

# Event Socket Library quick starter

Arnaldo Pereira <arnaldo@sangoma.com>

Sangoma Technologies, 2011

## Introduction

The Event Socket Library, or ESL, is a library that aims to ease controlling FreeSWITCH from external applications, that can be written in any language and run in any operating system. It's written in C and has bindings for many languages: Perl, Python, PHP, Lua, Ruby, C# and Java.

The library controls FreeSWITCH through TCP connections, so the application doesn't have to be on the same server where FreeSWITCH is running. An ESL application has access to all the API commands exported by FreeSWITCH, the same was as it's default console does. Actually, `fs_cli`, the command line interface to FreeSWITCH, is written using ESL.

## Installation

Download the nightly snapshot from <http://files.freeswitch.org/freeswitch-snapshot.tar.gz> and follow the steps for your operating system:

### Linux

1. Install dependencies, as described on [http://wiki.freeswitch.org/wiki/Installation\\_Guide#Common\\_Prerequisites](http://wiki.freeswitch.org/wiki/Installation_Guide#Common_Prerequisites)
2. Decompress `freeswitch-snapshot.tar.gz` and enter the newly created directory
3. Run: `./bootstrap.sh && ./configure && make && sudo`

```
make installfdsa
```

4. Install if desired: `sudo make install`

## Windows

1. Install Visual Studio 2008 or 2010. Either the express or full version
2. Decompress freeswitch-snapshot.tar.gz
3. Open the appropriate solution file for your Visual Studio version. Freeswitch. 2008.express.sln for Visual Studio 2008, for example.
4. Build the solution
5. Install if desired

## Operation modes

Applications can receive and generate calls and, for that to be possible, **ESL** has two modes of operation: *inbound* and *outbound*.

### Inbound mode

In this mode, the application originate calls. It can also for example, originate calls, check users' status, load a FreeTDM span, etc.

The application (client) actively connects to the FreeSWITCH (server) in this mode, and runs commands.

### Example

Here's an example, written in C, that runs the api command status, print it out and exists:

```
#include <stdio.h>
#include <stdlib.h>
#include <esl.h>

int main(void)
{
    esl_handle_t handle = {{0}};
    esl_connect(&handle, "localhost", 8021, NULL, "ClueCon");
```

```

    esl_send_recv(&handle, "api status\n\n");

    if (handle.last_sr_event && handle.last_sr_event->body) {
        printf("%s\n", handle.last_sr_event->body);
    } else {
        // this is unlikely to happen with api or bgapi (which is hardcoded above) but
        prefix but may be true for other commands
        printf("%s\n", handle.last_sr_reply);
    }

    esl_disconnect(&handle);

    return 0;
}

```

This code was copied from testclient.c, bundled with ESL source code, that comes on the FreeSWITCH tree under libs/esl. To manually compile testclient.c (or your own program), assuming you're on libs/esl directory, run:

```
gcc -o testclient testclient.c -lpthread -lm -lesl -L. -Isrc/include
```

Testclient output:

```

user@localhost.localdomain esl $ ./testclient
UP 0 years, 0 days, 3 hours, 6 minutes, 23 seconds, 506
milliseconds, 232 microseconds
0 session(s) since startup
0 session(s) 0/30
1000 session(s) max
min idle cpu 0.00/100.00

```

## Outbound mode

In this mode, the application receive calls.

FreeSWITCH (server) connects to the application (client) whenever it receives a call that's routed to event\_socket, with one of the following dialplan actions:

```

<action application="socket" data="fs_server-hostname:8084"/>
<action application="socket" data="fs_server-hostname:8084 async"/>
<action application="socket" data="fs_server-hostname:8084 full"/>
<action application="socket" data="fs_server-hostname:8084 async full"/>

```

Once FreeSWITCH hits one of those actions, it'll attempt to connect to `fs_server-hostname` on port 8084. The call isn't automatically answered by the socket application, but instead FreeSWITCH transfers the control of the call to the application, which can decide to answer the call or not.

There are two optional keywords passed as parameters to these actions. They are:

**async:** When used, replies to commands are returned promptly. This means that the application might receive a reply before the command has actually finished executing. When the command is finished, the application will receive an event indicating that. If an application requires the outbound socket to generally behave asynchronously and, just on some particular commands to wait until the command finishes executing (synchronously), the application has to set `event-lock to true: event-lock: true`

**full:** When not set, the application is limited to control the call in place. When set, it allows the application to fully control FreeSWITCH.

