SOAP/HTTP API

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: See "Supported functions". Other functions might be implemented in the future. One of the important implementation constraints is to provide services for proxy applications with a correct authorization evaluation process where authentication may differ from classic web methods. The following four cases can be used to access the service:

- The client sends credentials and then requests a service providing a cookie with an id sympa_session.
- The client authenticates the end user providing the sympa_session 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 a 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 their own password, 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 a mail interface or a normal web interface.

The server is running as FastCGI server, receiving the client SOAP requests via a HTTP server (Apache HTTP Server for example).

Supported functions

Note that all functions accessible through the SOAP interface apply the appropriate access control rules, given the user's privileges.

The following functions are currently available through the Sympa SOAP server:

- loginUser email and passwords are checked against Sympa user DB, or another backend.
- casLoginThis function will verify CAS proxy tickets against the CAS server.
authenticateAndRun

Useful for SOAP clients that can't set an HTTP cookie; they can provide both the Sympa session cookie and the requested command in a single call.

authenticateRemoteAppAndRun

Equivalent of the previous command used in a trusted context (see "Trust remote applications").

lists

Provides a list of available lists (authorization scenarios are applied).

complexLists

Same as the previous feature, but provides a complex structure for each list.

info

Provides description informations about a given list.

which

Gets the list of subscription of a given user.

complexWhich

Same as previous command, but provides a complex structure for each list.

ami

Tells if a given user is member of a given list.

review

Lists the members of a given list.

subscribe

Subscribes the current user to a given list.

signoff

Current user is removed from a given list.

addUsed

To add a given user to a given list (admin feature).

del

Removes a given user from a given list (admin feature).

createList

Creates a new mailing list (requires appropriate privileges).

closeList

Closes a given mailing list (admin feature).

Note that when a list parameter is required for a function, you can either provide the list name or the list address. However the domain part of the address will be ignored.

Check the WSDL service description for detailed API information.

Setup

Requirements

- Web interface has to be configured. See "Configure HTTP server" for details.
- SOAP-Lite Perl library.

HTTP server setup

See appropriate section in "Configure HTTP server".

Sympa configuration parameters

- `soap_url` The URL of SOAP/HTTP service itself.
- `wwsympa_url` This is URL prefix of WWSympa service without trailing slash (`/`). WSDL service description is published using a URL `wwsympa_url/wsdl`.

Other configuration files

- `sympa.wsdl` WSDL service description. Default description is placed under `DEFAULTDIR`.
- `trusted_applications.conf` Definitions of trusted SOAP applications. See also below.

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 `\$SYSCONFDIR\`/`domain` directory or in `\$SYSCONFDIR` directory depending on the scope you want for it. 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:

```
 bash $ sympa_soap_client.pl --soap_url=http://web.example.org/sympasoap --service=createList --trusted_application=myTestApp --trusted_application_password='myTestAppPwd' --proxy_vars='USER_EMAIL=userProxy@my.dom.ain' --service_parameters='listA,listSubject,discussion_list,description,myTopic'
```

```
 $ sympa_soap_client.pl --soap_url=http://web.example.org/sympasoap --service=add --trusted_application=myTestApp --trusted_application_password='myTestAppPwd' --proxy_vars='USER_EMAIL=userProxy@my.dom.ain' --service_parameters='listA,someone@some;domain,name'
```

Below is a sample Perl code that does a SOAP procedure call (for a SUBSCRIBE sympa command) using the `trusted_application` feature:

```
 perl use SOAP::Lite;

my $soap = SOAP::Lite->new(); $soap->uri('urn:sympasoap');
$soap->proxy('http://web.exemple.org/sympasoap');

my $response = $soap->authenticateRemoteAppAndRun('myTestApp',
    'myTestAppPwd',
    'USER_EMAIL=userProxy@my.server',
    'subscribe', ['myList@mail.example.org']);
```

S. Santoro wrote its own PHP Trusted Application library for Sympa.

**Client-side programming**

Sympa is distributed with two 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 `sampleClient.php` is found in sample directory of source distribution.

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 module), 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 fetched through WWSympa's dedicated URL, e.g. http://web.example.org/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**

Note:

- This section should be updated to support Apache Axis2. Please consider contributing your work (See CONTRIBUTING).

First, download Apache Axis.

