_owner (<listname>, <var>)
                | is_editor (<listname>, <var>)
                | is_listmaster (<var>)
                | older (<date>, <date>)    # true if first date is anterior to the second date
                | newer (<date>, <date>)    # true if first date is posterior to the second date
                | CustomCondition::<package_name> (<var>*)

<var> ::= [email] | [sender] | [user-><user_key_word>] | [previous_email]
                  | [remote_host] | [remote_addr] | [user_attributes-><user_attributes_keyword>]
```

```
                       | [subscriber-><subscriber_key_word>] | [list-><list_key_word>] | [env-><env_va
                       | [conf-><conf_key_word>] | [msg_header-><smtp_key_word>] | [msg_body]
                       | [msg_part->type] | [msg_part->body] | [msg_encrypted] | [is_bcc] | [current_c
                       | [topic-auto] | [topic-sender,] | [topic-editor] | [topic] | [topic-needed]
                       | <string>

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

[sender] ::= email address of the current user (used on web or mail interface). Default value is 'n

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

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

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

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

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

[topic]  ::= topic of the message

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

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

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

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

<epoch_date> ::= <integer>

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

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

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

<auth> ::= smtp|md5|smime

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

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

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

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

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

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

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

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

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

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

(Refer to Tasks for date format definition)

The function to evaluate scenario is described in section internals.

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

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

A bunch of authorization scenarios is provided with the Sympa distribution; they provide a large set of configuration that allow to create lists for most usages. But you will probably create authorization scenarios for your own need. In this case, do not forget to restart Sympa and *WWSympa*, because authorization scenarios are not reloaded dynamically.

These standard authorization scenarios are located in the /home/sympa/bin/etc/scenari/ directory. Default scenarios are named <command>.default.

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

Example: copy the previous scenario to scenari/subscribe.rennes1:

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

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

```
subscribe rennes1
```

Named Filters

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

As a consequence, you can grant privileges in a list to people belonging to an LDAP directory, an SQL database or a flat text file, thanks to an authorization scenario.

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

LDAP Named Filters Definition

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

You must give a few information in order to create a LDAP Named Filter:

- `host`
  A list of host:port LDAP directories (replicates) entries.
- `suffix`
  Defines the naming space covered by the search (optional, depending on the LDAP server).
- `filter`
  Defines the LDAP search filter (RFC 2254 compliant). But you must absolutely take into account the first part of the filter which is: `(mail_attribute = [sender])`, as shown in the example. You will have to replace `mail_attribute` by the name of the attribute for the email. Sympa checks whether the user belongs to the category of people defined in the filter.
- `scope`
  By default, the search is performed on the whole tree below the specified base object. This may be changed by specifying a scope:
  - `base`: search only the base object.
  - `one`: search the entries immediately below the base object.
  - `sub`: search the whole tree below the base object. This is the default option.
- `bind_dn`
  If anonymous bind is not allowed on the LDAP server, a DN and password can be used.
- `bind_password`
  This password is used, combined with the `bind_dn` above.

`example.ldap`: we want to select the teachers of mathematics in the University of Rennes 1 in France:

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

SQL Named Filters Definition

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

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

Please refer to section Database related for a detailed explanation of each parameter.

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

- `db_type`
  Format: `db_type mysql|SQLite|Pg|Oracle|Sybase`; Database management system used. Mandatory and case sensitive.
- `db_host`
  Database host name. Mandatory.
- `db_name`
  Name of database to query. Mandatory.
- `statement`
  Mandatory. The SQL statement to execute to verify authorization. This statement must returns 0 to refuse the action, or anything else to grant privileges. The `SELECT COUNT (*)...` statement is the perfect query for this parameter. The keyword in the SQL query will be replaced by the sender's email.
- Optional parameters
  Please refer to main `sympa.conf` section for description.
  - `db_user`
  - `db_password`
  - `db_options`
  - `db_env`
  - `db_port`
  - `db_timeout`

`example.sql`: we want to select the teachers of mathematics in the University of Rennes 1 in France:

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

Search condition

The search condition is used in authorization scenarios.

The syntax of this rule is:

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

The variable used by `search` is the name of the LDAP configuration file or a txt matching enumeration.

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

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

The method of authentication does not change.

Scenario inclusion

Scenarios can also contain includes:

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

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

Scenario implicit inclusion

You can define a set of common scenario rules, used by all lists. `include.<action>.header` is automatically added to evaluated scenarios. Note that you will need to restart Sympa processes to force reloading of list config files.

Blacklist implicit rule

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

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

The goal is to block messages or other service requests from unwanted users. The blacklist can be defined for the robot or for the list. At the list level, the blacklist is to be managed by list owner or list editor via the web interface.

Custom Perl package conditions

You can use a Perl package of your own to evaluate a custom condition. It can be useful if you have very complex tasks to carry out to evaluate your condition (web services queries...). In this case, you should write a Perl module, place it in the `CustomCondition` namespace, with one verify fonction that has to return `1` to grant access, `undef` to warn of an error, or anything else to refuse the authorization.

