

Contents

- [1 About](#)
- [2 Installation](#)
- [3 Usage](#)
- [4 Troubleshooting](#)
- [5 Examples](#)
 - ◆ [5.1 fs_ivrd Example](#)
 - ◆ [5.2 POE Example](#)

About

The Perl ESL module allows for native interaction with FreeSWITCH over the event socket interface. It allows for sending commands, receiving output and sending and receiving events and IVR interaction from the FreeSWITCH server. The Perl ESL module is an auto-generated swig perl module with a binary component to it. You CANNOT just copy the PM file and use that, you must properly compile the module and get the ESL.so file generated also (which must be kept in the same directory as the ESL.pm). Note any time a change to libesl occurs you will need to re-make and install the perl ESL module or else things may break or work un-expectedly.

Installation

In the freeswitch source directory change to libs/esl and run:

```
make perlmod
make perlmod-install
```

This should install the ESL module into your perl lib folder. If for some reason you want to manually install it or keep it locally you still must run the make perlmod command to compile it but then can copy the libs/esl/ESL* (that includes ESL.pm and ESL.so Along with ESL/Dispatch.pm and ESL/IVR.pm if you want their functionality) to a folder of your choosing, make sure to keep the sub folder ESL structure intact and not just put all 4 in the same folder.

The following libraries may be needed for things to work properly: libxml2 pcre bzip2 curl gmp aspell php libtermcap gdbm db4

CentOS - some dependencies

```
yum install db4-devel gdbm-devel
```

Usage

The perl module provides two classes ESLconnection and ESLevent they are documented generally (non-perl specific) at:

<http://docs.freeswitch.org/classESLconnection.html>

and

<http://docs.freeswitch.org/classESLevent.html>

You will want to first include the module:

```
require ESL;
```

If you installed them somewhere not in the default path do:

```
use lib '/path/to/ESL/Location';
```

You can then establish a connection using:

```
my $fs = new ESL::ESLconnection($host, $port, $password);
```

Troubleshooting

- DO NOT use alarm in combination with the module at all (or at least with recvEvent) it will cause you to become disconnected from the server, use recvEventTimed instead.
- If you call an invalid function like \$reply->blah it may throw a swig error and say unable to load X, unfortunately X most likely doesn't exist and you just made a mistake.
- Make sure after updating freeswitch you recompile the perl module with make perlmod and make perlmod-install to ensure the libesl is properly up to date.
- Do NOT use the old FreeSWITCH::Client esl module this is extremely outdated and does NOT work properly any more.

Examples

Some of the major commands for being an esl client:

- \$fs->connected() - if you are connected
- my \$reply = \$fs->recvEvent() - get the next event
- my \$reply = \$fs->recvEventTimed(timeout_ms) - recv the next event or timeout after X ms
- \$reply->getHeader("header-name") - get the value of a header from the event
- \$reply->getBody() - receive the body of the event
- \$fs->disconnect() - disconnect

Perl_ESL

- `$fs->events("event_type","events to subscribe to")` - subscribe to these events for example:
`$fs->events("plain","heartbeat CHANNEL_HANGUP_COMPLETE");`
- `my $result = $fs->api("api command")` - execute an api command

For more examples see the `libs/esl/perl` folder for various perl examples for doing things with Perl ESL.

fs_ivrd Example

Another way to do this is by using the `fs_ivrd` script.

To call this script from the dialplan use:

```
<extension name="ivr-application">
  <condition field="destination_number" expression="^.$">
    <action application="set" data="ivr_path=/usr/src/freeswitch/libs/esl/perl/ivr1.pl"/>
    <action application="socket" data="127.0.0.1:9090 full"/>
  </condition>
</extension>
```

To run `fs_ivrd` which needs to running in the background, this is started by:

```
cd /usr/local/freeswitch/bin
```

```
./fs_ivrd -h 127.0.0.1 -p 9090
```

It would be nice if there was an init file to start `fs_ivrd` but for now we do it as above.

