perl Module

Bastian Friedrich
Collax GmbH

Edited by
Bastian Friedrich

perl Module

Edited by Bastian Friedrich and Bastian Friedrich

Revision History

Revision $Revision: 1678 $ $Date: 2007-02-21 17:01:06 +0100 (Mi, 21 Feb 2007) $

Table of Contents

List of Examples

Chapter 1. User's Guide

1.1. Overview

The time needed when writing a new OpenSER module unfortunately is quite high, while the options provided by the configuration file are limited to the features implemented in the modules.

With this Perl module, you can easily implement your own OpenSER extensions in Perl. This allows for simple access to the full world of CPAN modules. SIP URI rewriting could be implemented based on regular expressions; accessing arbitrary data backends, e.g. LDAP or Berkeley DB files, is now extremely simple.

1.2. Installing the module

This Perl module is loaded in openser.cfg (just like all the other modules) with loadmodule("/path/to/perl.so");.

For the Perl module to compile, you need a reasonably recent version of perl (tested with 5.8.8) linked dynamically. It is strongly advised to use a threaded version. The default binary packages from your favorite Linux distribution should work fine.

Cross compilation is supported by the Makefile. You need to set the environment variables PERLLDOPTS, PERLCCOPTS and TYPEMAP to values similar to the output of

```
PERLLDOPTS: perl -MExtUtils::Embed -e ldopts
PERLCCOPTS: perl -MExtUtils::Embed -e ccopts
TYPEMAP:    echo "`perl -MConfig -e 'print $Config{installprivlib}'`/ExtUtils/typemap"
```

The exact position of your (precompiled!) perl libraries depends on the setup of your environment.

1.3. Using the module

The Perl module has two interfaces: The perl side, and the OpenSER side. Once a Perl function is defined and loaded via the module parameters (see below), it may be called in OpenSER's configuration at an arbitary point. E.g., you could write a function "ldap_alias" in Perl, and then execute

```
...
if (perl_exec("ldap_alias")) {
 ...
}
```

. . .

just as you would have done with the current alias_db module.

The functions you can use are listed in the L<"Exported Functions"> section below.

On the Perl side, there are a number of functions that let you read and modify the current SIP message, such as the RURI or the message flags. An introduction to the Perl interface and the full reference documentation can be found below.

1.4. Dependencies

1.4.1. OpenSER Modules

The following modules must be loaded before this module:

- The "sl" module is needed for sending replies upon fatal errors. All other modules can be accessed from the Perl module, though.

1.4.2. External Libraries or Applications

The following libraries or applications must be installed before running OpenSER with this module loaded:

- *Perl 5.8.x or later*

Additionally, a number of perl modules should be installed. The OpenSER::LDAPUtils package relies on Net::LDAP to be installed. One of the sample scripts needs IPC::Shareable

This module has been developed and tested with Perl 5.8.8, but should work with any 5.8.x release. Compilation is possible with 5.6.x, but its behavior is unsupported. Earlier versions do not work.

On current Debian systems, at least the following packages should be installed:

- perl
- perl-base
- perl-modules
- libperl5.8
- libperl-dev

- libnet-ldap-perl
- libipc-shareable-perl

It was reported that other Debian-style distributions (such as Ubuntu) need the same packages.

On SuSE systems, at least the following packages should be installed:

- perl
- perl-ldap
- IPC::Shareable perl module from CPAN

Although SuSE delivers a lot of perl modules, others may have to be fetched from CPAN. Consider using the program "cpan2rpm" - which, in turn, is available on CPAN. It creates RPM files from CPAN.

1.5. Exported Parameters

1.5.1. `filename` (string)

This is the file name of your script. This may be set once only, but it may include an arbitary number of functions and "use" as many Perl module as necessary.

May not be empty!

Example 1-1. Set `filename` parameter

```
...
modparam("perl", "filename", "/home/john/openser/myperl.pl")
...
```

1.5.2. `modpath` (string)

The path to the Perl modules included (OpenSER.pm et.al). It is not absolutely crucial to set this path, as you *may* install the Modules in Perl's standard path, or update the "%INC" variable from within your script. Using this module parameter is the standard behavior, though.