This Perl module:

- must be placed in a subdirectoy `custom_conditions` of the `etc` directory of your Sympa installation, or of a robot;
- its filename must be lowercase;

- must be placed in the CustomCondition namespace;
- must contain one verify static fonction;
- will receive all condition arguments as parameters.

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

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

       #!/usr/bin/perl

       package CustomCondition::yes;

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

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

We can use this custom condition that way:

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

Note that the ,, are optional, but it is the way you can pass information to your package. Our yes.pm will print the values in the logs.

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

Hidding scenario files

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

Example:

```
/home/sympa/etc/my.domain.org/scenari/send.intranetorprivate:ignore
```

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

Virtual host

Sympa is designed to manage multiple distinct mailing list servers on a single host with a single

Sympa installation. Sympa virtual hosts are like Apache virtual hosting. Sympa virtual host definition includes a specific email address for the robot itself and its lists and also a virtual HTTP server. Each robot provides access to a set of lists, each list is related to only one robot.

Most configuration parameters can be redefined for each robot, except for general Sympa installation parameters (binary and spool location, SMTP engine, antivirus plugin, ...).

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

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

How to create a virtual host

You do not need to install several Sympa servers. A single `sympa.pl` daemon and one or more fastcgi servers can serve all virtual host. Just configure the server environment in order to accept the new domain definition.

- **The DNS** must be configured to define a new mail exchanger record (MX) to route message to your server. A new host (A record) or alias (CNAME) are mandatory to define the new web server.
- Configure your **MTA** (sendmail, postfix, exim, ...) to accept incoming messages for the new robot domain. Add mail aliases for the robot. Examples (with sendmail):

```
sympa@your.virtual.domain:      "| /home/sympa/bin/queue sympa@your.virtual.domain"
listmaster@your.virtual.domain: "| /home/sympa/bin/queue listmaster@your.virtual.domain"
bounce+*@your.virtual.domain:          "| /home/sympa/bin/bouncequeue sympa@your.virtual.domain"
```

- Define a **virtual host in your HTTPD server**. The fastcgi servers defined in the common section of you HTTPD server can be used by each virtual host. You do not need to run dedicated fascgi server for each virtual host. Examples:

```
FastCgiServer /home/sympa/bin/wwsympa.fcgi -processes 3 -idle-timeout 120
.....
<VirtualHost 195.215.92.16>
  ServerAdmin webmaster@your.virtual.domain
  DocumentRoot /var/www/your.virtual.domain
  ServerName your.virtual.domain
  <Location /sympa>
     SetHandler fastcgi-script
  </Location>
  ScriptAlias /sympa /home/sympa/bin/wwsympa.fcgi
  Alias /static-sympa /home/sympa/your.virtual.domain/static_content
</VirtualHost>
```

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

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

robot.conf

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

- `http_host`
  This hostname will be compared with the `SERVER_NAME` environment variable in `wwsympa.fcgi` to determine the current Virtual Host. You can add a path at the end of this parameter if you are running multiple virtual hosts on the same host. Examples:

```

        http_host  myhost.mydom
        http_host  myhost.mydom/sympa
```

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

These settings overwrite the equivalent global parameter defined in `/etc/sympa.conf` for the `my.domain.org` robot; the main `listmaster` still has privileges on Virtual Robots though. The `http_host` parameter is compared by *WWSympa* with the `SERVER_NAME` environment variable to recognize which robot is in use.

Robot customization

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

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

The `/home/sympa/etc/my.domain.org/web_tt2/`, `/home/sympa/etc/my.domain.` and `/home/sympa/etc/my.domain.org/scenari/` directories are searched when loading templates or scenarios before searching into `/home/sympa/etc` and `/home/sympa/bin/etc`. This allows to define different privileges and a different GUI for a Virtual Host.

Managing multiple virtual hosts

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

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

Interaction between Sympa and other applications

Soap

See Sympa SOAP server.

RSS channel

See Sympa's RSS channel.

Sharing WWSympa's authentication with other applications

See Sharing WWSympa's authentication with other applications.

Sharing data with other applications

You may extract subscribers, owners and editors for a list from any of:

- a text file;
- a relational database;
- a LDAP directory.

See the user_data_source list parameter.

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

See Data inclusion file.

Subscriber count

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

Example:

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

Customizing Sympa/WWSympa

Template file format

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

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

Here are some aspects regarding templates that are specific to Sympa:

- References to PO catalogues are noted with the **[% loc %]** tag that may include parameters. Example:

  ```
  [%|loc(list.name,list.host)%]Welcome to list %1 %2[%END%]
  ```

- Few exceptions apart, templates cannot insert or parse a file given its full or relative path, for security reasons. Only the file name should be provided; the TT2 parser will then use the `INCLUDE_PATH` provided by Sympa to find the relevant file to insert/parse.

- The **qencode** filter should be used if a template includes SMTP header fields that should be Q-encoded. Example:

```
[% FILTER qencode %]Message à modérer[%END%]
```

- You can write different versions of a template file in different languages, each of them being located in a subdirectory of the `tt2` directory. Example: `/mail_tt2/fr_FR/helpfile.tt2`.

Site template files

These files are used by Sympa as service messages for the `HELP`, `LISTS` and "REMIND *" commands. These files are interpreted (parsed) by Sympa and respect the TT2 template format; every file has a **.tt2** extension. See Template file format.

