This page was obsoleted.

× List.pm

This module includes list processing functions.

Here are described functions about:

- Message distribution in a list
- Sending using templates
- Service messages
- Notification message
- Topic messages
- Scenario evaluation

Follows a description of structure and access on list parameters.

Functions for message distribution

distribute_message(), send_msg(), send_msg_digest().

These functions are used to message distribution in a list.

distribute_msg()

Prepares and distributes a message to a list:

- updates the list stats
- Loads information from message topic file if exists and adds X-Sympa-Topic header
- hides the sender if the list is anonymoused (list config: anonymous_sender) and changes name of msg topic file if exists.
- adds custom subject if necessary (list config: custom_subject)
- archives the message
- changes the reply-to header if necessary (list config: reply_to_header)
- removes unwanted headers if present (config: remove_headers))
- adds useful headers (X-Loop,X-Sequence,Errors-to,Precedence,X-no-archive - list config: custom_header)
- adds RFC 2919 header field (List-Id) and RFC 2369 header fields (list config: rfc2369_header_fields)
- stores message in digest if the list accepts digest mode (encrypted message can't be included in digest)
- sends the message by calling List::send_msg() (see list-send-msg).

IN:

1. `self` (+): ref(List) - the list concerned by distribution
2. `message` (+): ref(Message) - the message to distribute

OUT: result of List::send_msg() function (number of sendmail process)

send_msg()

This function is called by List::distribute_msg() (see list-distribute-msg) to select subscribers according to their reception mode and to the `Content-Type` header field of the message. Sending are grouped according to their reception mode:

- normal: add a footer if the message is not protected (then the message is `altered`) In a message topic context, selects only one who are subscribed to the topic of the message to distribute (calls to select_subcribers_for_topic(), see list-select-subscribers-for-topic).
- notice
- txt: add a footer
- html: add a footer
- urlize: add a footer and create an urlize directory for Web access

The message is sent by calling List::mail_message() (see mail-mail-message). If the message is `smime\_crypted` and the user has not got any certificate, a message service is sent to him.

IN:

1. `self` (+): ref(List) - the list concerned by distribution
2. `message` (+): ref(Message) - the message to distribute

OUT: `$numsmtp`: addition of mail::mail_message() function results (= number of sendmail process) undef

send_msg_digest()

Sends a digest message to the list subscribers with reception digest, digestplain or summary: it creates the list of subscribers in various digest modes and then creates the list of messages. Finally sending to subscribers is done by calling List::send_file() function (see list-send-file, page) with mail template `digest`, `digestplain` or `summary`.

IN:

1. `self` (+): ref(List) - the concerned list

OUT:

- 1 *if sending*
- 0 *if no subscriber for sending digest, digestplain or summary*
- undef

Functions for template sending

send_file(), send_global_file().

These functions are used by others to send files. These files are made from template given in parameters.

send_file()

Sends a message to a user, relative to a list. It finds the $tpl.tt2 file to make the message. If the list has a key and a certificat and if openssl is in the configuration, the message is signed. The parsing is done with variable $data set up first with parameter $context and then with configuration, here are set keys:

- *if $who=SCALAR then*
  - `user.password`
  - *if $user key is not defined in $context then* `user.email`(:= $who), `user.lang` (:= list lang) *and if the user is in DB then* `user.attributes` (:= attributes in DB user_table) are defined
  - *if $who is subscriber of $self then* `subscriber.date subscriber.update\_date` *and if exists then* `subscriber.bounce subscriber.first\_bounce` are defined
- `return\_path`: used for SMTP `MAIL From` field or `X-Sympa-From:` field
- `lang`: the user lang or list lang or robot lang
- `fromlist`: `From:` field, pointed on list
- `from`: `From:` field, pointed on list *if no defined in $context*
- `replyto`: *if openssl is is sympa.conf and the list has a key ('private_key') and a certificat ('cert.pem') in its directory*
- `boundary`: boundary for multipart message *if no contained in $context*
- `conf.email conf.host conf.sympa conf.request conf.listmaster conf.wwsympa\_url conf.title`: updated with robot config
- `list.lang list.name list.domain list.host list.subject list.dir list.owner`(ref(ARRAY)): updated with list config

The message is sent by calling mail::mail_file() function (see mail-mail-file, page).