More information on outbound socket can be found on [http://wiki.freeswitch.org/wiki/Event\\_Socket\\_Outbound](http://wiki.freeswitch.org/wiki/Event_Socket_Outbound)

## Examples

The following example was copied from `testserver.c`, also bundled with ESL. It answers a call and put the caller into a conference. Comments were added for clarity:

```
#include <stdio.h>
#include <stdlib.h>
#include <esl.h>

static void mycallback(esl_socket_t server_sock, esl_socket_t client_sock, struct
sockaddr_in *addr)
{
    esl_handle_t handle = {{0}};
    int done = 0;
    esl_status_t status;
    time_t exp = 0;

    if (fork()) {
        close(client_sock);
        return;
    }
```

```

}

/* attach our handle to the socket */
esl_attach_handle(&handle, client_sock, addr);

esl_log(ESL_LOG_INFO, "Connected! %d\n", handle.sock);

/* filter just the events from our channel, ignore the rest */
esl_filter(&handle, "unique-id", esl_event_get_header(handle.info_event, "caller-unique-id"));

/* subscribe to these events */
esl_events(&handle, ESL_EVENT_TYPE_PLAIN, "SESSION_HEARTBEAT CHANNEL_ANSWER
CHANNEL_ORIGINATE CHANNEL_PROGRESS CHANNEL_HANGUP "
"CHANNEL_BRIDGE CHANNEL_UNBRIDGE CHANNEL_OUTGOING CHANNEL_EXECUTE
CHANNEL_EXECUTE_COMPLETE DTMF CUSTOM conference::maintenance");

/* instruct FreeSWITCH to wait for the last channel event before closing the TCP
connection */
esl_send_rcv(&handle, "linger");

/* answer the call */
esl_execute(&handle, "answer", NULL, NULL);

/* put the caller on a conference */
esl_execute(&handle, "conference", "3000@default", NULL);

/* poll the socket until the last event is received, or an error occurs */
while((status = esl_rcv_timed(&handle, 1000)) != ESL_FAIL) {
    if (done) {
        if (time(NULL) >= exp) {
            break;
        }
    } else if (status == ESL_SUCCESS) {
        const char *type = esl_event_get_header(handle.last_event, "content-type");
        if (type && !strcasecmp(type, "text/disconnect-notice")) {
            const char *dispo = esl_event_get_header(handle.last_event, "content-
disposition");
            esl_log(ESL_LOG_INFO, "Got a disconnection notice disposition: [%s]\n",
dispo ? dispo : "");
            if (!strcmp(dispo, "linger")) {
                done = 1;
                esl_log(ESL_LOG_INFO, "Waiting 5 seconds for any remaining
events.\n");
                exp = time(NULL) + 5;
            }
        }
    }
}

esl_log(ESL_LOG_INFO, "Disconnected! %d\n", handle.sock);
esl_disconnect(&handle);
}

int main(void)
{
    esl_global_set_default_logger(7);
    esl_listen("localhost", 8084, mycallback);

    return 0;
}

```

SIP to TDM gateway example:

```

#include <stdio.h>
#include <stdlib.h>
#include <esl.h>

```

```

static void mycallback(esl_socket_t server_sock, esl_socket_t client_sock, struct
sockaddr_in *addr)
{
    esl_handle_t handle = {{0}};
    int done = 0;
    esl_status_t status;
    time_t exp = 0;
    char call_url[80];

    if (fork()) {
        close(client_sock);
        return;
    }

    /* attach our handle to the socket */
    esl_attach_handle(&handle, client_sock, addr);

    /* filter just the events from our channel, ignore the rest */
    esl_filter(&handle, "unique-id", esl_event_get_header(handle.info_event, "caller-
unique-id"));

    /* subscribe to these events */
    esl_events(&handle, ESL_EVENT_TYPE_PLAIN, "SESSION_HEARTBEAT CHANNEL_ANSWER
CHANNEL_ORIGINATE CHANNEL_PROGRESS CHANNEL_HANGUP "
"CHANNEL_BRIDGE CHANNEL_UNBRIDGE CHANNEL_OUTGOING CHANNEL_EXECUTE
CHANNEL_EXECUTE_COMPLETE DTMF CUSTOM conference::maintenance");

    /* instruct FreeSWITCH to wait for the last channel event before closing the TCP
connection */
    esl_send_recv(&handle, "linger");

    /* get the destination number and create the call url.
    here we'll use any channel from the span named "wp1" to dial */
    snprintf(call_url, sizeof(call_url), "freetdm/wp1/a/%s", esl_event_get_header
(handle.info_event, "channel-destination-number"));
    esl_log(ESL_LOG_INFO, "Bridging call to %s...\n", call_url);

    /* bridge the call to freetdm */
    esl_execute(&handle, "bridge", call_url, NULL);

    /* poll the socket until the last event is received, or an error occurs */
    while((status = esl_recv_timed(&handle, 1000)) != ESL_FAIL) {
        if (done) {
            if (time(NULL) >= exp) {
                break;
            }
        } else if (status == ESL_SUCCESS) {
            const char *type = esl_event_get_header(handle.last_event, "content-type");
            if (type && !strcasecmp(type, "text/disconnect-notice")) {
                const char *dispo = esl_event_get_header(handle.last_event, "content-
disposition");
                esl_log(ESL_LOG_INFO, "Got a disconnection notice dispostion: [%s]\n",
dispo ? dispo : "");
                if (!strcmp(dispo, "linger")) {
                    done = 1;
                    esl_log(ESL_LOG_INFO, "Waiting 5 seconds for any remaining
events.\n");
                    exp = time(NULL) + 5;
                }
            }
        }
    }

    esl_log(ESL_LOG_INFO, "Disconnected! %d\n", handle.sock);
    esl_disconnect(&handle);
}

int main(void)
{
    esl_global_set_default_logger(7);
    esl_listen("localhost", 8084, mycallback);
}

