Perl API Reference for Linux
Table of Contents

Perl API for Linux Reference

1. Introduction

This document details specific information you will need to develop an SRS client for the Linux operating system using Perl. It is assumed at this point that you have read the Technical Overview and Developer's Guide documents. As you develop your SRS client, you will want to have a copy of the Developer's Guide at hand since it details all the legal input and output values for each SRS command. We will not duplicate that information here since it is common to all API versions. Rather, we will focus on issues and features specific to the Perl API for Linux.

2. Development Tasks

The development of your SRS client will at a minimum involve the following steps:

- Download and uncompress the Perl API Toolkit for Linux
- Download, install and configure GnuPG, the Gnu Privacy Guard (as documented in the Developer's Guide, section 5.1)
- Download and install required 3rd party Perl modules and libraries.
- Customize the API module with your configuration information (e.g., your partner ID, your cryptographic key name and password, etc.)
- Create your client application/website.
- Test your client.
- Run the certification tests.
- After passing certification, reconfigure the API module to use the SRS production (live) server.

This Reference, along with supplemental material from the Developer's Guide, provides detailed instructions for completing all the above steps, except for the creation of your custom client. We do provide many code examples that you can use as a base for your own development efforts.

3. The API Toolkit

If you have not done so already, download the Perl API Toolkit for Linux from the Developer's Library section of our website:

`http://www.srsplus.com/en/srsplus/partners_dev_library.shtml`

Uncompress the archive contents to a directory of your choosing.

3.1 Organization

The Toolkit is organized into the following directory structure:

`Toolkit`	Contains `readme` and/or `releasenotes` files. Read these files with a text viewer for the latest information about the API.
`Toolkit/docs`	Contains a copy of this document and the Developer's Guide.
`Toolkit/keys`	Contains a keyblock file called `SRS.KEY` containing the SRS public keys for importing into your GnuPG database with the "`gpg --import SRS.KEY`" command.
`Toolkit/examples`	Contains an example script, `certify.pl`, that doubles as the certification script.

Toolkit/source	Contains the single API module, DotSRS_Client.pm.

Be certain to copy the DotSRS_Client.pm file to a directory that is reachable by your scripts (i.e., in an accessible path).

3.2 Dependencies

The Perl API for Linux requires additional components that are not part of the Toolkit download. All of these 3rd party components must be installed before using the API module. The required components are detailed in the following sections.

3.2.1 Perl 5.005

It is assumed that you are using a distribution of Linux that includes an appropriate Perl interpreter installation. The minimal version number supported by the API is 5.005.

3.2.2 OpenSSL

The Perl API for Linux uses modules that depend on the OpenSSL library to perform SSL operations. OpenSSL is available for download at http://www.openssl.org/. The download contains instructions for configuring and installing the OpenSSL library. We also provide a HOW-TO document on our website at:

http://www.srsplus.com/en/srsplus/partners_faq_tech.shtml.

3.2.3 GnuPG

Follow the instructions in section 5.1 of the Developer's Guide for installing and configuring GnuPG. In particular, you need to create your public/private key pair and submit your public key to registry@srsplus.com as soon as practicable. After your key submission is processed, you will be sent back your partner ID, which is a vital piece of information. You will need your partner ID for configuring the API. Without the partner ID, you will not be able to communicate with the SRS servers.

3.2.4 Perl Modules

The DotSRS_Client module is dependent on a number of 3rd party Perl modules, all available at http://www.cpan.org/, the Comprehensive Perl Archive Network. The required modules are:

- Crypt-SSLeay-0.16
- GnuPG-Interface-0.09
- libwww-perl-5.48
- URI-1.06
- Class-MethodMaker-0.95

Version numbers should be interpreted as minimum version numbers. Some of these modules may already be installed on your machine depending on your Linux distribution.

3.2.4.1 Installing Perl Modules

As a Perl developer, you should already be familiar with the two basic methods of installing Perl modules: 1) Using the CPAN module, and 2) Manual installation. The subject of module procurement and installation is beyond the scope of this guide, but if you need a refresher course, we provide a HOW-TO document on our website at:

http://www.srsplus.com/en/srsplus/partners_faq_tech.shtml.

NOTE: Each module upon which the API is dependant may have other internal dependencies. Using the CPAN module will automatically handle these for you. If you choose to manually

install modules, remember that you may encounter these dependencies and will have to handle them yourself.

4. Configuration

The API requires information about your public/private keypair and your partner ID in order for API calls to succeed. In addition, the API module needs to know where your trust database is located (i.e., where GnuPG stores its keys).