You must add the libraries provided with Apache Axis to your 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:

```
bash $ java org.apache.axis.wsdl.WSDL2Java -av <WSDL URL>
```

For example:

```
bash $ java org.apache.axis.wsdl.WSDL2Java -av http://web.example.org/sympa/wsdl
```

Exemple of screen output during generation of Java files:

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

https://www.sympa.org/
If you need more information or more generated classes (to have the server-side classes or junit testcases for example), you can get a list of switches:

code $ java org.apache.axis.wsdl.WSDL2Java -h

See the documentation for more details.

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:

code 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();

The test command line SOAP client

Sympa distribution includes a simple command line application that allows you to test SOAP request towards your Sympa SOAP server. This script is named `sympa_soap_client.pl` and is located in `$SCRIPTDIR` directory.

The four methods available through the Sympa SOAP server can be tested using this tool. There is no explicit option to tell what access method is used. It is inferred based on what options are provided to the script.

Getting the email associated to a session id

You must use the id of a session actually used at the time you launch the command. It is the value of the "sympa_session" cookie set when accessing to the HTTP interface.

Command line

bash $ sympa_soap_client.pl \ --soap_url=<SOAP server URL> \ --cookie=<cookie identifier>

- `--soap_url` The URL to your Sympa SOAP server.
- `--cookie` The value of the "sympa_session" cookie set when accessing to the Sympa web interface.

Expected output

`code error : get_email_cookie cookie : 65354224256806
getEmailUserByCookie.... 0

'mail@renater.fr'
`

Sympa mailing list server - https://www.sympa.org/
Using the Sympa SOAP functions with the command line tool

It is done by calling the script and providing two kind of arguments:

- the argument required by the service usage: SOAP URL, service name and service parameters,
- the arguments allowing to authenticate the user requesting the service.

Authentication using an HTTP session cookie

Actually, providing the HTTP cookie to a command line sums up in providing a session id, i.e. a simple number. You must use the value of a session cookie actually used at the time you launch the command. It is the "sympa_session" cookie set when accessing to the Sympa web interface.

```
bash $ sympa_soap_client.pl --soap_url=<SOAP server URL> \  --service=<a sympa service> \  --service_parameters=<value1,value2,value3> \  --session_id=<cookie identifier>
```

The options used are:

- `--soap_url` The URL to your Sympa SOAP server.
- `--service` The requested SOAP service. See below;
- `--service_parameters` The parameters needed to use the service. They must be provided as a comma separated list, without spaces. See below.
- `--session_id` The value of the "sympa_session" cookie set when accessing to the Sympa web interface.

Authentication using a user name and password

```
bash $ sympa_soap_client.pl --soap_url=<SOAP server URL> \  --service=<a sympa service> \  --service_parameters=<value1,value2,value3> \  --user_email=<email> \  --user_password=<password>
```

The options used are:

- `--soap_url` The URL to your Sympa SOAP server.
- `--service` The requested SOAP service. See below.
- `--service_parameters` The parameters needed to use the service. They must be provided as a comma separated list, without spaces. See below.
- `--user_email` The email of the user requesting the service.
- `--user_password` The password of this user.

Access through a trusted application

```
bash $ sympa_soap_client.pl --soap_url=<SOAP server URL> \  --service=<a sympa service> \  --service_parameters=<value1,value2,value3> \  --cookie=<cookie identifier> \  --trusted_application=<app name> \  --trusted_application_password=<password> \  --proxy_vars=<id=value,id2=value2>
```

The options used are:

- `--soap_url` The URL to your Sympa SOAP server.
- `--service` The requested SOAP service. See below.
- `--service_parameters` The parameters needed to use the service. They must be provided
as a comma separated list, without spaces. See below.

- `--cookie` The value of the "sympa_session" cookie set when accessing to the Sympa web interface.
- `--trusted_application` The trusted application name as defined in trusted_applications.conf.
- `--trusted_application_password` The password of the trusted application as defined in trusted_applications.conf.
- `--proxy_vars` The proxy vars of the trusted application as defined in trusted_applications.conf. This is a comma-separated list of values. For example, if you have defined in trusted_applications.conf the following variables:
  proxy_for_variables USER_EMAIL,remote_host, then you will use it this way in the proxy_vars option: --proxy_vars=USER_EMAIL=user.email@domain.tld,remote_host=remote.host.domain.tld.