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

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

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

The following variables may be used in these template files:

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

helpfile.tt2

This file is sent in response to a `HELP` command. You may use additional variables:

- [% is_owner %]: TRUE if the user is list owner;
- [% is_editor %]: TRUE if the user is list editor.

lists.tt2

File returned by the LISTS command. An additional variable is available:

- [% lists %]: this is a hash table indexed by list names and containing lists' subjects. Only lists visible to the user (according to the visibility list parameter) are listed.

Example:

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

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

[% END %]
```

global_remind.tt2

This file is sent in response to a REMIND * command. (see Owner commands) You may use additional variables:

-[% lists %]: this is an array containing the names of the lists the user is subscribed to.

Example:

```
This is a subscription reminder.

You are subscribed to the following lists:
[% FOREACH l = lists %]

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

[% END %]

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

your_infected_msg.tt2

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

Web template files

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

Your list web template files should be placed in the /home/sympa/expl/mylist/web_tt2 directory, and your site web templates in the `~/home/sympa/etc/web_tt2` directory.

Note that web colors are defined in Sympa's main Makefile (see Compilation and installation).

Internationalization

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

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

Sympa internationalization

Every message sent by Sympa to users, owners and editors is outside the code, in a message catalog. These catalogs are located in the `/home/sympa/locale` directory.

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

List internationalization

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

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

User internationalization

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

Topics

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

This parameter is different from the `msg_topic` parameter used to tag list messages.

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

```
<topic1_name>
title   <topic1 title>
```

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

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

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

Authorization scenarios

See Authorization scenarios.

Loop detection

Sympa uses multiple heuristics to avoid loops in Mailing lists.

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

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

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

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

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

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

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

`loop_command_decrease_factor` parameter

Tasks

A task is a sequence of simple actions which realize a complex routine. It is executed in the background by the task manager daemon and allows listmasters to automate the processing of recurring tasks. For example, a task sends to the subscribers of a list a message to remind them of their subscription on a yearly basis.

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

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

A task can be whether global or related to a list.

List task creation

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

1. `/home/sympa/expl/`"<list name>"`/`;
2. `/home/sympa/etc/list_task_models/`;
3. `/home/sympa/bin/etc/list_task_models/`.

See also List model files to know more about standard list models provided with Sympa.

Global task creation

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

1. `/home/sympa/etc/global_task_models/`;
2. `/home/sympa/bin/etc/global_task_models/`.

Model file format

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

- Comment lines begin by '#' and are not interpreted by the task manager.
- Label lines begin by '/' and are used by the next command (see below).
- References are enclosed between brackets '[]'. They refer to a value depending on the object

of the task (for instance `[list→name]`). Those variables are instantiated when a task file is created from a model file. The list of available variables is the same as for templates (see List template files) plus `[creation_date]` (see below).

- Variables store results of some commands and are paramaters for others. Their names begin with '@'.
- A date value may be written in two ways:
  - Absolute dates follow the format: xxxxYxxMxxDxxHxxMin. Y is the year, M the month (1-12), D the day (1-28|30|31, leap-years are not managed), H the hour (0-23), Min the minute (0-59). H and Min are optional. For instance, 2001y12m4d44min is the 4th of December 2001 at 00h44.
  - Relative dates use the `[creation_date]` or `[execution_date]` references. `[creation_date]` is the date when the task file is created, `[execution_date]` when the command line is executed. A duration may follow with the '+' or '-' operators. The duration is expressed like an absolute date whose all parameters are optional. Examples: [creation_date], [execution_date]+1y, [execution_date]-6m4d.
- Command arguments are separated by commas and enclosed between parenthesis '()'.

Here is the list of the currently available commands:

- `stop ()`
  Stops the execution of the task and deletes the task file.
- `next (<date value>, <label>)`
  Stops the execution. The task will go on at the date value and begin at the label line.
- `<@deleted_users> = delete_subs (@<user_selection>)`
  Deletes the `@user_selection` email list and stores user emails successfully deleted in `@deleted_users`.
- `send_msg (<@user_selection>, <template>)`
  Sends the template message to emails stored in `@user_selection`.
- `@user_selection = select_subs (<condition>)`
  Stores emails which match the condition in `@user_selection`. See Authorization Scenarios to know how to write conditions. Only available for list models.
- `create (global | list (<list name>), <model type>, <model>)`
  Creates a task for object with model file `~model type.model.task`.
- `chk_cert_expiration (<template>, <date value>)`
  Sends the template message to emails whose certificate has expired or will expire before the date value.
- `update_crl (<file name>, <date value>)`
  Updates certificate revocation lists (CRL) which are expired or will expire before the date value. The file stores the CRL's URLs.
- `purge_orphan_bounces()`
  Cleans bounces by removing the unsubscribed users' archive.
- `eval_bouncers()`
  Evaluates all bouncing users of all lists and gives them a score from 0 to 100. (`0` is for non bouncing users and `100` is for users who should be removed).
- `process_bouncers()`
  Executes actions defined in list configuration on all bouncing users, according to their score.