Example 1-2. Set `modpath` parameter

```
...
modparam("perl", "modpath", "/usr/local/lib/openser/perl/")
...
```

1.6. Exported Functions

1.6.1. perl_exec_simple(func, [param])

Calls a perl function *without* passing it the current SIP message. May be used for very simple simple requests that do not have to fiddle with the message themselves, but rather return information values about the environment.

The first parameter is the function to be called. An arbitrary string may optionally be passed as a parameter.

This function can be used from REQUEST_ROUTE, FAILURE_ROUTE, ONREPLY_ROUTE and BRANCH_ROUTE.

Example 1-3. perl_exec_simple() usage

```
...
if (method=="INVITE") {
 perl_exec_simple("dosomething", "on invite messages");
};
...
```

1.6.2. perl_exec(func, [param])

Calls a perl function *with* passing it the current SIP message. The SIP message is reflected by a Perl module that gives you access to the information in the current SIP message (OpenSER::Message).

The first parameter is the function to be called. An arbitrary string may be passed as a parameter.

This function can be used from REQUEST_ROUTE, FAILURE_ROUTE, ONREPLY_ROUTE and BRANCH_ROUTE.

Example 1-4. perl_exec() usage

```
...
if (perl_exec("ldapalias")) {
 ...
};
...
```

Chapter 2. Developer's Guide

The module does not provide any API to use in other OpenSER modules.

Chapter 3. OpenSER Perl API

3.1. OpenSER

This module provides access to a limited number of OpenSER core functions. As the most interesting functions deal with SIP messages, they are located in the OpenSER::Message class below.

3.1.1. log(level,message)

Logs the message with OpenSER's logging facility. The logging level is one of the following:

* L_ALERT
* L_CRIT
* L_ERR
* L_WARN
* L_NOTICE
* L_INFO
* L_DBG

Please note that this method is *NOT* automatically exported, as it collides with the perl function log (which calculates the logarithm). Either explicitly import the function (via `use OpenSER qw ( log );`), or call it with its full name:

```
OpenSER::log(L_INFO, "foobar");
```

3.2. OpenSER::Message

This package provides access functions for an OpenSER `sip_msg` structure and its sub-components. Through its means it is possible to fully configure alternative routing decisions.

3.2.1. getType()

Returns one of the constants SIP_REQUEST, SIP_REPLY, SIP_INVALID stating the type of the current message.

3.2.2. getStatus()

Returns the status code of the current Reply message. This function is invalid in Request context!

3.2.3. getReason()

Returns the reason of the current Reply message. This function is invalid in Request context!

3.2.4. getVersion()

Returns the version string of the current SIP message.

3.2.5. getRURI()

This function returns the recipient URI of the present SIP message:

```
my $ruri = $m->getRURI();
```

getRURI returns a string. See "getParsedRURI()" below how to receive a parsed structure.

This function is valid in request messages only.

3.2.6. getMethod()

Returns the current method, such as `INVITE`, `REGISTER`, `ACK` and so on.

```
my $method = $m->getMethod();
```

This function is valid in request messages only.

3.2.7. getFullHeader()

Returns the full message header as present in the current message. You might use this header to further work with it with your favorite MIME package.

```
my $hdr = $m->getFullHeader();
```

3.2.8. getBody()

Returns the message body.

3.2.9. getMessage()

Returns the whole message including headers and body.

3.2.10. getHeader(name)

Returns the body of the first message header with this name.

```
print $m->getHeader("To");
```

```
"John" <sip:john@doe.example>
```

3.2.11. getHeaderNames()

Returns an array of all header names. Duplicates possible!

3.2.12. moduleFunction(func,string1,string2)

Search for an arbitrary function in module exports and call it with the parameters self, string1, string2.

`string1` and/or `string2` may be omitted.

As this function provides access to the functions that are exported to the OpenSER configuration file, it is autoloaded for unknown functions. Instead of writing

```
$m->moduleFunction("sl_send_reply", "500", "Internal Error");
$m->moduleFunction("xlog", "L_INFO", "foo");
```

you may as well write

```
$m->sl_send_reply("500", "Internal Error");
$m->xlog("L_INFO", "foo");
```

WARNING

In OpenSER 1.2, only a limited subset of module functions is available. This restriction will be removed in a later version.

