

# Contents

- [1 Introduction](#)
- [2 Installation](#)
- [3 Reference](#)
  - ◆ [3.1 ESL Object](#)
    - ◇ [3.1.1 eslSetLogLevel](#)
  - ◆ [3.2 ESLevent Object](#)
    - ◇ [3.2.1 new](#)
    - ◇ [3.2.2 serialize](#)
    - ◇ [3.2.3 setPriority](#)
    - ◇ [3.2.4 getHeader](#)
    - ◇ [3.2.5 getBody](#)
    - ◇ [3.2.6 getType](#)
    - ◇ [3.2.7 addBody](#)
    - ◇ [3.2.8 addHeader](#)
    - ◇ [3.2.9 delHeader](#)
    - ◇ [3.2.10 firstHeader](#)
    - ◇ [3.2.11 nextHeader](#)
  - ◆ [3.3 ESLconnection Object](#)
    - ◇ [3.3.1 new](#)
    - ◇ [3.3.2 socketDescriptor](#)
    - ◇ [3.3.3 connected](#)
    - ◇ [3.3.4 getInfo](#)
    - ◇ [3.3.5 send](#)
    - ◇ [3.3.6 sendRecv](#)

- ◇ [3.3.7](#)  
[api](#)
- ◇ [3.3.8](#)  
[bgapi](#)
- ◇ [3.3.9](#)  
[sendEvent](#)
- ◇ [3.3.10](#)  
[recvEvent](#)
- ◇ [3.3.11](#)  
[recvEventTimed](#)
- ◇ [3.3.12](#)  
[filter](#)
- ◇ [3.3.13](#)  
[events](#)
- ◇ [3.3.14](#)  
[execute](#)
- ◇ [3.3.15](#)  
[executeAsync](#)
- ◇ [3.3.16](#)  
[setAsyncExecute](#)
- ◇ [3.3.17](#)  
[setEventLock](#)
- ◇ [3.3.18](#)  
[disconnect](#)
- [4 Examples](#)
  - ◆ [4.1](#)  
[Getting a](#)  
[uuid](#)
  - ◆ [4.2](#)  
[Simple](#)  
[Perl](#)  
[Example](#)
  - ◆ [4.3 Ruby](#)  
[Example](#)
  - ◆ [4.4 Java](#)  
[Example](#)
- [5 See Also](#)

## Introduction

Event Socket Library - see [mod\\_event\\_socket](#) to see examples and uses for ESL. This page describes how to use the Event Socket Library to talk to FreeSWITCH. ESL has no dependencies on FreeSWITCH. It can be built and moved to client/remote machines. It just encapsulates all the necessary socket stuff to allow talking to FreeSWITCH from an application.

## Installation

There are several options to configure mod\_esl to add support for multiple languages.

## Event\_Socket\_Library

In libs/esl, type make

Then, to enable specific languages type make + one of:

- perlmod to add Perl support
- phpmod to add PHP support
- luamod to add Lua support
- pymod to add Python support
- rubymod to add Ruby support
- javamod to add Java support
- managedmod to add mono support
- phpmod-install installs phpmod
- everymod builds perlmod phpmod luamod pymod rubymod managedmod javamod

**CentOS 5.x** you will need to install the following packages:

```
yum install libxml2-devel pcre-devel bzip2-devel curl-devel gmp-devel aspell-devel libtermcap-devel
```

**CentOS 6.x** you will also need to install **libedit-devel** in addition to the above packages.

and per language the respective dependencies:

for PHP:

```
yum install php-devel
```

for RUBY:

```
yum install ruby-devel ruby
```

for LUA:

```
yum install lua-devel
```

for JAVA:

```
yum install java-1.6.0-openjdk-devel
```

for PERL:

```
yum install perl-devel perl-ExtUtils-Embed
```

for PYTHON:

```
yum install python-devel
```

**Debian 4.x/5.x (Ubuntu as well)** you will need to install the following packages:

```
sudo apt-get install libxml2-dev libpcre3-dev libcurl4-openssl-dev libgmp3-dev libaspell-dev python-dev  
php5-dev libonig-dev libqdbm-dev libedit-dev
```

**NOTE:** You need to edit your Python version number in `libs/esl/python/Makefile` (replace 2.4 with 2.5 or 2.6...)

**NOTE:** For PHP you must also install `libdb-dev`.

## Reference

The following objects/methods should apply to any language that can build ESL extensions. Optional arguments are listed in brackets in the function signature. This section is a work in progress -- if you find any of the documentation to be in error, please correct!