4.1 Partner Configuration

Before you run any Perl scripts, you will need to customize the partner information in the API module file `DotSrs_Client.pm`. All of the scalars you will need to modify are clearly identified near the top of the file. Specifically, you will need these four pieces of information:

- The email name used when creating your public/private keypair in GnuPG.
- The passphrase (i.e., password) for your private key.
- Your partner ID, as given to you by SRSplus
- The path to GnuPG's trust database. This is the value to which you set the `GNUPGHOME` environment variable when you installed GnuPG.

Edit the file `DotSrs_Client.pm` and make the following changes:

- Enter the email name for your keypair as the value for `$registrar_email`
- Enter the password for your private key as the value for `$passphase`
- Enter your partner ID as the value for `$registrar_id`
- Enter the path specified in the `GNUPGHOME` environment variable for `$gpg_dir`

4.2 Server Configuration

The scalar `$test_mode` determines which server environment the API will use. When set to a non-zero value, `$test_mode` routes all traffic to the test SRS server. If `$test_mode` is zero, all traffic will be sent to the production SRS server. Section 6.3 below details the use of this value as it relates to switching from the test environment to the production environment *after* passing the certification tests. For now, you should leave `$test_mode` equal to 1.

5. Method Reference

Each SRS command has a corresponding method in the `DotSRS_Client` module. You will need to consult the SRS Command Reference in the Developer's Guide to determine legal input and output parameters for the functions. In the Developer's Guide, all parameters are referred to as **strings** or lists of **name-value pairs**. In the Perl API, these two types correspond directly to the types scalar and associative array (i.e., hash). As a Perl programmer, you should be quite familiar with these types.

NOTE: In the case of an error, often a descriptive error string will be available in the `self->{'error'}`member of the `DotSRS_Client` object. An example use is included in the example code for the contact methods below.

5.1 Query Methods

These methods correspond directly to the SRS Query Commands. Refer to section 4.2 of the Developer's Guide for a detailed listing of legal input and output parameters.

account_balance

Description:
Returns balance information from your partner account.

Called as:
```
($status, \%balance_ref) = account_balance ($tld)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

# create the object
$srs_client = new DotSRS_Client;

($status, $ref) = $srs_client->account_balance('tv');

unless ($status){
    print "account_balance failed (status=$status)\n";
}

foreach $key (keys %{ $ref }) {
    print "$key : ", $ref->{$key},"\n";
}
```

domain_info

Description:
Returns availability information for a specific domain name.

Called as:
```
(\%info_ref) = domain_info ($domain, $tld)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

if( @ARGV){
    $srs_client = new DotSRS_Client;

    ($ref) = $srs_client->domain_info(shift @ARGV, 'tv');

    foreach $key (keys %{ $ref }) {
       print "$key : ", $ref->{$key},"\n";
    }
}
else{
    print "Usage: dinfo <domain>\n";
}
```

multidomain_info

Description:
Returns availability and pricing information for many domain names at once.

Called as:
```
(\%info_ref) = multidomain_info (\%query_ref)
```

Example code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

# create the object
$srs_client = new DotSRS_Client;

$domain_ref = {     'DOMAIN 1' => 'domain-to-check1',
                    'TLD 1'    => 'tv',
                    'DOMAIN 2' => 'domain-to-check2',
                    'TLD 2'    => 'tv'
              };
($ref) = srs_client->multidomain_info($domain_ref);

foreach $key (keys %{ $ref }) {
    print "$key => ", $ref->{$key},"\n";
}
```

whois

Description:
Obtains domain name server and contact information for the given domain and TLD combination.

Called as:
```
(\%info_ref) = whois ($domain, $tld)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

if( @ARGV )
{
    $srs_client = new DotSRS_Client;

    ($ref) = $srs_client->whois( shift @ARGV, 'tv');