Here is a list of functions that are expected to be working (not claiming completeness):

* alias_db_lookup
* consume_credentials
* is_rpid_user_e164
* append_rpid_hf
* bind_auth
* avp_print
* cpl_process_register
* cpl_process_register_norpl
* load_dlg
* ds_next_dst
* ds_next_domain
* ds_mark_dst
* ds_mark_dst
* is_from_local
* is_uri_host_local
* dp_can_connect
* dp_apply_policy
* enum_query (without parameters)
* enum_fquery (without parameters)
* is_from_user_enum (without parameters)
* i_enum_query (without parameters)
* imc_manager
* jab_* (all functions from the jabber module)
* load_gws (without parameters)
* next_gw
* from_gw (without parameters)
* to_gw (without parameters)
* load_contacts
* next_contacts
* sdp_mangle_ip
* sdp_mangle_port
* encode_contact
* decode_contact
* decode_contact_header
* fix_contact
* use_media_proxy
* end_media_session
* m_store
* m_dump
* fix_nated_contact
* unforce_rtp_proxy
* force_rtp_proxy
* fix_nated_register
* add_rcv_param
* options_reply
* checkospheader
* validateospheader

```
* requestosprouting
* checkosproute
* prepareosproute
* prepareallosproutes
* checkcallingtranslation
* reportospusage
* mangle_pidf
* mangle_message_cpim
* add_path (without parameters)
* add_path_received (without parameters)
* prefix2domain
* allow_routing (without parameters)
* allow_trusted
* pike_check_req
* handle_publish
* handle_subscribe
* stored_pres_info
* bind_pua
* send_publish
* send_subscribe
* pua_set_publish
* loose_route
* record_route
* load_rr
* sip_trace
* sl_reply_error
* sms_send_msg
* sd_lookup
* sstCheckMin
* append_time
* has_body (without parameters)
* is_peer_verified
* t_newtran
* t_release
* t_relay (without parameters)
* t_flush_flags
* t_check_trans
* t_was_cancelled
* uac_restore_from
* uac_auth
* has_totag
* tel2sip
* check_to
* check_from
* radius_does_uri_exist
* ul_* (All functions exported by the usrloc module for user access)
* xmpp_send_message
```

3.2.13. log(level,message) (deprecated type)

Logs the message with OpenSER's logging facility. The logging level is one of the following:

* L_ALERT
* L_CRIT
* L_ERR
* L_WARN
* L_NOTICE
* L_INFO
* L_DBG

The logging function should be accessed via the OpenSER module variant. This one, located in OpenSER::Message, is deprecated.

3.2.14. rewrite_ruri(newruri)

Sets a new destination (recipient) URI. Useful for rerouting the current message/call.

```
if ($m->getRURI() =~ m/\@somedomain.net/) {
  $m->rewrite_ruri("sip:dispatcher\@organization.net");
}
```

3.2.15. setFlag(flag)

Sets a message flag. The constants as known from the C API may be used, when Constants.pm is included.

3.2.16. resetFlag(flag)

Resets a message flag.

3.2.17. isFlagSet(flag)

Returns whether a message flag is set or not.

3.2.18. pseudoVar(string)

Returns a new string where all pseudo variables are substituted by their values. Can be used to receive the values of single variables, too.

Please remember that you need to escape the '$' sign in perl strings!

3.2.19. append_branch(branch,qval)

Append a branch to current message.

3.2.20. serialize_branches(clean_before)

Serialize branches.

3.2.21. next_branches()

Next branches.

3.2.22. getParsedRURI()

Returns the current destination URI as an OpenSER::URI object.

3.3. OpenSER::URI

This package provides functions for access to sip_uri structures.

3.3.1. user()

Returns the user part of this URI.

3.3.2. host()

Returns the host part of this URI.

3.3.3. passwd()

Returns the passwd part of this URI.

3.3.4. port()

Returns the port part of this URI.

3.3.5. params()

Returns the params part of this URI.

3.3.6. headers()

Returns the headers part of this URI.

3.3.7. transport()

Returns the transport part of this URI.

3.3.8. ttl()

Returns the ttl part of this URI.

3.3.9. user_param()

Returns the user_param part of this URI.

3.3.10. maddr()

Returns the maddr part of this URI.

3.3.11. method()

Returns the method part of this URI.

3.3.12. lr()

Returns the lr part of this URI.

3.3.13. r2()

Returns the r2 part of this URI.