## ESL Object

### **eslSetLogLevel**

```
eslSetLogLevel($loglevel)
```

Calling this function within your program causes your program to issue informative messages related to the Event Socket Library on `STDOUT`. `$loglevel` is an integer between 0 and 7. The values for `$loglevel` mean:

0 is EMERG

1 is ALERT

2 is CRIT

3 is ERROR

4 is WARNING

5 is NOTICE

6 is INFO

7 is DEBUG

Messages that have a lower value than `$loglevel` will be output on `STDOUT`, so higher values of `$loglevel` will cause the Library to log more information. Once this function is called, you can reduce the amount of log messages by calling this function again with a `$loglevel` of 0, but it cannot be completely turned off.

`eslSetLogLevel` is implemented as class-level method, as opposed to an instance-level method, so you do not need to create a new instance of the class to call this method. In Perl, for example:

```
use ESL;  
ESL::eslSetLogLevel(7);
```

## ESLevent Object

### **new**

```
new($event_type [, $event_subclass])
```

Instantiates a new event object that can use the methods below.

### **serialize**

```
serialize([$format])
```

Turns an event into colon-separated 'name: value' pairs similar to a sip/email packet (the way it looks on '/events plain all'). \$format can be:

- "xml"
- "json"
- "plain" (default)

### **setPriority**

```
setPriority([$number])
```

Sets the priority of an event to \$number in case it's fired.

### **getHeader**

```
getHeader($header_name)
```

Gets the header with the key of \$header\_name from an event object.

### **getBody**

```
getBody()
```

Gets the body of an event object.

### **getType**

```
getType()
```

Gets the event type of an event object.

### **addBody**

```
addBody($value)
```

Add \$value to the body of an event object. This can be called multiple times for the same event object.

### **addHeader**

```
addHeader($header_name, $value)
```

Add a header with key = \$header\_name and value = \$value to an event object. This can be called multiple times for the same event object.

### **delHeader**

```
delHeader($header_name)
```

Delete the header with key \$header\_name from an event object.

### **firstHeader**

```
firstHeader()
```

Sets the pointer to the first header in an event object, and returns it's key name. This must be called before nextHeader is called.

### **nextHeader**

```
nextHeader()
```

Moves the pointer to the next header in an event object, and returns it's key name. firstHeader must be called before this method to set the pointer. If you're already on the last header when this method is called, then it will return NULL.

## **ESLconnection Object**

### **new**

```
new($host, $port, $password)
```

Initializes a new instance of ESLconnection, and connects to the host \$host on the port \$port, and supplies \$password to freeswitch.

Intended only for an event socket in "Inbound" mode. In other words, this is only intended for the purpose of creating a connection to FreeSWITCH that is not initially bound to any particular call or channel.

## Event\_Socket\_Library

Does not initialize channel information (since inbound connections are not bound to a particular channel). In plain language, this means that calls to `getInfo()` will always return NULL.

```
new($fd)
```

Initializes a new instance of `ESLconnection`, using the existing file number contained in `$fd`.

Intended only for Event Socket Outbound connections. It will fail on Inbound connections, even if passed a valid inbound socket.

The standard method for using this function is to listen for an incoming connection on a socket, accept the incoming connection from FreeSWITCH, fork a new copy of your process if you want to listen for more connections, and then pass the file number of the socket to `new($fd)`.

### **socketDescriptor**

```
socketDescriptor()
```

Returns the UNIX file descriptor for the connection object, if the connection object is connected. This is the same file descriptor that was passed to `new($fd)` when used in outbound mode.