    foreach $key (keys %{ $ref }) {
       print "$key : ", $ref->{$key},"\n";
    }
}
else{
    print "Usage: whois <domain>\n";
}
```

5.2 Contact Management Methods

These methods correspond directly to the SRS Contact Management Commands. Refer to section 4.3 of the Developer's Guide for a detailed listing of legal input and output parameters.

create_contact
Description: Creates a new contact record and returns its unique ID. A return value of 0 for `$contact_id` indicates an error, in which case a descriptive string of the error(s) is then available in the `self->{'error'}` member of the `DotSRS_Client` object. (See example below)
Called as: `($contact_id, $request_id) = create_contact($transaction_id, \%contact_ref)`

Example code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

# create the object
$srs_client = new DotSRS_Client;

$contact_ref = {
    'TLD' => 'tv',
    'FNAME' => 'John',
    'LNAME' => 'Public',
    'ORGANIZATION' => 'John Q. Public Co.',
    'EMAIL' => 'johnq@public.com',
    'ADDRESS1' => '123 Main St.',
    'ADDRESS2' => 'Suite 100',
    'CITY'  => 'Metropolis',
    'PROVINCE' => 'CA',
    'POSTAL CODE' => '90024',
    'COUNTRY' => 'US',
    'PHONE' => '(310)555-1212'
    };

($contact_id, $request_id) =
$srs_client->create_contact( 1, $contact_ref );

if ($contact_id) {
    print "contactID=$contact_id\n";
    print "requestID=$request_id\n";
}
else{
    # 'error' member of client has descriptive error string(s)
    print "Error(s) creating contact:\n";
    print $srs_client->{'error'};
    print "\n";
}
```

edit_contact

Description:
Modifies fields in an existing contact record. A return value of 0 for `$contact_id` indicates an error, in which case a descriptive string of the error(s) is then available in the `self->{'error'}` member of the `DotSRS_Client` object. (See example below)

Called as:
```
($contact_id, $request_id) = edit_contact($transaction_id, \%contact_ref)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

# create the object
$srs_client = new DotSRS_Client;

$contact_ref = {
    'CONTACTID' => 171206,
    'EMAIL' => 'jqp@public.com',
    'ADDRESS2' => 'Suite 200',
};

($contact_id, $request_id) =
    $srs_client->edit_contact( 1, $contact_ref );

if ($contact_id) {
    print "contactID=$contact_id\n";
    print "requestID=$request_id\n";
}
else{
    # 'error' member of client has descriptive error string(s)
    print "Error(s) editing contact:\n";
    print $srs_client->{'error'};
    print "\n";
}
```

get_contact_info

Description:
Retrieves fields of an existing contact record.

Called as:
```
(\%info_ref) = get_contact_info($contact_id)
```

Example Code: Print contact fields for contact id entered on command line

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

# create the object
$srs_client = new DotSRS_Client;

$ref = $srs_client->get_contact_info( shift @ARGV );

foreach $key (keys %{ $ref }) {
    print "$key : ", $ref->{$key},"\n";
}
```

5.3 Domain Registration and Manipulation Methods

These methods correspond directly to the SRS Domain Registration and Manipulation Commands. Refer to section 4.4 of the Developer's Guide for a detailed listing of legal input and output parameters.

register_domain

Description:
Register (buy) a domain.

Called as:

```
($request_id, \%response_ref) =
register_domain ($transaction_id, \%domain_info_ref)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

# create the object
$srs_client = new DotSRS_Client;

$domain_ref = {
    'DOMAIN' => 'querydomain',
    'TLD'    => 'tv',
    'TERM YEARS' => 1,
    'RESPONSIBLE PERSON' => 17206,
    'TECHNICAL CONTACT' => 0,
    'PRICE'             => 100.00
    };

($request_id, $ref) = $srs_client->register_domain( 2, $domain_ref );

print $request_id;

if( $request_id ){
    print "requestID = $request_id\n";
}
else{
    print "register_domain failed.\n";
}

foreach $key (keys %{ $ref }) {
    print "$key : ", $ref->{$key},"\n";
}
```

change_domain

Description:
Modify contact and domain name server information for a domain.

Called as:

```
($request_id) = change_domain ($transaction_id, \%domain_info_ref)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
```

```
use DotSRS_Client;

# create the object
$srs_client = new DotSRS_Client;

$domain_ref = {
             'DOMAIN' => 'querydomain',
             'TLD' => 'tv',
             'RESPONSIBLE PERSON' => 0,
             'TECHNICAL CONTACT' => 0,
             'DNS SERVER NAME 1' => 'ns1.yahoo.com',
             };
$request_id = $srs_client->change_domain( 1, $domain_ref );

if( $request_id ){
    print "requestID = $request_id\n";
}
else{
    print "register_domain failed.\n";
}

foreach $key (keys %{ $ref }) {
    print "$key : ", $ref->{$key},"\n";
}
```

release_domain

Description:
Release a domain back into the pool of available domain names.

Called as:
`($request_id) = release_domain ($transaction_id, \%domain_info_ref)`

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

# create the object
$srs_client = new DotSRS_Client;

$domain_ref = {
             'DOMAIN' => 'querydomain',
             'TLD' => 'tv',
             };

$request_id = $srs_client->release_domain( 1, $domain_ref );


if( $request_id ){
    print "requestID = $request_id\n";
}
else{
    print "release_domain failed.\n";
}
```

renew_domain
Description: Renews a domain registration for a given number of years