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

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

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

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

Model file examples

- remind.annual.task;
- expire.annual.task;
- crl_update.daily.task.

<code>

```
title.gettext daily update of the certificate revocation list
/ACTION
update_crl (CA_list, [execution_date]+1d)
next ([execution_date] + 1d, ACTION)
```

</CODE>

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 especialy 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

Sympa will use these files only if the list is configured in `include2 user_data_source` mode. 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). This files can be template files.

Sympa looks for them in the following order:

1. `/home/sympa/expl/mylist/data_sources/<file>.incl`;
2. `/home/sympa/etc/data_sources/<file>.incl`;
3. `/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_list`, `include_remote_sympa_list`, `include_sql_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_inlude`.

Example:

- in the list configuration file:

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

- in `myfile.incl`:

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

- resulting data inclusion file:

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

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`;
2. `/home/sympa/etc/mail_tt2/<file>.tt2`;
3. `/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:

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

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

Example:

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

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

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

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

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 prefixe 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;
- `[% url %]`: 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 alises. It is used by the `alias_manager` script.

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 algorythm:

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

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

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>
        <sql>
           <type>oracle</type>
           <host>sqlserv.admin.univ-x.fr</host>
           <user>stdutilisateur</user>
           <pwd>monsecret</pwd>
           <name>les_etudiants</name>
           <query>SELECT DISTINCT email FROM etudiant</query>
        </sql>
</list>

subject [% subject %]

status [% status %]

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

[% END %]
visibility noconceal

send privateoreditorkey

Web_archive
  access public

subscribe open_notify

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

lang [% language %]

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

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

```
    [% END %]
 [% END %]

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

 [% END %]
 ttl 360
```

The XML file format should comply with the following rules:

- The root element is `<list>`.
- One XML element is mandatory: `<listname>` contains the name of the list. That 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:

```
# Do not allow the public_anonymous profile
public_anonymous hidden
* read
```

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 named `default` means any other parameter.

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

For example, `listmaster` will not match rules refering to `owner` or `privileged_owner`.

Examples:

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

        priority  owner,privileged_owner                  read
        priority  listmaster                              write

        # only privileged owner can modify  editor parameter, send, ...
        editor privileged_owner write

        send                owner                                   read
        send                privileged_owner,listmaster   write

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

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

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

The following rules are hard coded in *WWSympa*:

- Only 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 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 -remove_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 [% subject %]

status [% status %]

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

[% END %]
```

```
visibility noconceal

send privateoreditorkey

web_archive
  access public

subscribe open_notify

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

lang [% language %]

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

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

   [% END %]
[% END %]

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

[% END %]
ttl 360
```

param_constraint.conf

This file is 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.

Example:

```
<?xml version="1.0" ?>
<family>
  <list>
    <listname>liste1</listname>
    <subject>a list example</subject>
    <description/>
    <status>open</status>
    <shared_edit>editor</shared_edit>
    <shared_read>private</shared_read>
    <language>fr</language>
    <owner multiple="1">
      <email>foo@cru.fr</email>
```

```
        <gecos>C.R.U.</gecos>
      </owner>
      <owner multiple="1">
        <email>foo@emnsp.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@enmsp.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 instanciation.
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 trigerred 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

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.

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

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

/etc/postfix/master.cf
    sympa     unix  -       n       n       -       -       pipe
      flags=R user=sympa argv=/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.

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
    tokens = nemes.1.split('\.');
    THROW autofamily "SYNTAX ERROR: wrong listname syntax" IF (tokens.size != 2 || ! occupations.${
    age = tokens.1 div 10;
    %]

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

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

    include_ldap_query
    attrs mail
    filter (&(objectClass=inetOrgPerson)(employeeType=[% occupations.${tokens.0}.abbr %])(personAge
    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)
- `visibility intranet` (view)
- `visibility noconceal` (view)
- `visibility secret` (view)

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;
- `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.
- 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)
- `send editorkey` (view)
- `send editorkeyonly` (view)
- `send editorkeyonlyauth` (view)
- `send intranet` (view)
- `send intranetorprivate` (view)
- `send newsletter` (view)
- `send newsletterkeyonly` (view)
- `send private` (view)
- `send private_smime` (view)
- `send privateandeditorkey` (view)
- `send privateandnomultipartoreditorkey` (view)
- `send privatekey` (view)
- `send privatekeyandeditorkeyonly` (view)
- `send privateoreditorkey` (view)
- `send privateorpublickey` (view)
- `send public` (view)
- `send public_nobcc` (view)
- `send publickey` (view)
- `send publicnoattachment` (view)
- `send publicnomultipart` (view)

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.

- `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.

custom_subject

`custom_subject` *value*

This parameter is optional. It specifies a string which is added to the subject of distributed messages (intended to help users who do not use automatic tools to sort incoming messages). This string will be surrounded by '[]' characters.

The custom subject can also refer to list variables ([list->sequence] in the example below).