3.3.14. transport_val()

Returns the transport_val part of this URI.

3.3.15. ttl_val()

Returns the ttl_val part of this URI.

3.3.16. user_param_val()

Returns the user_param_val part of this URI.

3.3.17. maddr_val()

Returns the maddr_val part of this URI.

3.3.18. method_val()

Returns the method_val part of this URI.

3.3.19. lr_val()

Returns the lr_val part of this URI.

3.3.20. r2_val()

Returns the r2_val part of this URI.

3.4. OpenSER::AVP

This package provides access functions for OpenSER's AVPs. These variables can be created, evaluated, modified and removed through this package.

Please note that these functions do NOT support the notation used in the configuration file, but directly work on strings or numbers. See documentation of add method below.

3.4.1. add(name,val)

Add an AVP.

Add an OpenSER AVP to its environment. name and val may both be integers or strings; this function will try to guess what is correct. Please note that

```
OpenSER::AVP::add("10", "10")
```

is something different than

```
OpenSER::AVP::add(10, 10)
```

due to this evaluation: The first will create _string_ AVPs with the name 10, while the latter will create a numerical AVP.

You can modify/overwrite AVPs with this function.

3.4.2. get(name)

get an OpenSER AVP:

```
my $numavp = OpenSER::AVP::get(5);
my $stravp = OpenSER::AVP::get("foo");
```

3.4.3. destroy(name)

Destroy an AVP.

```
OpenSER::AVP::destroy(5);
OpenSER::AVP::destroy("foo");
```

3.5. OpenSER::Utils::PhoneNumbers

OpenSER::Utils::PhoneNumbers - Functions for canonical forms of phone numbers.

```
use OpenSER::Utils::PhoneNumber

my $phonenumber = new OpenSER::Utils::PhoneNumber(
     publicAccessPrefix => "0",
     internationalPrefix => "+",
     longDistancePrefix => "0",
     areaCode => "761",
     pbxCode => "456842",
     countryCode => "49"
   );

$canonical = canonicalForm("07612034567");
$number    = dialNumber("+497612034567");
```

A telphone number starting with a plus sign and containing all dial prefixes is in canonical form. This is usally not the number to dial at any location, so the dialing number depends on the context of the user/system.

The idea to canonicalize numbers were taken from hylafax.

Example: +497614514829 is the canonical form of my phone number, 829 is the number to dial at Pyramid, 4514829 is the dialing number from Freiburg are and so on.

To canonicalize any number, we strip off any dial prefix we find and then add the prefixes for the location. So, when the user enters the number 04514829 in context pyramid, we remove the publicAccessPrefix (at Pyramid this is 0) and the pbxPrefix (4514 here). The result is 829. Then we add all the general dial prefixes - 49 (country) 761 (area) 4514 (pbx) and 829, the number itself => +497614514829

To get the dialing number from a canonical phone number, we substract all general prefixes until we have something

As said before, the interpretation of a phone number depends on the context of the location. For the functions in this package, the context is created through the `new` operator.

The following fields should be set:

```
'longDistancePrefix'
'areaCode'
'pbxCode'
'internationalPrefix'
'publicAccessPrefix'
'countryCode'
```

This module exports the following functions when `use`ed:

3.5.1. new(publicAccessPrefix,internationalPrefix,longDistancePrefix,country

The new operator returns an object of this type and sets its locational context according to the passed parameters. See OpenSER::Utils::PhoneNumbers above.

3.5.2. canonicalForm(number [, context])

Convert a phone number (given as first argument) into its canonical form. When no context is passed in as the second argument, the default context from the systems configuration file is used.

3.5.3. dialNumber(number [, context])

Convert a canonical phone number (given in the first argument) into a number to to dial. WHen no context is given in the second argument, a default context from the systems configuration is used.

3.6. OpenSER::LDAPUtils::LDAPConf

OpenSER::LDAPUtils::LDAPConf - Read openldap config from standard config files.

```
use OpenSER::LDAPUtils::LDAPConf;
my $conf = new OpenSER::LDAPUtils::LDAPConf();
```

This module may be used to retrieve the global LDAP configuration as used by other LDAP software, such as `nsswitch.ldap` and `pam-ldap`. The configuration is usualy stored in `/etc/openldap/ldap.conf`

When used from an account with sufficient privilegs (e.g. root), the ldap manager passwort is also retrieved.