Called as:
`($request_id, \%response_ref) =`
`renew_domain ($transaction_id, \%domain_info_ref)`

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

# create the object
$srs_client = new DotSRS_Client;

$domain_ref = {
              'DOMAIN' => 'querydomain',
              'TLD' => 'tv',
              'TERM YEARS' => 2,
              'PRICE' => 50
              };

($request_id,$ref) = $srs_client->renew_domain( 1, $domain_ref );

if( $request_id ){
    print "requestID = $request_id\n";
    foreach $key (keys %{ $ref }) {
       print "$key : ", $ref->{$key},"\n";
    }
}
else{
    print "renew_domain failed.\n";
}
```

5.4 Nameserver Methods

These methods correspond directly to the SRS Nameserver Commands. Refer to section 4.5 of the Developer's Guide for a detailed listing of legal input and output parameters.

register_nameserver

Description:
Add a nameserver.

Called as:

```
($request_id) = register_nameserver($transaction_id, \%ns_info_ref)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

$srs_client = new DotSRS_Client;

$ns_ref = {   'DNS SERVER NAME' => 'ns1.querydomain.tv',
              'DNS SERVER IP' => '127.0.0.1' };

$request_id = $srs_client->register_nameserver( 1, $ns_ref );

if( $request_id ){
    print "requestID = $request_id\n";
}
else{
    die "register_nameserver failed.\n";
}
```

release_nameserver

Description:
Remove a nameserver.

Called as:

```
($request_id) = release_nameserver($transaction_id, \%ns_info_ref)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

$srs_client = new DotSRS_Client;

$ns_ref = { 'DNS SERVER NAME' => 'ns1.querydomain.tv' };

$request_id = $srs_client->release_nameserver( 1, $ns_ref );

if( $request_id ){
    print "requestID = $request_id\n";
}
else{
    die "register_nameserver failed.\n";
}
```

get_nameserver_info

Description:
Obtain information about nameserver.

Called as:
```
(\%response_ref) = get_nameserver_info($transaction_id, \%ns_info_ref)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

$srs_client = new DotSRS_Client;

$ns_ref = { 'DNS SERVER NAME' => 'ns1.querydomain.tv' };

$ref = $srs_client->get_nameserver_info( 1, $ns_ref );

if( $ref ){
    foreach $key (keys %{ $ref }) {
       print "$key : ", $ref->{$key},"\n";
    }
}
else{
    die "nameserver_info failed\n";
}
```

5.5 Domain Transfer Methods

These methods correspond directly to the SRS Domain Transfer Commands. Refer to section 4.6 of the Developer's Guide for a detailed listing of legal input and output parameters.

request_transfer

Description:
Request a transfer of a domain to the SRSplus registrar from another registrar.

Called as:
```
($request_id, \%response_ref) =
request_transfer($transaction_id, \%transfer_info_ref)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

$srs_client = new DotSRS_Client;

$trans_ref = {'DOMAIN' => shift @ARGV,
              'TLD' => shift @ARGV,
              'CURRENT ADMIN EMAIL' => shift @ARGV
              };

($request_id, $ref) = $srs_client->request_transfer ( 1, $trans_ref);

if( $ref ){
    foreach $key (keys %{ $ref }) {
       print "$key : ", $ref->{$key},"\n";
```

```
    }
}
else{
    die "request_transfer failed\n";
}
```

outbound_transfer_response

Description:
Respond to a transfer request where you are the losing partner. You may ACCEPT or DENY the pending request.

Called as:
```
(\%response_ref) = outbound_transfer_response ($transaction_id, $domain, $tld,
$response_string)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

$srs_client = new DotSRS_Client;

$ref = $srs_client->outbound_transfer_response
                    ( 1, “TESTDOMAIN”, “ORG”, “ACCEPT” );

if( $ref ){
    foreach $key (keys %{ $ref }) {
       print "$key : ", $ref->{$key},"\n";
    }
}
else{
    die "outbound_transfer_response failed\n";
}
```

view_pending_transfers

Description:
Returns information about pending transfers, either INBOUND or OUTBOUND.

Called as:
```
(\%response_ref) = view_pending_transfers ($transaction_id, $transfer_type)
```

Example Code:

```
#!/usr/bin/perl
use lib qw ( /usr/local/www/perl-libs );
use DotSRS_Client;