### **connected**

```
connected()
```

Test if the connection object is connected. Returns 1 if connected, 0 otherwise.

### **getInfo**

```
getInfo()
```

When FS connects to an "Event Socket Outbound" handler, it sends a "CHANNEL\_DATA" event as the first event after the initial connection. `getInfo()` returns an `ESLevent` that contains this Channel Data.

`getInfo()` returns NULL when used on an "Event Socket Inbound" connection.

### **send**

```
send($command)
```

Sends a command to FreeSWITCH.

Does not wait for a reply. You should immediately call `recvEvent` or `recvEventTimed` in a loop until you get the reply. The reply event will have a header named "content-type" that has a value of "api/response" or "command/reply".

To automatically wait for the reply event, use `sendRecv()` instead of `send()`.

`new`

### sendRecv

```
sendRecv($command)
```

Internally `sendRecv($command)` calls `send($command)` then `recvEvent()`, and returns an instance of `ESLevent`.

`recvEvent()` is called in a loop until it receives an event with a header named "content-type" that has a value of "api/response" or "command/reply", and then returns it as an instance of `ESLevent`.

Any events that are received by `recvEvent()` prior to the reply event are queued up, and will get returned on subsequent calls to `recvEvent()` in your program.

### api

```
api($command[, $arguments])
```

Send an API command to the FreeSWITCH server. This method blocks further execution until the command has been executed.

`api($command, $args)` is identical to `sendRecv("api $command $args")`.

### bgapi

```
bgapi($command[, $arguments][, $custom_job_uuid])
```

Send a background API command to the FreeSWITCH server to be executed in it's own thread. This will be executed in it's own thread, and is non-blocking.

`bgapi($command, $args)` is identical to `sendRecv("bgapi $command $args")`

Here is a Perl snippet that demonstrates the custom Job-UUID setting. (Works for all swigged languages)

```
my $command = shift;
my $args = join(" ", @ARGV);

my $con = new ESL::ESLconnection("localhost", "8021", "ClueCon");

my $e = $con->bgapi($command, $args, "my-job-id");
print $e->serialize("json");
```

This code returns:

```
{
  "Event-Name": "SOCKET_DATA",
  "Content-Type": "command/reply",
  "Reply-Text": "+OK Job-UUID: my-job-id",
  "Job-UUID": "my-job-id"
}
```

### sendEvent

```
sendEvent ($send_me)
```

### recvEvent

```
recvEvent ()
```

Returns the next event from FreeSWITCH. If no events are waiting, this call will block until an event arrives.

If any events were queued during a call to sendRecv(), then the first one will be returned, and removed from the queue. Otherwise, then next event will be read from the connection.

### recvEventTimed

```
recvEventTimed ($milliseconds)
```

Similar to recvEvent(), except that it will block for at most \$milliseconds.

A call to recvEventTimed(0) will return immediately. This is useful for polling for events.

### filter

```
filter($header, $value)
```

See the [event socket filter command](#).

### events

```
events ($event_type, $value)
```

\$event\_type can have the value "plain" or "xml". Any other value specified for \$event\_type gets replaced with "plain".

See the [event socket event command](#) for more info.

### execute

```
execute($app[, $arg][, $uuid])
```

Execute a [dialplan application](#), and wait for a response from the server. On socket connections not anchored to a channel (most of the time inbound), all three arguments are required -- \$uuid specifies the channel to execute the application on.

Returns an ESLevent object containing the response from the server. The getHeader("Reply-Text") method of this ESLevent object returns the server's response. The server's response will contain "+OK [Success Message]" on success or "-ERR [Error Message]" on failure.

## Event\_Socket\_Library

See the examples below for information on how to get a uuid to use when calling execute().

### **executeAsync**

```
executeAsync($app[, $arg][, $uuid])
```

Same as execute, but doesn't wait for a response from the server.

This works by causing the underlying call to execute() to append "async: true" header in the message sent to the channel.