3.6.1. Constructor new()

Returns a new, initialized `OpenSER::LDAPUtils::LDAPConf` object.

3.6.2. Method base()

Returns the servers base-dn to use when doing queries.

3.6.3. Method host()

Returns the ldap host to contact.

3.6.4. Method port()

Returns the ldap servers port.

3.6.5. Method uri()

Returns an uri to contact the ldap server. When there is no ldap_uri in the configuration file, an `ldap:` uri is constucted from host and port.

3.6.6. Method rootbindpw()

Returns the ldap "root" password.

Note that the `rootbindpw` is only available when the current account has sufficient privilegs to access `/etc/openldap/ldap.secret`.

3.6.7. Method rootbinddn()

Returns the DN to use for "root"-access to the ldap server.

3.6.8. Method binddn()

Returns the DN to use for authentication to the ldap server. When no bind dn has been specified in the configuration file, returns the `rootbinddn`.

3.6.9. Method bindpw()

Returns the password to use for authentication to the ldap server. When no bind password has been specified, returns the `rootbindpw` if any.

3.7. OpenSER::LDAPUtils::LDAPConnection

OpenSER::LDAPUtils::LDAPConnection - Perl module to perform simple LDAP queries.

OO-Style interface:

```
use OpenSER::LDAPUtils::LDAPConnection;
my $ldap = new OpenSER::LDAPUtils::LDAPConnection;
my @rows = $ldap-search("uid=andi","ou=people,ou=coreworks,ou=de");
```

Procedural interface:

```
use OpenSER::LDAPUtils::LDAPConnection;
my @rows = $ldap->search(
      new OpenSER::LDAPUtils::LDAPConfig(), "uid=andi","ou=people,ou=coreworks,ou=de");
```

This perl module offers a somewhat simplified interface to the `Net::LDAP` functionality. It is intended for cases where just a few attributes should be retrieved without the overhead of the full featured `Net::LDAP`.

3.7.1. Constructor new([config, [authenticated]])

Set up a new LDAP connection.

The first argument, when given, should be a hash reference pointing to to the connection parameters, possibly an `OpenSER::LDAPUtils::LDAPConfig` object. This argument may be `undef` in which case a new (default) `OpenSER::LDAPUtils::LDAPConfig` object is used.

When the optional second argument is a true value, the connection will be authenticated. Otherwise an anonymous bind is done.

On success, a new `LDAPConnection` object is returned, otherwise the result is `undef`.

3.7.2. Function/Method search(conf, filter, base, [requested_attributes ...])

perform an ldap search, return the dn of the first matching directory entry, unless a specific attribute has been requested, in wich case the values(s) fot this attribute are returned.

When the first argument (conf) is a `OpenSER::LDAPUtils::LDAPConnection`, it will be used to perform the queries. You can pass the first argument implicitly by using the "method" syntax.

Otherwise the `conf` argument should be a reference to a hash containing the connection setup parameters as contained in a `OpenSER::LDAPUtils::LDAPConf` object. In this mode, the `OpenSER::LDAPUtils::LDAPConnection` from previous queries will be reused.

3.7.2.1. Arguments:

conf

: configuration object, used to find host,port,suffix and use_ldap_checks

filter

: ldap search filter, eg '(mail=some@domain)'

base

: search base for this query. If undef use default suffix, concat base with default suffix if the last char is a ','

requested_attributes

: retrieve the given attributes instead of the dn from the ldap directory.

3.7.2.2. Result:

Without any specific `requested_attributes`, return the dn of all matching entries in the LDAP directory.

When some `requested_attributes` are given, return an array with those attibutes. When multiple entries match the query, the attribute lists are concatenated.

3.8. OpenSER::Constants

This package provides a number of constants taken from enums and defines of OpenSER header files. Unfortunately, there is no mechanism for updating the constants automatically, so check the values if you are in doubt.

Chapter 4. Perl samples

4.1. sample directory

There are a number of example scripts in the “samples/”. They are documented well. Read them, it will explain a lot to you :)

If you want to use any of these scripts directly in your implementation, you can use Perl’s “require” mechanism to import them (just remember that you need to use quotes when require’ing .pl files).