$srs_client = new DotSRS_Client;

$ref = $srs_client->view_pending_transfers ( 1, “OUTBOUND”);

if( $ref ){
    foreach $key (keys %{ $ref }) {
       print "$key : ", $ref->{$key},"\n";
    }
}
else{
    die " view_pending_transfers failed\n";
}
```

6. Testing and Certification

Once you send in your public key and receive back your partner ID, you will have access to the test SRS server. While developing and debugging your client, you can perform SRS commands in the test environment without fear of incurring charges or destroying registry data. This server is functionally equivalent to the production server, but it separate and isolated. There are other differences, which we explore next.

6.1 The Test Environment vs. the Production Environment

The main difference between the two is that the database used by the test server is isolated and separate from the production server database. This test database does not necessarily reflect real-time conditions. It gets updated periodically with data from the production database, but you should never make assumptions about the state of the test database at any time. A name you register in it one day may not be there the next.

The test server does not actually charge against your account when you perform **RegisterDomain** commands. In fact, while in the testing environment, your account will have a zero balance, but you will have nearly unlimited buying power. When you perform the **AccountBalance** command you will see a response like the following:

Name	Value
BUYING_POWER	10000000.00
STORED_VALUE	0.00
UNPAID_CHARGES	100.00

Since the test server is updated from the live server occasionally, your `STORED_VALUE` may reset to the current live value from time to time. Once the `STORED_VALUE` reaches zero, the `UNPAID_CHARGES` will increase until it reaches the limit specified in `BUYING_POWER`. `BUYING_POWER` will be sufficiently large to accommodate about 200,000 test registrations before your test `BUYING_POWER` needs to be reset.

6.2 Running the Certification Tests

A sample script called `certify.pl` is included in the API Toolkit. It is used to perform the certification tests. You must successfully complete the certification tests and submit a log file of your client's activity during this test in order to gain access to the live production server. Once the log file is reviewed and approved, and if all other financial and contractual obligations have been met, you will gain "Active" status and may begin selling domain names to your customers. You will be notified of this accomplishment via email.

6.2.1 Logging Client Activity

You will need to log client activity while running the certification tests. This is accomplished most easily by redirecting the output of the script to a file. The log file will show a record of the raw command messages being sent back and forth between your SRS client and the SRS server. It is this log file that you must submit to `registry@srsplus.com` in order to achieve certification and access to the live SRS server.

6.2.2 Running the Test Script

The sample `certify.pl` file will perform all the SRS commands required for the certification test. The steps required to run the test are:

- Ensure that the scalar `$test_mode` in the file `DotSRS_Client.pm` is set to 1.
- Ensure that `DotSRS_Client.pm` is reachable by the script. You may need to edit the path in the `use lib` statement.
- Run the `certify.pl` script and redirect the output to a log file.
  Example: `perl certify.pl > srs.log`

After the test has finished, email the log file (as an attachment) to registry@srsplus.com with the subject line "Certification Log." Be sure to specify your organization's name in the body of the message. Once the log is reviewed you will be sent a reply email with your results. You will either pass or fail the test. In the case of failure, you will be given specific reasons why your test was rejected and you may try again after making corrective changes. Once you pass the test, you are given access to the live production server and your partner account is made "Active" in our system. You may then modify the API configuration to use the live SRS server.

6.3 Switching Environments

As discussed in section 4.2, the server environment is determined by the value of the scalar `$test_mode`, which is declared in the `DotSrs_Client.pm` file. When `$test_mode` is non-zero, you will be working against the test SRS server. When it is zero, you will be working against the live production server. Attempting to use the production server before completing the certification tests will result in error responses, typically "`Client not known to server`." After you do pass the certification tests, you may still work against the test server. In fact you may switch back and forth whenever necessary, or even run concurrently in both environments from two separate systems.

7. The Entire Integration Pathway

For your easy reference, the complete process from download to certification is presented in figure 7.1. You may want to make a separate copy of this flowchart and use it to keep track of your progress.

8. Next Steps

You should now feel prepared to start the development of your custom SRS client. This is the last formal document in the progression specified in the documentation roadmap. For further detailed help, take a look at the various FAQs and HOW-TO documents on the SRSplus website at http://www.srsplus.com/en/srsplus/partners_faq_tech.shtml. These supplemental documents are based on feedback from developers who have already used the APIs to create SRS clients. If you are having a problem with a specific process, or with a certain step in the integration pathway, it is likely that others have had the same problem and that we have an available FAQ or HOW-TO document addressing the topic.

Figure 7.1: Pathway from Signup to Certification