IN:

1. `self` (+): ref(List)
2. `tpl` (+): template file name without .tt2 extension ($tpl.tt2)
3. `who` (+): SCALAR ref(ARRAY) - recepient(s)
4. `robot` (+): robot
5. `context`: ref(HASH) - for the $data set up

OUT: 1 undef

send_global_file()

Sends a message to a user not relative to a list. It finds the $tpl.tt2 file to make the message. The parsing is done with variable $data set up first with parameter $context and then with configuration, here are set keys:

- `user.password user.lang`
- *if $user key is not defined in $context then* `user.email` (:= $who)

- `return\_path`: used for SMTP `MAIL From` field or `X-Sympa-From:` field
- `lang`: the user lang or robot lang
- `from`: `From:` field, pointed on SYMPA *if no defined in $context*
- `boundary`: boundary for multipart message *if no defined in $context*
- `conf.email conf.host conf.sympa conf.request conf.listmaster conf.wwsympa\_url conf.title`: updated with robot config
- `conf.version`: Sympa version
- `robot\_domain`: the robot

The message is sent by calling mail::mail_file() function (see mail-mail-file, page).

IN:

1. `tpl` (+): template file name (filename.tt2), without .tt2 extension
2. who (+): SCALAR ref(ARRAY) - recepient(s)
3. `robot` (+): robot
4. `context`: ref(HASH) - for the $data set up

OUT: 1 undef

Functions for service messages

archive_send(), send_to_editor(), request_auth(), send_auth().

These functions are used to send services messgase, correponding to a result of a command.

archive_send()

Sends an archive file ($file) to $who. The archive is a text file, independant from web archives. It checks if the list is archived. Sending is done by callingList::send_file() (see list-send-file, page) with mail template `archive`.

IN:

1. `self` (+): ref(List) - the concerned list
2. who (+): recepient
3. `file` (+): name of the archive file to send

OUT: - undef

send_to_editor()

Sends a message to the list editor for a request concerning a message to distribute. The message awaiting for moderation is named with a key and is set in the spool queuemod. The key is a reference on the message for editor. The message for the editor is sent by calling List::send_file() (see list-send-file, page) with mail template `moderate`. In msg_topic context, the editor is asked to tag the message.

IN:

1. `self` (+): ref(List) - the concerned list
2. `method`: 'md5' - for `editorkey` 'smtp' - for `editor`
3. `message` (+): ref(Message) - the message to moderate

OUT: $modkey - the moderation key for naming message waiting for moderation in spool queuemod. undef

request_auth()

Sends an authentification request for a requested command. The authentification request contains the command to be send next and it is authentified by a key. The message is sent to user by calling List::send_file() (see list-send-file, page) with mail template `request\_auth`.

IN:

1. `self`: ref(List) *not required if $cmd = `remind`*.
2. `email`(+): recepient, the requesting command user
3. `cmd`:
   - *if $self then* 'signoff' 'subscribe' 'add' 'del' 'remind'
   - *else* 'remind'
4. `robot` (+): robot
5. `param`: ARRAY
   - 0: used *if $cmd =*'*subscribe*' '*add*' '*del*' '*invite*'
   - 1: used *if $cmd =*'*add*'

OUT: 1 undef

send_auth()

Sends an authentifiaction request for a message sent for distribution. The message for distribution is copied in the authqueue spool to wait for confirmation by its sender . This message is named with a key. The request is sent to user by calling List::send_file() (see list-send-file, page) with mail template `send\_auth`. In msg_topic context, the sender is asked to tag his message.

IN:

1. `self`(+): ref(List) - the concerned list
2. `message`(+): ref(Message) - the message to confirm

OUT: $modkey, the key for naming message waiting for confirmation in spool queuemod. undef

Functions for message notification

send_notify_to_listmaster(), send_notify_to_owner(), send_notify_to_editor(), send_notify_to_user().

These functions are used to notify listmaster, list owner, list editor or user about events.

send_notify_to_listmaster()

Sends a notice to listmaster by parsing `listmaster\_notification` template. The template makes a specified or a generic treatement according to variable $param.type (:= $operation parameter). The message is sent by calling List::send_file() (see list-send-file, page) or List::send_global_file() (see list-send-global-file, page) according to the context: global or list context. Available variables for the template are set up by this function, by $param parameter and by List::send_global_file() or List::send_file().

IN:

1. `operation` (+): notification type, corresponds to $type in the template
2. `robot` (+): robot
3. `param` (+): ref(HASH) ref (ARRAY) - values for variable used in the template:
   - *if ref(HASH) then* variables used in the template are keys of this HASH. These following keys are required in the function, according to $operation value:
     - 'listname'(+) *if $operation=('request_list_creation'* '*automatic_bounce_management')*
   - *if ref(ARRAY) then* variables used in template are named as: $param0, $param1, $param2, ...

OUT: 1 undef

send_notify_to_owner()

Sends a notice to list owner(s) by parsing `listowner\_notification` template. The template makes a specified or a generic treatement according to variable $param.type (:= $operation parameter). The message is sent by calling List::send_file() (see list-send-file, page). Available variables for the template are set up by this function, by $param parameter and by List::send_file().