```

```

    return 0;
}

```

If we want to create a TDM to SIP gateway, we just have to change the call\_url from the last example. The full program would then become:

```

#include <stdio.h>
#include <stdlib.h>
#include <esl.h>

static void mycallback(esl_socket_t server_sock, esl_socket_t client_sock, struct
sockaddr_in *addr)
{
    esl_handle_t handle = {{0}};
    int done = 0;
    esl_status_t status;
    time_t exp = 0;
    char call_url[80];

    if (fork()) {
        close(client_sock);
        return;
    }

    /* attach our handle to the socket */
    esl_attach_handle(&handle, client_sock, addr);

    /* filter just the events from our channel, ignore the rest */
    esl_filter(&handle, "unique-id", esl_event_get_header(handle.info_event, "caller-
unique-id"));

    /* subscribe to these events */
    esl_events(&handle, ESL_EVENT_TYPE_PLAIN, "SESSION_HEARTBEAT CHANNEL_ANSWER
CHANNEL_ORIGINATE CHANNEL_PROGRESS CHANNEL_HANGUP "
"CHANNEL_BRIDGE CHANNEL_UNBRIDGE CHANNEL_OUTGOING CHANNEL_EXECUTE
CHANNEL_EXECUTE_COMPLETE DTMF CUSTOM conference::maintenance");

    /* instruct FreeSWITCH to wait for the last channel event before closing the TCP
connection */
    esl_send_rcv(&handle, "linger");

    /* get the destination number and create the call url.
    here we assume we're receiving a call from TDM and we bridge it to a local
extension,
    registered on the 'internal' sofia profile */
    snprintf(call_url, sizeof(call_url), "sofia/internal/%s192.168.2.19",
    esl_event_get_header(handle.info_event, "channel-destination-number"));
    esl_log(ESL_LOG_INFO, "Bridging call to %s...\n", call_url);

    /* bridge the call to freetdm */
    esl_execute(&handle, "bridge", call_url, NULL);

    /* poll the socket until the last event is received, or an error occurs */
    while((status = esl_rcv_timed(&handle, 1000)) != ESL_FAIL) {
        if (done) {
            if (time(NULL) >= exp) {
                break;
            }
        } else if (status == ESL_SUCCESS) {
            const char *type = esl_event_get_header(handle.last_event, "content-type");
            if (type && !strcasecmp(type, "text/disconnect-notice")) {
                const char *dispo = esl_event_get_header(handle.last_event, "content-
disposition");
                esl_log(ESL_LOG_INFO, "Got a disconnection notice dispostion: [%s]\n",
dispo ? dispo : "");
            }
        }
    }
}

```

```

        if (!strcmp(dispo, "linger")) {
            done = 1;
            esl_log(ESL_LOG_INFO, "Waiting 5 seconds for any remaining
events.\n");
            exp = time(NULL) + 5;
        }
    }
}

esl_log(ESL_LOG_INFO, "Disconnected! %d\n", handle.sock);
esl_disconnect(&handle);
}

int main(void)
{
    esl_global_set_default_logger(7);
    esl_listen("localhost", 8084, mycallback);

    return 0;
}

```

## mod\_event\_socket configuration

In order to configure FreeSWITCH to listen for TCP connections, the user has to configure mod\_event\_socket and instruct FreeSWITCH to automatically load it. The configuration for mod\_event\_socket lives on the file conf/autoload\_configs/event\_socket.conf.xml and here's a working example:

```

<configuration name="event_socket.conf" description="Socket Client">
  <settings>
    <param name="listen-ip" value="0.0.0.0"/>
    <param name="listen-port" value="8021"/>
    <param name="password" value="ClueCon"/>
    <param name="apply-inbound-acl" value="lan"/>
  </settings>
</configuration>

```

With this configuration, mod\_event\_socket will listen for TCP connections on port 8021 on all the available network addresses. But, it'll just accept the connections coming from LAN (which is a variable that's defined at conf/vars.xml file). The client has to authenticate with "ClueCon" as password.

## References



1. FreeSWITCH installation guide: [http://wiki.freewitch.org/wiki/Installation\\_Guide](http://wiki.freewitch.org/wiki/Installation_Guide)
2. Windows installation guide: [http://wiki.freewitch.org/wiki/Installation\\_for\\_Windows](http://wiki.freewitch.org/wiki/Installation_for_Windows)
3. Mod\_event\_socket: [http://wiki.freewitch.org/wiki/Mod\\_event\\_socket](http://wiki.freewitch.org/wiki/Mod_event_socket)
4. **ESL**, Event Socket Library: [http://wiki.freewitch.org/wiki/Event\\_Socket\\_Library](http://wiki.freewitch.org/wiki/Event_Socket_Library)
5. Event Socket Outbound: [http://wiki.freewitch.org/wiki/Event\\_Socket\\_Outbound](http://wiki.freewitch.org/wiki/Event_Socket_Outbound)