### **setAsyncExecute**

```
setAsyncExecute($value)
```

Force async mode on for a socket connection. This command has no effect on outbound socket connections that are set to "async" in the dialplan and inbound socket connections, since these connections are already set to async mode on.

\$value should be 1 to force async mode, and 0 to not force it.

Specifically, calling setAsyncExecute(1) operates by causing future calls to execute() to include the "async: true" header in the message sent to the channel. Other event socket library routines are not affected by this call.

### **setEventLock**

```
setEventLock($value)
```

Force sync mode on for a socket connection. This command has no effect on outbound socket connections that are not set to "async" in the dialplan, since these connections are already set to sync mode.

\$value should be 1 to force sync mode, and 0 to not force it.

Specifically, calling setEventLock(1) operates by causing future calls to execute() to include the "event-lock: true" header in the message sent to the channel. Other event socket library routines are not affected by this call.

See Also:

[Q: Ordering and async keyword](#)

[Q: Can I bridge a call with an Outbound Socket?](#)

**disconnect**

```
disconnect()
```

Close the socket connection to the FreeSWITCH server.

**Examples****Getting a uuid**

On inbound connections, you get a new \$uuid to supply to execute() using:

```
$uuid = $esl->api("create_uuid")->getBody();
```

On outbound connections, the \$uuid to use can be acquired using:

```
$uuid = $esl->getInfo()->getHeader("unique-id")
```

**Simple Perl Example**

```
#!/usr/bin/perl
use strict;
use warnings;
require ESL;

my $host = "localhost";
my $port = "8021";
my $pass = "ClueCon";
my $con = new ESL::ESLconnection($host, $port, $pass);
if (! $con) { die "Unable to establish connection to $host:$port\n"; }
$con->events("plain", "all");

my $target = shift;
my $uuid = $con->api("create_uuid")->getBody();
my $res =
    $con->bgapi("originate", "{origination_uuid=$uuid}$target 9664");
my $job_uuid = $res->getHeader("Job-UUID");
print "Call to $target has Job-UUID of $job_uuid and call uuid of $uuid \n";

my $stay_connected = 1;
while ( $stay_connected ) {
    my $e = $con->recvEventTimed(20);
    if ( $e ) {
        my $ev_name = $e->getHeader("Event-Name");
        # Should we check for the $job_uuid to match the background job ?
        if ( $ev_name eq 'BACKGROUND_JOB' ) {
            if ( $e->getHeader("Job-UUID") eq $job_uuid ) {
                my $call_result = $e->getBody();
                print "Result of call to $target was $call_result\n\n";
            }
        }
        } elsif ( $ev_name eq 'DTMF' ) {
        my $digit = $e->getHeader("DTMF-Digit");
        print "Received DTMF digit: $digit\n";
    }
}
```

## Event\_Socket\_Library

```
        if ( $digit =~ m/\D/ ) {
            print "Exiting...\n";
            $stay_connected = 0;
        }
    } else {
        # Some other event
    }
} else {
    # do other things while waiting for events...
}
}
$con->api("uuid_kill",$uuid);
```

## Ruby Example

ESL in Inbound mode.

```
require 'ESL'

con = ESL::ESLconnection.new('127.0.0.1', '8021', 'ClueCon')
esl = con.sendRecv('api sofia status')
puts esl.getBody
```

ESL in Outbound mode in sync operation.

```
require 'socket'
require 'ESL'

server = TCPServer.new(8084)
loop do
  con = server.accept
  fd = con.to_i
  esl = ESL::ESLconnection.new(fd)
  esl.execute('answer')
  esl.execute('playback', '/usr/local/freeswitch/sounds/music/8000/suite-espanola-op-47-leyenda.w')
  esl.execute('hangup')
end
```