IN:

1. `self` (+): ref(List) - the list for owner notification
2. `operation` (+): notification type, corresponds to $type in the template
3. `param` (+): ref(HASH) ref (ARRAY) - values for variable used in the template:
   - *if ref(HASH) then* variables used in the template are keys of this HASH.
   - *if ref(ARRAY) then* variables used in template are named as: $param0, $param1, $param2, ...

OUT: 1 undef

send_notify_to_editor()

Sends a notice to list editor(s) by parsing `listeditor\_notification` template. The template makes a specified or a generic treatement according to variable $param.type (:= $operation parameter). The message is sent by calling List::send_file() (see list-send-file, page). Available

variables for the template are set up by this function, by $param parameter and by List::send_file().

IN:

1. `self` (+): ref(List) - the list for editor notification
2. `operation` (+): notification type, corresponds to $type in the template
3. `param` (+): ref(HASH) ref (ARRAY) - values for variable used in the template:
   - *if ref(HASH) then* variables used in the template are keys of this HASH.
   - *if ref(ARRAY) then* variables used in template are named as: $param0, $param1, $param2, ...

OUT: 1 undef

send_notify_to_user()

Sends a notice to a user by parsing `user\_notification` template. The template makes a specified or a generic treatement according to variable $param.type (:= with $operation parameter). The message is sent by calling List::send_file() (see list-send-file, page). Available variables for the template are set up by this function, by $param parameter and by List::send_file().

IN:

1. `self` (+): ref(List) - the list for owner notification
2. `operation` (+): notification type, corresponds to $type in the template
3. `user` (+): user email to notify
4. `param` (+): ref(HASH) ref (ARRAY) - values for variable used in the template:
   - *if ref(HASH) then* variables used in the template are keys of this HASH.
   - *if ref(ARRAY) then* variables used in template are named as: $param0, $param1, $param2, ...

OUT: 1 undef

Functions for topic messages

is_there_msg_topic(), is_available_msg_topic(), get_available_msg_topic(), is_msg_topic_tagging_required, automatic_tag(), compute_topic(), tag_topic(), load_msg_topic_file(), modifying_msg_topic_for_subscribers(), select_subscribers_for_topic().

These functions are used to manages message topics.

N.B.: There is some exception to use some parameters: msg_topic.keywords for list parameters and topics_subscriber for subscribers options in the DB table. These parameters are used as string splitted by ',' but to access to each one, use the function tools::get_array_from_splitted_string() (see tools-get-array-from-splitted-string, page) allows to access the enumeration.

is_there_msg_topic()

Tests if some message topic are defined (`msg\_topic` list parameter, see , page).

IN: `self` (+): ref(List)

OUT: 1 - some msg_topic are defined 0 - no msg_topic

is_available_msg_topic()

Checks for a topic if it is available in the list: look foreach `msg\_topic.name` list parameter (see , page).

IN:

1. `self` (+): ref(List)
2. `topic` (+): the name of the requested topic

OUT: `topic` *if it is available* undef

get_available_msg_topic()

Returns an array of available message topics (`msg\_topic.name` list parameter, see , page).

IN: `self` (+): ref(List)

OUT: ref(ARRAY)

is_msg_topic_tagging_required()

Returns if the message must be tagged or not (`msg\_topic\_tagging` list parameter set to 'required', see , page).

IN: `self` (+): ref(List)

OUT: 1 - the message must be tagged 0 - the msg can be no tagged

automatic_tag()

Computes topic(s) (with compute_topic() function) and tags the message (with tag_topic() function) if there are some topics defined.

IN:

1. `self` (+): ref(List)
2. `msg` (+): ref(MIME::Entity)- the message to tag
3. `robot` (+): robot

OUT: list of tagged topic: strings separated by ','. It can be empty. undef

compute_topic()

Computes topic(s) of the message. If the message is in a thread, topic is got from the previous message else topic is got from applying a regexp on the subject and/or the body of the message (`msg\_topic\_keywords\_apply\_on` list parameter, see, page). Regexp is based on `msg\_topic.keywords` list parameters (See , page).

IN:

1. `self` (+): ref(List)
2. `msg` (+): ref(MIME::Entity)- the message to tag

OUT: list of computed topic: strings separated by ','. It can be empty.

tag_topic()

Tags the message by creating its topic information file in the `/home/sympa/spool/topic/` spool. The file contains the topic list and the method used to tag the message. Here is the format:

```
TOPIC topicname,...
METHOD editor|sender|auto
```

IN:

1. `self` (+): ref(List)
2. `msg\_id` (+): string - the message ID of the message to tag
3. `topic\_list` (+): the list of topics (strings splitted by `,`)
4. `method` (+): `auto|editor|sender` - the method used for tagging

OUT: name of the created topic information file (`directory/listname.msg\_id`) undef

load_msg_topic_file()

Searches and loads msg topic file corresponding to the message ID (`directory/listname.msg\_id`). It returns information contained inside.