An example of `ivr1.pl` is below:

```
#!/usr/bin/perl
use ESL::IVR;

$| = 1;

select STDERR;
my $sound_root = '/usr/local/freeswitch/sounds/en/us/callie';
my $authenticated = 0;
my $tries = 0;
my $con = new ESL::IVR;
my $number;
my $pin;

#$con->setVar('tts_engine', 'flite');
#$con->setVar('tts_voice', 'slt');
#$con->setVar('sound_prefix', $sound_root);

my $uuid = $con->{_uuid};
$con->execute("answer", "", $uuid);
$con->execute("sleep", "500");

while (!$authenticated && $tries < 3) {
  my $number = $con->playAndGetDigits(
    4, 4, 3, 10000, "#",
    "ivr/ivr-please_enter_extension_followed_by_pound.w",
    "ivr/ivr-that_was_an_invalid_entry.wav",
    "EXT", "^\\d{4}\\$"
```

Perl_ESL

```
);

if (not defined $number) {
    $con->playback("voicemail/vm-goodbye.wav");
    $con->execute("hangup");
}

$con->execute("sleep", "1000");
$con->execute("flush_dtmf");
$pin = $con->playAndGetDigits(
    4, 4, 3, 10000, "#",
    "ivr/ivr-please_enter_pin_followed_by_pound.wav",
    "ivr/ivr-that_was_an_invalid_entry.wav",
    "PIN", "^\\d{4}"
);

$tries++;

$authenticated = 1;
$con->execute("sleep", "1000");
$con->execute("flush_dtmf");
}

$con->execute("hangup");
```

See also this page for more info: [Ivrd](#)

POE Example

This is an example of perl daemon that checks the event socket connection to FreeSWITCH every 5 seconds, if connected, it listens for Valet Parking events and does some MySQL stuff. I'm not a perl expert so I'm sure this code could be cleaned up a bit, but it works!

```
#!/usr/bin/perl
# Written by ES on 02/07/12
use warnings;
use strict;
use DBI;
use POE;
use IO::Handle;
require ESL;

# become daemon
my $pid = fork();
if($pid) {
    open ("PID", ">/var/run/perl-esl-out.pid");
    print PID $pid;
    close (PID);
    #end parent process
    #print "#parent process";
    exit(0);
}

# set new process group
setpgrp;

open STDOUT, '>>/var/log/perl-esl-out.log' or die "$0: Can't write to /var/log/perl-esl-out.log: ";
open STDERR, '>>/var/log/perl-esl-out.log' or die "$0: Can't write to /var/log/perl-esl-out.log: ";
open my $log_fh, '>>', '/var/log/perl-esl-out.log' or die "failed to open log: $!";
$log_fh->autoflush(1);
```

fs_ivrd Example

Perl_ESL

```
my $switch = "fs_switch01";
my $database = "perl-esl-out";
my $host = "perl-esl-out";
my $port = "3306";
my $user = "perl-esl-out";
my $pass = "perl-esl-out";
my $dsn = "DBI:mysql:database=$database;host=$host;port=$port";

my $ping = 0;
my $inbound;
my $success = 0;
my $sth;

POE::Session->create(
    inline_states => {
        _start => sub {
            $_[KERNEL]->delay(checkESL => 1);
        },
        checkESL => sub {
            if($inbound) {
                $ping = $inbound->api("eval 1");
                if (!$ping) {
                    $inbound = undef;
                } elsif($success) {
                    printf $log_fh "success!\n";
                    $success = 0;
                }
            }
            if(!$inbound) {
                $inbound = new ESL::ESLconnection("localhost", "8021", "ClueCon");
                if($inbound->connected()) {
                    $success = 1;
                    # event plain CUSTOM valet_parking::info
                    $inbound->events("plain", "CUSTOM valet_parking::info");
                    # filter Action hold
                    # filter Action exit
                    $inbound->filter("Action", "hold");
                    $inbound->filter("Action", "exit");
                    my $descriptor = $inbound->socketDescriptor();
                    open my $fh, "+<&=$descriptor" or die $!;
                    $_[KERNEL]->select_read($fh, 'receiveESL');
                    #$fh->blocking(1);
                }
                printf $log_fh "Connecting...";
            }
            $_[KERNEL]->delay(checkESL => 5);
        },
        receiveESL => sub {
            printf $log_fh "Received ESL.\n";
            if(!$inbound->connected()) {
                $_[KERNEL]->select_read($_[ARG0], undef);
            } else {
                my $e = $inbound->recvEvent();
                if ($e) {
                    my $dbh = DBI->connect($dsn, $user, $pass) || die "Could not connect";
                    my $action = $e->getHeader("Action");
                    if ($action) {
                        if ($action eq "exit") {
                            my $query = "DELETE FROM parking_lots WHERE lot=?";
                            if (!($sth = $dbh->prepare($query))) {
                                die ("Failed to prepare statement: " . DB

```