**Sympa SOAP services and the command line tool**

This is a description of how to use the Sympa SOAP services using the command line tool. The parameters are given in the same order they must be found in the command tool option service_parameters. They must be provided as a comma separated list, without spaces. Don't forget to escape characters that would break the command line, such as spaces, exclamation marks and so on.

---

**Note:**

- If the list of parameters is: then the service_parameters option will look like: --service_parameters=mylist,mail@my.domain
  - list name
  - user email

---

**login**

No object here: this is the service used to log when the command tool uses a username and password.

**casLogin**

No object here.

**authenticateAndRun**

No object here: this the service used by the command line tool to call the other services, when authentication is done through session id or user name + password.

**authenticateRemoteAppAndRun**

No object here: this the service used by the command line tool to call the other services, when testing trusted applications.
lists

The parameters are optional.

Parameters:

- topic: the topic of the lists to return
- subtopic: the subtopic of this topic

Output example:

```
` code lists.... 0

'homepage=http://domain.tld/sympa/info/amietestdv01;subject=Amical;listAddress=amietestdv01@domain.tld'

1

'homepage=http://domain.tld/sympa/info/archeologie;subject=Liste sur l'archéologie;listAddress=archeologie@domain.tld'

2

'homepage=http://domain.tld/sympa/info/blackmambo;subject=A black mambo;listAddress=blackmambo@domain.tld'

3

'homepage=http://domain.tld/sympa/info/bluemambo;subject=Another mambo. This one is blue.;listAddress=bluemambo@domain.tld'
```

complexLists

The parameters are optional.

Parameters:

- topic: the topic of the lists to return
- subtopic: the subtopic of this topic

Output example:

```
` code AuthenticateAndRun complexLists.... 0

_ homepage_ 'http://domain.tld/sympa-dv/info/amietestdv01'
_ listAddress_ 'amietestdv01@domain.tld'
_ subject_ 'Amical'
```
info

Parameters:

- listname (mandatory): the name of the list for which info are requested

Output example:

code

which

All arguments are mandatory (at least with an empty value).

Parameters:

- no parameters

Output example:

`code which.... 0

`isOwner=1;homepage=http://domain.tld/sympa/info/amietestdv01;subject=Amical

Sympa mailing list server - https://www.sympa.org/
complexWhich

All arguments are mandatory (at least with an empty value).

Parameters:

- no parameters

Output example:

`code complexWhich.... 0

```bash
    _homepage_    
    'http://dev-sympa.renater.fr/sympa-dv/info/redmambo'
    _isEditor_    
    '0'
    _isOwner_     
    '1'
    _isSubscriber_
    '0'
    _listAddress_  
    'redmambo@dev-sympa.renater.fr'
    _subject_     
    'Amical'
```

1
amI

Parameters:

- list name (mandatory): the name of the list for which the function is tested;
- function (mandatory): the function the existence of which we will test. The allowed values are: subscriber, owner and editor;
- user (mandatory): the email address of the user for whom we want to know if she has the function indicated in the target list.

Sympa mailing list server - https://www.sympa.org/
Output example:

```
code param: blackmambo param: owner param: david.verdin@renater.fr Using Session_id 48339436597794
AuthenticateAndRun amI.... 0

'1'
```

**review**

Parameters:

- the name of the list for which we want the subscribers list (mandatory).

Output example:

```
'code review.... 0

'mail1@renater.fr'
1
'mail2@renater.fr'
2
'mail3@renater.fr'
```

**subscribe**

Parameters:

- list name (mandatory)

Output example:

```
'code subscribe.... 0

'1'
```

**signoff**

Parameters:

- list name (mandatory)
Output example:
` code signoff.... 0
`` 1`
``

**add**

Parameters:

- listname (mandatory): the name of the list we want to subscribe the mail address to;
- email (mandatory): the email to subscribe to the list;
- gecos: the name under which this email will be subscribed (for example: “John Doe”);
- quiet: if set to '1', the user doesn't receive a subscription notification

Output example:
` code add.... 0
```````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````
`code param: orangemambo param: Dude ! param: hotline param: La liste verte param: computing
Using Session_id 4860001445687

AuthenticateAndRun createList.... 0

' `1`

`

closeList

Parameters:

- the name of the list to close (mandatory).

Output example:

`code param: orangemambo Using Session_id 4860001445687
AuthenticateAndRun closeList.... 0

' `1`

' `