Example:

```
custom_subject sympa-users
```

Other example:

```
custom_subject newsletter num [list->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/expl/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.

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);
- `subscribe` `auth_notify` (view);
- `subscribe` `auth_owner` (view);
- `subscribe` `closed` (view);
- `subscribe` `intranet` (view);
- `subscribe` `intranetorowner` (view);
- `subscribe` `open` (view);
- `subscribe` `open_notify` (view);
- `subscribe` `open_quiet` (view);
- `subscribe` `owner` (view);
- `subscribe` `smime` (view);
- `subscribe` `smimeorowner` (view).

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);
- `unsubscribe` `auth_notify` (view);
- `unsubscribe` `closed` (view);
- `unsubscribe` `open` (view);
- `unsubscribe` `open_notify` (view);
- `unsubscribe` `owner` (view).

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);
- `add` `closed` (view);
- `add` `owner` (view);
- `add` `owner_notify` (view).

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);
- `del closed` (view);
- `del owner` (view);
- `del owner_notify` (view).

invite

(Default value: `owner`)

The invite command is used to invite someone to subscribe. It should be prefered to the `add` command in most cases. This parameter define 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);
- `review intranet` (view);
- `review listmaster` (view);
- `review owner` (view);
- `review private` (view);
- `review public` (view).

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);
- `remind owner` (view).

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);
- `d_read private` (view);
- `d_read p` (view);
- `d_read public` (view).

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);
- `d_edit owner` (view);
- `d_edit private` (view);
- `d_edit p` (view);
- `d_edit public` (view).

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. Version prior to 5.2, archives wre 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 “wwsympa.conf” (parameter arc_path. version 5.2 and later use only this archive repository.

archive

If the “config” file contains an “archive” paragraph, Sympa will manage an archive for this list.

Example:

```
archive
period week
access private
```

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

`period day | week | month | quarter | year`

This parameter specifies how archiving is organized: by `day`, `week`, `month`, `quarter` or `year`. Generation of automatic list archives requires the creation of an archive directory at the root of the list directory (`/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
```

access

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

Predefined authorization scenarios are:

- `access closed` (view);
- `access intranet` (view);
- `access listmaster` (view);
- `access owner` (view);
- `access private` (view);
- `access public` (view).

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.

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:

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

bouncers_level1

- `rate`
  (Default value: `bouncers_level1_rate config 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 80 are level 1 bouncers.
- `action`
  (Default value: `bouncers_level1_action config parameter`)
  This parameter defines which task is automaticaly applied on level 1 bouncing users: for example, automatically notify all level 1 bouncers.
- `Notification`
  (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 adresses of the users concerned and the name of the action perform.

bouncers_level2

- `rate`
  (Default value: `bouncers_level2_rate config 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`
  (Default value: `bouncers_level1_action config parameter`)
  This parameter defines which task is automatically applied on level 2 bouncing users: for example, automatically notify all level 2 bouncers.
- `Notification`
  (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 adresses of the users concerned and the name of the action performed.

Example:

```
    ## All bouncing adresses 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% VERP 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: `file|database`, if using an RDBMS)

`user_data_source file | database | include | include2`

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

- `user_data_source file`
  When this value is used, subscriber data is 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 in place 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 maitained 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 parameter. Their TTL (time-to-live) within Sympa can be controlled using this parameter. The default value is 3600.

include_list

`include_list` `listname`

This parameter will be interpreted only if `user_data_source` is set to `include` or `include2`. All subscribers of list `listname` become members of the current list. You may include as many lists as required, using one `include_list listname` line for each included list. Any list at all may be included; the `user_data_source` definition of the included list is irrelevant, and you may therefore include lists which are also defined by the inclusion of other lists. Be careful, however, not to include list `A` in list `B` and then list `B` in list `A`, since this 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 controled by the review authorization scenario.

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

- `remote_host` *remote_host_name*;
- `port` *port* (Default 443);
- `path` *absolute path* (in most cases, for a list name foo /sympa/dump/foo).

Because HTTPS offert 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 adress). 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`

This parameter will be interpreted only if the `user_data_source` value is set to `include`. 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_envORACLE_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 whicj datasource each list member comes (usefullwhen 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 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.
- `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_version` *ciphers used* (Default value: `ALL`)
  If using SSL, this parameter specifies which subset of cipher suites are permissible for this connection, using the standard OpenSSL string format. The default value of Net::LDAPS for ciphers is `ALL`, which 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_version` *ciphers used* (Default value: `ALL`)
  If using SSL, this parameter specifies which subset of cipher suites are permissible for this connection, using the standard OpenSSL string format. The default value of Net::LDAPS for ciphers is `ALL`, which 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`

This parameter will be interpreted only if the `user_data_source` value is set to `include`. 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. This paragraph is used only if `user_data_source` is set to `include`. Using this method you should be able to include any *exotic* data source that is not supported by Sympa. The paragraph is made of the following entries:

- `url` *url_of_remote_file*

  This is the URL of the remote file to include.
- `user` *user_name*

  This entry is optional. 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 regurlaly sends to the subscribers a message which reminds them of their subscription to the list.

Example:

```
remind annual
```

expire_task

(Default value: no default value)

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

Example:

```
expire 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);
- `review intranet` (view);
- `review listmaster` (view);
- `review owner` (view);
- `review private` (view);
- `review public` (view).