IN:

1. `self` (+): ref(List)
2. `msg\_id` (+): the message ID
3. `robot` (+): the robot

OUT: undef ref(HASH), keys are:

- `topic`: list of topics (strings separated by `,`)
- `method`: `auto|editor|sender` - the method used for tagging
- `msg\_id`: message ID of the message tagged
- `filename`: name of the file

modifying_msg_topic_for_subscribers()

Deletes topics of subscribers that does not exist anymore and sends a notification to subscribers concerned. (Makes a diff on the `msg\_topic` parameter between the list configuration before modification and a new state by calling the `tools::diff\_on\_arrays()` function, see diff_on_arrays). This function is used by `wwsympa::do\_edit\_list()`.

IN:

1. `self` (+): ref(List) - the list concerned before modification
2. `new\_msg\_topic` (+): ref(ARRAY) - new state of the `msg\_topic` parameters

OUT:

1. `1`: if some subscriber topics have been deleted
2. `0`: else

select_subscribers_for_topic()

Selects subscribers that are subscribed to one or more topic appearing in the topic list incoming when their delivery mode is `mail`, and selects the other subscribers (delivery mode different from `mail`). This function is used by the `List::send\_msg()` function during message diffusion (see Functions for message distribution).

IN:

1. `self` (+): ref(List)
2. `string\_topic` (+): string splitted by `,` - the topic list
3. `subscribers` (+): ref(ARRAY) - list of subscriber emails

OUT: ARRAY - list of selected subscribers

Scenario evaluation

The following function is used to evaluate scenario file `<action>.<parameter\_value>`, where `<action>` corresponds to a configuration parameter for an action and `<parameter\_value>` corresponds to its value.

request_action()

Returns the action to perform for one sender using one authentication method to perform an operation .

IN:

1. `operation` (+): SCALAR - the requested action corresponding to config parameter
2. `auth\_method` (+): `smtp|md5|pgp|smime`

3. `robot` (+): robot
4. `context` (): ref(HASH) - contains value to instantiate scenario variables (hash keys)
5. `debug` (): boolean - if true adds keys `condition` and `auth\_method` to the hash returned.

OUT: undef ref(HASH) with keys:

1. `action`: `do\_it|reject|request\_auth|owner|editor|editorkey|listmaster`
2. `reason`: 'value' if `action == 'reject'` in scenario and if there is `reject(reason='value')` to match a key in `mail\_tt2/authorization\_reject.tt2`. This is used in errors reports (see report.pm)
3. `tt2`: template name if `action == 'reject'` in scenario and there is `reject(tt2='template_name')`.
4. `condition`: the checked condition.
5. `auth\_method`: the checked `auth\_method`

Structure and access to list configuration parameters

List parameters are represented in the list configuration file, in the list object `(list->{'admin'})` and on the web interface. Here are translation and access functions: other (3) (1) (5) CONFIG FILE LIST OBJECT WEB INTERFACE (2) (4) (6)

1. Loading file in memory:

```
List::_load_admin_file(),_load_include_admin_user_file(),_load_list_param()
```

1. Saving list configuration in file:

```
List::_save_admin_file(),_save_list_param()
```

1. Tools to get parameter values:

```
List::get_param_value(),_get_param_value_anywhere(),_get_single_param_value(
)
```

1. Tools to initialize list parameters with defaults:

```
List::_apply_default()
```

1. To present list parameters on the web interface:

```
wwsympa::do_edit_list_request(),_prepare_edit_form(),_prepare_data()
```

1. To get updates on list parameters from the web interface:

```
wwsympa::do_edit_list(),_check_new_value
```

List parameters can be simple or composed in paragraphs, they can be unique or multiple and they can be singlevalued or multivalued. Here are the different kinds of parameters with exemples:

parameters SIMPLE COMPOSED SINGLE singlevalued (a) (b) lang archiv.period multivalued © (d) topics available_user_option.reception MULTIPLE singlevalued (e) (f) include_list owner.email multi

values not defined not defined

Here are these list parameters format in list configuration file in front of Perl representation in memory:

```
    List Configuration FILE    $list->{'admin'}
(a) param value    'scalar'
(b) param
    p1 val1    'HASHscalar'
    p2 val2
(c) param val1,val2,val3    'ARRAY(scalar & split_char)'
(d) param
    p1 val11, val12, val13    'HASHARRAY(scalar & split_char)'
    p2 val21, val22, val23
(e) param val1    'ARRAY(scalar)'
    param val2
(d) param
    p1 val11    'ARRAY(HASHscalar)'
    p2 val12
    param
    p1 val21
    p2 val22
```