0. About

This page describes how to use the Event Socket Library (ESL) to talk to FreeSWITCH's event system using an application, and not via the C primitives. These applications can be

- internal to FreeSWITCH (such as fs_cli, communicating with FreeSWITCH via mod_event_socket)
- external and possibly remote (that is, residing on a different host).

ESL (libesl) is a C library that has no dependencies on FreeSWITCH, and 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 (i.e., provides an external API to interact with the Event System.

TODO What is the relation between mod_event_socket (moderlangevent, modkazoo, etc) and ESL?

in the source, mod_esl uses libesl, modeventsocket doesn't even touches it (same goes to the other related mods as well)

Saving my comments here.

This page should be the main source describing modes of operation (inbound, outbound).

Right now, it is described on individual module pages (mod_event_socket, mod_erlang_event) but they are just redundant.
Also, there is a subpage beneath Event Socket Library describing outbound mode (Event Socket Outbound) that kind of proves this point.

2019.11.06

Attila Gulyas

Going further, there should be a top level page telling about the event system in FreeSWITCH (that's where the Event List, Event Types (Classes, or Names, of Events), CUSTOM Event SUBCLASSES (Event-Subclass) would go, and only then start introducing Event Socket Library, with its modes of operation, libraries, modules, etc.

2019.11.06

1. Prerequisites

TODO this should be under installation divided up into section such as Dependencies, and the thing under Installation should be called build (the config part should go there too).

Dependencies MUST be installed before the initial "configure" of the source directory (i.e.: before compiling FS).

If you add the DEPENDENCIES AFTER the first "configure", then after adding the dependencies you must do:

```bash
recheck configuration

# connect to the source code directory
cd /usr/src/freeswitch
rm config.cache
rm .config.status
recheck
```

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

```
yum install libxml2-devel pcres-devel bzip2-devel curl-devel gmp-devel aspell-devel libtermcap-devel gdbm-devel db4-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 libdb4-dev libedit-dev
```
Installation

There are several options to configure mod_esl to add support for multiple languages.

First, in lib/esl, type make

```bash
# Under the FreeSWITCH source top-level directory
cd /usr/src/freeswitch/libs/esl
make
```

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

- perlmod-install 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

Example:

```bash
make perlmod-install
```

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!

Quoting and Escaping

When passing arguments to functions such as `originate`, if they contain any characters that affect parsing (spaces, quotes), they should be enclosed in single quotes. Single quotes within the data should be escaped with a backslash.

Example:

```bash
originate {origination_caller_id_name='fred\'s tire shop'}sofia/internal/100@mydomain.com 3000
```

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:

```perl
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.

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().

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 its own thread. This will be executed in its 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)

```perl
my $command = shift;
my $args = join(" ", @ARGV);
my $con = new ESL::ESLconnection("127.0.0.1", "8021", "ClueCon");
my $e = $con->bgapi($command, $args, "my-job-id");
print $e->serialize("json");
```

This code returns:

```json
{
  "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, then the next event will be read from the connection in sequence.

**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".

don't use this call if you want only myevents; you must use sendRecv("myevents") instead
See the event socket event command for more info.

if you want to use noevents; you must use sendRecv("noevents") instead

**execute**

```php
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.

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

**executeAsync**

```php
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**

```php
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 not to 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**

```php
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 not to 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**

```php
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:

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

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

```perl
$uuid = $esl-&gt;getInfo()-&gt;getHeader("unique-id")
```

Simple Perl Example
#!/usr/bin/perl
use strict;
use warnings;
require ESL;

my $host = "127.0.0.1";
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";
            }
        } elsif ( $ev_name eq 'DTMF' ) {
            my $digit = $e->getHeader("DTMF-Digit");
            print "Received DTMF digit: $digit\n";
            if ( $digit =~ m/\D/ ) {
                print "Exiting...\n";
                $stay_connected = 0;
            } else {  # Some other event
            } else {  # do other things while waiting for events...
            }
        }
    } else {  # do other things while waiting for events...
    }
}$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 (be aware that in sync operation many things will not work as expected. e.g. no DTMFs. Prefer async).
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.wav')
    esl.execute('hangup')
end

ESL in Outbound async full mode (<action application="socket" data="localhost:8086 async full">). It forks a process for 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}"
loop do
  client_socket, client_sockaddr = socket.accept
  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", ", Wee-wa-see-wa")
    @con.execute("info", ")
    @con.execute("answer", ")
    @con.execute("playback", "/usr/local/freeswitch/sounds/music/8000/suite-espanola-op-47-leyenda.wav")
    while @con.connected
      e = @con.recvEvent
      if e
        name = e.getHeader("Event-Name")
        puts "EVENT: #{name}"
        break if name == "SERVER_DISCONNECTED"
      else
        puts "SERVER_DISCONNECTED"
      end
      if name == "DTMF"
        digit = e.getHeader("DTMF-DIGIT")
        duration = e.getHeader("DTMF-DURATION")
        puts "DTMF DIGIT #{digit} (#{duration})"
        if digit == "5"
          @con.execute("log", ", WHA HA HA. User pressed 5")
        elsif digit == "8"
          # TODO: close connection without hangup in order to proceed in dialplan. How?
          @con.execute("transfer", "99355151")
        end
      end
    end
    puts "Connection closed"
  end
  end
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 turn some
gears.
 * I'd like to get an AJAX version of FOP (Flash Op Panel) and this could/would be the groundwork for it.
 * Anthony Cosgrove (zorprime)
 * acosgrove@aretta.com
 * Aretta Communications
 * http://www.aretta.com
 * */

// 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 path and
         * 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 ESLconnection and
         * ESLevent object in a static reference here so remember if you don't plan on doing everything
         * in main
         * you will need to instantiate your class first or you will get compile-time errors.
         *
         */
        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 updated so may
        // not be a good test
        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 the wiki
            // usually takes no arguments
            // but if you do not put in something you will not get any output so I just stuck plain
            // in.
            System.out.println(evt.serialize("plain"));
        }
    }
}
C Example

There are C examples for both inbound and outbound in the source code libs/esl dir. Check testclient.c and testserver.c for example.

Note: There's a difference in testserver.c of v1.2.stable and master(1.4).

Basically, the esl_listen_threaded takes a callback function that, when a new connection comes in, runs the callback - mycallback in this case.

```c
int main(void)
{
    esl_global_set_default_logger(7);
    esl_listen_threaded("localhost", 8040, mycallback, 100000);
    return 0;
}
```

the 1.4 version (master from 2014-02-16) added a private user_data param that allows you pass user_data to the callback, e.g.

```c
int main(void)
{
    char *private_data = "some private string or struct ...";
    esl_global_set_default_logger(7);
    esl_listen_threaded("localhost", 8040, mycallback, private_data, 100000);
    return 0;
}
```

See Also

- mod_event_socket
- Perl ESL
- PHP ESL
- Python ESL
- Ruby ESL