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
```

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: `spam_protection robot parameter`)

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: `web_archive_spam_protection robot parameter`)

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.

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 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 suscriber 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.

Bounce management

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

Bounces are received at the `mylist-owner` address (note that the `-owner` suffix can be customized, see return_path_suffix), which should be sent to the `bouncequeue` program through aliases:

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

`bouncequeue` (see Binaries) stores bounces in a `/home/sympa/spool/bounce/` spool.

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

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

With this information, the automatic bounce management is possible:

The automatic task `eval_bouncer` gives a score for each bouncing user. The score, between 0 to 100, allows the classification of bouncing users in two levels (level 1 or 2). According to the level, automatic actions are executed periodically by the `process_bouncers` task.

The score evaluation main parameters are:

- `Bounces count`: the number of bouncing messages received by Sympa for the user.
- `Type rate`: bounces are classified depending on the type of errors generated on the user side. If the error type is `mailbox is full` (i.e. a temporary 4.2.2 error type), the type rate will be 0.5, whereas permanent errors (5.x.x) have a type rate equal to 1.
- `Regularity rate`: this rate tells whether bounces were received regularly, compared to list traffic. The list traffic is deduced from the `msg_count` file data.

The score formula is:

```
Score = bounce_count * type_rate * regularity_rate
```

To avoid making decisions (i.e. defining a score) without enough relevant data, the score is not evaluated if:

- The number of received bounces is lower than `minimum_bouncing_count` (see minimum_bouncing_count).
- The bouncing period is shorter than `minimum_bouncing_period` (see minimum_bouncing_period).

Bouncing list member entries expire after a given period of time. The default period is 10 days, but it can be customized if you write a new `expire_bounce` task (see expire_bounce_task).

You can define the limit between each level through the **List configuration pannel**, in subsection **Bounce settings** (see bouncers_level1). The principle consists in associating a score interval with a level.

You can also define which action must be applied on each category of user (see bouncers_level1). Each time an action will be performed, a notification email will be sent to the person of your choice (see bouncers_level1).

VERP

VERP (Variable Envelop Return Path) is used to ease automatic recognition of subscribers email addresses when receiving a bounce. If VERP is enabled, the subscriber address is encoded in the return path itself, so that the Sympa bounce management process (bounced) will use the address the bounce was received for to retrieve the subscriber email. This is very useful because sometimes, non delivery report do not contain the initial subscriber email address but an alternative address where messages are forwarded. VERP is the only solution to detect automatically these subscriber errors. However, the cost of VERP is significant, indeed VERP requires to distribute a separate message for each subscriber and breaks the bulk emailer grouping optimization.

In order to benefit from VERP and keep the distribution process fast, Sympa enables VERP only for a share of the list members. If texttt `verp_rate` (see verp_rate) is 10%, then after 10 messages distributed in the list all subscribers have received at least one message where VERP was enabled. Later, distribution message enables VERP also for all users where some bounces were collected and analyzed by the previous VERP mechanism.

If VERP is enabled, the format of the messages return path are as follows:

```
Return-Path: <bounce+user==a==userdomain==listname@listdomain>
```

Note that you need to set a mail alias for the generic bounce+* alias (see Robot aliases).

ARF

ARF (Abuse Feedback Reporting Format) is a standard for reporting abuse. It is implemented mainly in the AOL email user interface. AOL servers propose to mass mailer to received automatically the users complain by formated messages. Because many subscribers do not remember how to unsubscribe they use ARF when provided by their user interface. It may be useful to configure the ARF management in Sympa. It is really simple: all what you have to do is to create a new alias for each virtual robot as the following:

```
abuse-feedback-report:       "| /home/sympa/bin/bouncequeue sympa@samplerobot"
```

Then register this address as your loop back email address with ISP (for exemple AOL). This way, messages to that email adress are processed by the bounced deamon and opt-out opt-out-list abuse and automatically processed. If the bounce service can remove a user, the message report feedback is forwarded to the list owner. Unrecognized messages are forwarded to the listmaster.

Antivirus

Sympa lets you use an external antivirus solution to check incoming mails. In this case you must set the `antivirus_path` and `antivirus_args` configuration parameters (see Antivirus_plug-in. Sympa is already compatible with McAfee/uvscan, Fsecure/fsav, Sophos, AVP, Trend Micro/VirusWall and Clam Antivirus. For each email received, Sympa extracts its MIME parts in the `/home/sympa/spool/tmp/antivirus` directory and then calls the antivirus software to check them. When a virus is detected, Sympa looks for the virus name in the virus scanner STDOUT and sends a `your_infected_msg.tt2` warning to the sender of the email. The dmail is saved as 'bad' and the working directory is deleted (except if Sympa is running in debug mode).

Using Sympa with LDAP

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

- authentication using LDAP directory instead of the Sympa internal storage of password (see auth.conf);
- named filters used in authorization scenario condition (see Named Filters);
- LDAP extraction of list subscribers (see user_data_source);
- LDAP extraction of list owners or editors (see Data inclusion file);
- mail aliases stored in LDAP (see Alias manager).

Sympa with S/MIME and HTTPS

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

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

Signed message distribution

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

Use of S/MIME signatures by Sympa itself

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

So Sympa cannot apply the S/MIME authentication method to a command parsed in the `Subject:` field of a message or through the `-subscribe` or `-unsubscribe` email addresses.

Use of S/MIME encryption

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

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

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

S/Sympa configuration

Installation

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

Managing user certificates

User certificates are automatically caught by Sympa when receiving a signed S/MIME messsage, so if Sympa needs to send encrypted messages to this user, it can perform encryption using this certificate. This works fine, but it is not conpliant with the PKI theory: Sympa should be able to search for user certificates using a PKI certificate directory (LDAP).

That‘s why Sympa tests the key usage certificate attribute to known if the certificate allows both encryption and signature.

Certificates are stored as PEM files in the `/home/sympa/expl/X509-user-certs/` directory. Files are named user@some.domain@enc or user@some.domain@sign (the `@enc` and `@sign` suffixes are used according to certificates usage). No other tool is provided by Sympa in order to collect this certificate repository, but you can easily imagine your own tool to create those files.

Configuration in sympa.conf

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

The basic requirement is to let Sympa know where to find the binary file for the OpenSSL program and the certificates of the trusted certificate authority. This is made using the optional parameters `openSSL` and `capath` and / or `cafile`.

- `openssl`: the path for the OpenSSL binary file, usually `/usr/local/ssl/bin/openSSL`;
- `cafile` (or `capath`): the path of a bundle (or path of the directory) of trusted CA certificates. The file `~/home/sympa/bin/etc/cabundle.crt` included in Sympa distribution can be used.
  The `cafile` file (or the `capath` directory) should be shared with your Apache+mod_ssl configuration. This is required because Sympa's web interface gets user certificates information from Apache mod_ssl module;
- `key_password`: the password used to protect all list private keys.

Configuration to recognize S/MIME signatures

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

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

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

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

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

It as also possible to mix various authentication methods in a single authorization scenario. The following example, `send.private_key`, requires either an MD5 return key or an S/MIME signature:

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

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

distributing encrypted messages

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

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

The S/Sympa encryption feature in the distribution process assumes that Sympa has received an encrypted message for some list. To be able to encrypt a message for a list, the sender must have some access to an X509 certificate for the list. So the first requirement is to install a certificate and a private key for the list. The mechanism whereby certificates are obtained and managed is complex. Current versions of S/Sympa assume that list certificates and private keys are installed by the listmaster using the `/home/sympa/bin/p12topem.pl` script. This script allows you to install a PKCS#12 bundle file containing a private key and a certificate using the appropriate format.

It is a good idea to have a look at the OpenCA documentation (http://www.openssl.org) and/or PKI providers' web documentation. You can use commercial certificates or home-made ones. Of course, the certificate must be approved of for email applications, and issued by one of the trusted CA's described in the `cafile` file or the `capath` Sympa configuration parameter.

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

Use of navigator to obtain X509 list certificates

In many cases email X509 certificates are distributed through a web server and loaded into the browser using your mouse: Mozilla or internet Explorer allow certificates to be exported to a file.

Here is a way to install a certificat for a list:

- Get a list certificate is to obtain a personal email certificate for the canonical list address in your browser as if it was your personal certificate.
- Export the intended certificate it. The format used by Netscape is `pkcs#12`. Copy this file to the list home directory.
- Convert the pkcs#12 file into a pair of PEM files: `cert.pem` and `private_key`, using the `/home/sympa/bin/p12topem.pl` script. Use `p12topem.pl -help` for details.
- Be sure that `cert.pem` and `private_key` are owned by sympa with `r` access.
- As soon as a certificate is installed for a list, the list homepage includes a new link to load the certificate in the user's browser, and the welcome message is signed by the list.

Managing certificates with tasks

You may automate the management of certificates with two global task models provided with Sympa. See Tasks to know more about tasks. Report to the chk_cert_expiration_task and crl_update_task `sympa.conf` parameters to configure your Sympa to use these facilities.

chk_cert_expiration.daily.task model

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

crl_update.daily.task model

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

Using Sympa commands

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

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

Example:

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

LISTS
INFO Sympa-users
REVIEW Sympa-users
QUIT
```

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

User commands

- `HELP`

  Provides instructions for the use of Sympa commands. The result is the content of the `helpfile.tt2` template file.
- `INFO` *listname*

  Provides the parameters of the list specified (owner, subscription mode, etc.) and its description. The result is the content of `~welcome[.mime]`.
- `LISTS`

  Provides the names of lists managed by Sympa. This list is generated dynamically, using the `visibility` parameter (see Visibility). The `lists.tt2` template defines the message returned by the `LISTS` command.
- `REVIEW` *listname*

Provides the addresses of subscribers if the run mode authorizes it. See the review parameter for the configuration file of each list, which controls read authorizations for the subscriber list. Since subscriber addresses can be abused by spammers, it is strongly recommended that you **only authorize owners to access the subscriber list**.