4.1.1. Script descriptions

The included sample scripts are described below:

4.1.1.1. branches.pl

The minimal function in branches.pl demonstrates that you can access the "append_branch" function from within perl, just as you would have done from your normal configuration file. You’ll find documentation on the concepts of branching in the OpenSER documentation.

4.1.1.2. firstline.pl

Message’s first_line structure may be evaluated. Message can be either of SIP_REQUEST or SIP_REPLY. Depending on that, different information can be received. This script demonstrates these functions.

4.1.1.3. flags.pl

The perl module provides access to OpenSER’s flagging mechanism. The flag names available for OpenSER modules are made available through the OpenSER::Constants package, so you can flag messages as "green", "magenta" etc.

The first function, setflag, demonstrates how the "green" flag is set. In the second function, readflag, the "green" and "magenta" flags are evaluated.

4.1.1.4. functions.pl

This sample script demonstrates different things related to calling functions from within perl, and the different types of functions you can offer for OpenSER access.

“exportedfuncs” simply demonstrates that you can use the moduleFunction method to call functions offered by other modules. The results are equivalent to calling these functions from your config file. In the demonstrated case, telephone calls with a destination number beginning with 555... are rejected with an internal server error. Other destination addresses are passed to the alias_db module.

Please note that the moduleFunction method is not fully available in OpenSER 1.2. See the method’s documentation for details.

“paramfunc” shows that you can pass arbitrary strings to perl functions. Do with them whatever you want :)

“autotest” demonstrates that unknown functions in OpenSER::Message objects are automatically transformed into calls to module functions.

The “diefunc”s show that dying perl scripts - by "manual" dying, or because of script errors - are handled by the OpenSER package. The error message is logged through OpenSER’s logging mechanism. Please note that this only works correctly if you do NOT overwrite the default die handler. Oh, yes, that works for warnings, too.

4.1.1.5. headers.pl

Header extraction is among the most crucial functionalities while processing SIP messages. This sample script demonstrates access to header names and values within two sample functions.

“headernames” extracts all header names and logs their names.

“someheaders” logs the contents of the two headers, “To” and “WWW-Contact”. As you can see, headers that occur more than once are retrieved as an array, which may be accessed by Perl’s array accessing methods.

4.1.1.6. logging.pl

For debugging purposes, you probably want to write messages to the syslog. The “logdemo” shows three ways to access the OpenSER log function: it is available through the OpenSER class as well as through the OpenSER::Message class.

Remember that you can use exported functions from other modules. You may thus as well use the “xlog” module and it’s xlog function.

The L_INFO, L_DBG, L_ERR, L_CRIT... constants are available through the OpenSER::Constants package.

4.1.1.7. messagedump.pl

This script demonstrates how to access the whole message header of the current message. Please note that modifications on the message made by earlier function calls in your configuration script may NOT be reflected in this dump.

4.1.1.8. persistence.pl

When processing SIP messages, you may want to use persistent data across multiple calls to your Perl functions. Your first option is to use global variables in your script. Unfortunately, these globals are not visible from the mulitple instances of OpenSER. You may want to use a mechanism such as the IPC::Shareable shared memory access package to correct this.

4.1.1.9. phonenumbers.pl

The OpenSER::Utils::PhoneNumbers package provides two methods for the transformation of local to canonical telephone numbers, and vice versa. This script demonstrates it’s use.

4.1.1.10. pseudovars.pl

This script demonstrates the Perl module’s “pseudoVar” method. It may be used to retrieve the values of current pseudo variables.

You might notice that there is no particular function for setting pseudo variables; you may use the exported functions from the avpops module, though.

Chapter 5. Frequently Asked Questions

1. Where can I find more about OpenSER?

Take a look at http://openser.org/.

2. Where can I post a question about this module?

First at all check if your question was already answered on one of our mailing lists:

- User Mailing List - http://openser.org/cgi-bin/mailman/listinfo/users
- Developer Mailing List - http://openser.org/cgi-bin/mailman/listinfo/devel

E-mails regarding any stable OpenSER release should be sent to <`users@openser.org`> and e-mails regarding development versions should be sent to <`devel@openser.org`>.

If you want to keep the mail private, send it to <`team@openser.org`>.

3. How can I report a bug?

Please follow the guidelines provided at: http://sourceforge.net/tracker/?group_id=139143.