ESL in Outbound async full mode (<action application="socket" data="localhost:8086 async full"/>). It forks a process of every connection in order to support simultaneous sessions.

```
#!/usr/bin/ruby

require "ESL"
require 'socket'
include Socket::Constants
bind_address = "127.0.0.1"
bind_port = 8086

def time_now
  Time.now.strftime("%Y-%m-%d %H:%M:%S")
end

socket = Socket.new(AF_INET, SOCK_STREAM, 0)
sockaddr = Socket.sockaddr_in(bind_port, bind_address)
socket.bind(sockaddr)
socket.listen(5)
puts "Listening for connections on #{bind_address}:#{bind_port}"
```

## Simple Perl Example

## Event\_Socket\_Library

```
loop do
  client_socket, client_sockaddr = socket.accept #_nonblock
  pid = fork do
    @con = ESL::ESLconnection.new(client_socket.fileno)
    info = @con.getInfo
    uuid = info.getHeader("UNIQUE-ID")
    @con.sendRecv("myevents")
    @con.sendRecv("divert_events on")

    puts "#{time_now} [#{uuid}] Call to [#{info.getHeader("Caller-Destination-Number")}]"
    @con.execute("log", "1, Wee-wa-wee-wa")
    @con.execute("info", "")
    @con.execute("answer", "")
    @con.execute("playback", "/usr/local/freeswitch/sounds/music/8000/suite-espanola-op-47-legend")

    while @con.connected
      e = @con.recvEvent

      if e
        name = e.getHeader("Event-Name")
        puts "EVENT: #{name}"
        break if name == "SERVER_DISCONNECTED"
        if name == "DTMF"
          digit = e.getHeader("DTMF-DIGIT")
          duration = e.getHeader("DTMF-DURATION")
          puts "DTMF DIGIT #{digit} (#{duration})"
          if digit == "5"
            @con.execute("log", "1, WHA HA HA. User pressed 5")
          elsif digit == "8"
            # TODO: close connection without hangup in order to proceed in dialplan. How?
          elsif digit == "9"
            @con.execute("transfer", "99355151")
          end
        end
      end
    end

    end
  end
  puts "Connection closed"
end

Process.detach(pid)
end
```

## Java Example

Java inbound example that turns on events and prints them serialized to stdout.

```
/*
 * MyESLTest.java -- 16-Jun 2010
 *
 * An extremely simple java example of the javamod ESL wrapper for FreeSWITCH. Hopefully this will
 * I'd like to get an AJAX version of FOP (Flash Op Panel) and this could/would be the groundwork
 *
 * Anthony Cosgrove (zorprime)
 * acosgrove@aretta.com
 * Aretta Communications
 * http://www.aretta.com
 *
 */
```

## Event\_Socket\_Library

```
// Be sure to include the esl.jar in your project (I'm using eclipse to develop)
import org.freeswitch.esl.*;

public class MyESLTest {

    public static void main(String [] args) {

        /*
        *
        * Once you get libesljni.so compiled you can either put it in your java library
        * use System.loadlibrary or just use System.load with the absolute path.
        *
        */

        System.load("/lib64/fs/libesljni.so");

        /*
        *
        * Trying to keep this simple (and I'm no java expert) I am instantiating the ESL
        * ESLevent object in a static reference here so remember if you don't plan on do
        * you will need to instantiate your class first or you will get compile-time err
        *
        */

        ESLconnection con = new ESLconnection("127.0.0.1","8021","ClueCon");

        ESLevent evt;

        if (con.connected() == 1) System.out.println("connected");
        con.events("plain","all");

        // Loop while connected to the socket -- not sure if this method is constantly up
        while (con.connected() == 1) {

            // Get an event - recvEvent will block if nothing is queued
            evt = con.recvEvent();

            // Print the entire event in key : value format. serialize() according to
            // but if you do not put in something you will not get any output so I ju
            System.out.println(evt.serialize("plain"));

        }

    }

}
```

This and my attempt at a Java console GUI are in [freeswitch-contrib](#).

## See Also

- [mod\\_event\\_socket](#)
- [Perl ESL](#)
- [PHP ESL](#)
- [Python ESL](#)
- [Ruby ESL](#)