- `WHICH`

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

  Provides statistics for the specified list: number of messages received, number of messages sent, megabytes received, megabytes sent. This is the contents of the `stats` file.
  Access to this command is controlled through the `review` parameter.
- `INDEX` *listname*

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

  To retrieve archives for list (see above). Access rights are the same as for the `REVIEW` command. See the review parameter.
- `LAST` *listname*

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

  Requests sign-up to the specified list. The *firstname* and *name* parameters are optional. If the list is configured with a restricted subscription (see the subscribe parameter), this command is sent to the list owner for approval.
- `INVITE` *listname user@host name*

  Invites someone to subscribe to the list specified. The *name* parameter is optional. The command is similar to `ADD`, but the person specified is not added to the list but invited to subscribe to it in accordance with the subscribe parameter.
- `SIGNOFF` *listname* [*user@host*]

  Requests unsubscription from the specified list. `SIGNOFF *` means unsubscription from all lists.
- `SET` *listname* `DIGEST`

  Puts the subscriber in *digest* mode for the *listname* list. Instead of receiving email from the list in a normal manner, the subscriber will periodically receive it in a digest. This digest compiles a group of messages from the list, using multipart/digest mime format. The sending period for these digests is regulated by the list owner using the digest parameter. See the SET LISTNAME MAIL command and the reception parameter.
- `SET` *listname* `SUMMARY`

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

  Puts subscriber in `nomail` mode for the *listname* list. This mode is used when a subscriber no longer wants to receive email from the list, but nevertheless wishes to retain the possibility of posting to the list. This mode therefore prevents the subscriber from unsubscribing and subscribing later on. See the SET LISTNAME MAIL command and the reception parameter.

- `SET` *listname* `TXT`
  Puts subscriber in `txt` mode for the *listname* list. This mode is used when a subscriber wishes to receive emails sent in both format, txt/html and txt/plain only, in txt/plain format. See the reception parameter.
- `SET` *listname* `HTML`
  Puts subscriber in `html` mode for the *listname* list. This mode is used when a subscriber wishes to receive emails sent in both format, txt/html and txt/plain only, in txt/html format. See the reception parameter.
- `SET` *listname* `URLIZE`
  Puts subscriber in `urlize` mode for the *listname* list. This mode is used when a subscriber wishes not to receive attached files. The attached files are replaced by a URL leading to the file stored on the list site. See the reception parameter.
- `SET` *listname* `NOT_ME`
  Puts subscriber in `not_me` mode for the *listname* list. This mode is used when a subscriber wishes not to receive back the message that he/she has sent to the list. See reception parameter.
- `SET` *listname* `MAIL`
  Puts the subscriber in `normal` mode (default) for the *listname* list. This option is mainly used to cancel the `nomail`, `summary` or `digest` modes. If the subscriber was in `nomail` mode, he or she will receive email from the list in a normal manner again. See the SET LISTNAME NOMAIL command and the reception parameter. Moreover, this mode allows message topic subscription (see Message topics).
- `SET` *listname* `CONCEAL`
  Puts the subscriber in `conceal` mode for the *listname* list. The subscriber will then become invisible during `REVIEW` on this list. Only owners will see the whole subscriber list. See the SET LISTNAME NOCONCEAL command and the Visibility parameter.
- `SET` *listname* `NOCONCEAL`
  Puts the subscriber in `noconceal` mode (default) for the *listname* list. The subscriber will then become visible during `REVIEW` of this list. The `conceal` mode is therefore cancelled. See the SET LISTNAME CONCEAL command and the Visibility parameter.
- `QUIT`
  Ends acceptance of commands. This can be useful when the message contains additional lines, as for example in the case where a signature is automatically added by the user‘s email program (MUA).
- `CONFIRM` *key*
  If the `send` parameter of a list is set to `privatekey`, `publickey` or `privateorpublickey`, messages are only distributed in the list after an authentication phase by return mail, using a one-time password (numeric key). For this authentication, the sender of the message is requested to post the `CONFIRM` *key* command to Sympa.
- `QUIET`
  This command is used for silent (mute) processing: no performance report is returned for commands prefixed with `QUIET`.

Owner commands

Some administrative requests are only available to list owners. They are essential for all procedures in limited access mode, and to perform requests in place of users. These comands are:

- `ADD` *listname user@host firstname name*
  Add command similar to `SUBSCRIBE`. You can avoid user notification by using the `QUIET` prefix (i.e.: `QUIET ADD`).
- `DELETE` *listname user@host*
  Delete command similar to `SIGNOFF`. You can avoid user notification by using the `QUIET` prefix (i.e.: `QUIET DELETE`).
- `REMIND` *listname*
  `REMIND` is used by list owners in order to send an individual service reminder message to each subscriber. This message is made by parsing the `remind.tt2` file.
- `REMIND *`
  `REMIND *` is used by the listmaster to send to each subscriber of any list a single message with a summary of his/her subscriptions. In this case, the message sent is built by parsing the `global_remind.tt2` file. For each list, Sympa tests whether the list is configured as hidden to each subscriber (parameter lparam visibility). By default, the use of this command is restricted to listmasters. Processing may take a lot of time!

These commands can be prefixed with `QUIET` to indicate processing without acknowledgment of receipt.

Moderator commands

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

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

test.txt · Last modified: 2007/02/20 15:04 by olivier.lumineau@cru.fr