Lua API Reference

About

This page provides the documentation of the Lua FreeSWITCH API.

Docs Help Needed

session:execSome API aren't documented, so please feel free to improve the documentation and complete the API examples.

API Events

These methods apply to generating events.

event:addBody

`event:addBody helps in creating custom event.`

```
--Create Custom event
custom_msg =         "dial_record_id: " .. dial_record_id .. "\n" ..
               "call_disposition: " .. Disposition .. "\n" ..
               "campaign_number: "  .. Campaign .. "\n" ..
               "called_number: "    .. dial_num .."\n"  ;
               local e = freeswitch.Event("custom", "dial::dial-result");

               e:addBody(custom_msg);
               e:fire();
```

You can add as much data to the body as you like, in this case 4 items are to be sent.

The result will be:

```
[Content-Length] => 555
    [Content-Type] => text/event-plain
    [Body] => Array
        (
            [Event-Subclass] => dial::dial-result
            [Event-Name] => CUSTOM
            [Core-UUID] => 2dc7cc50-b157-4868-ae16-04e5f4b95dae
            [FreeSWITCH-Hostname] => pp6.noble.co.uk
            [FreeSWITCH-IPv4] => 192.168.0.106
            [FreeSWITCH-IPv6] => ::1
            [Event-Date-Local] => 2009-02-17 23:15:49
            [Event-Date-GMT] => Tue, 17 Feb 2009 23:15:49 GMT
            [Event-Date-Timestamp] => 1234912549610060
            [Event-Calling-File] => switch_cpp.cpp
            [Event-Calling-Function] => fire
            [Event-Calling-Line-Number] => 297
            [Content-Length] => 85
            [Body] => Array
                (
                    [dial_record_id] => 1234
                    [call_disposition] => AA
                    [campaign_number] => 20
                    [called_number] => 7777777
                )
        )
)
```

event:addHeader

event:delHeader

event:fire

```
local event = freeswitch.Event("message_waiting");
event:addHeader("MWI-Messages-Waiting", "no");
event:addHeader("MWI-Message-Account", "sip:1000@10.0.1.100");
event:addHeader("Sofia-Profile", "internal");
event:fire();
```

event:getBody

event:getHeader

This is a generic API call.

```
event:getHeader("Caller-Caller-ID-Name")
```

Or, This can be used inside of a dialplan.lua to get certain information

```
params:getHeader("variable_sip_req_uri")
```

event:getType

Returns the type of this event, either CUSTOM for custom events or one of the built-in event types.

event:serialize

Use this to dump all available Headers to the console.

```
-- Print as text
io.write(params:serialize());
io.write(params:serialize("text"));

-- Print as JSON
io.write(params:serialize("json"));
```

Or this to display them as an info message.

```
freeswitch.consoleLog("info",params:serialize())
```

event:setPriority

Sets the priority for the event, but requires a value of type `switch_priority_t` which doesn't seem to be available in the Lua binding.

event:fire (Sending an Event)

Using luarun to execute this code you can toggle the MWI on a registered phone on and off.

```
local event = freeswitch.Event
("message_waiting");

event:addHeader("MWI-Messages-Waiting",
"no");

event:addHeader("MWI-Message-Account", "sip:1002@10.
0.1.100");
event:fire();
```

API Sessions

The following methods can be applied to existing sessions.

session:answer

Answer the session:

```
session:answer();
```

session:answered

Checks whether the session is flagged as answered (true anytime after the call has been answered)

```
session:answered();
```

session:bridged

Check to see if this session's channel is bridged to another channel.

```
if (session:bridged() == true) do
    -- Do something
end
```

session:check_hangup_hook

session:collectDigits

session:consoleLog

Log something to the FreeSWITCH logger from session. Arguments are log level and message.

```
session:consoleLog("info",   "lua rocks\n");
session:consoleLog("notice", "lua rocks\n");
session:consoleLog("err",    "lua rocks\n");
session:consoleLog("debug",  "lua rocks\n");
session:consoleLog("warning","lua rocks\n");
```

session:destroy

Destroys the session and releases resources. This is done for you when your script ends, but if your script contains an infinite loop you can use this to terminate the session.

session:execute

session:execute(app, data)

```
local mySound = "/usr/local/freeswitch/sounds/music/16000/partita-no-3-in-e-major-bwv-1006-1-preludio.wav"
session:execute("playback", mySound)
```

NOTE: Callbacks (DTMF and friends) CAN NOT EXECUTE during an execute.

~~session:executeString~~

~~session:execute(api_string)~~

~~**NOTE:** Callbacks (DTMF and friends) CAN NOT EXECUTE during an execute.~~

session:flushDigits

session:flushEvents

session:get_uuid

session:get_uuid()

session:getDigits

Get digits:

- getDigits has three arguments: max_digits, terminators, timeout
- max_digits: maximum number of DTMF tones that will be collected
- terminators: buffer of characters that will terminate the digit collection
- timeout: timeout in milliseconds allowed for no input or after last digit is pressed and terminator isn't
- interdigit: if no input is received timeout specified previously will be use, once input is received this becomes the new timeout. (optional default 0)
- return: buffer containing collected digits
- The method blocks until one of the exit criteria is met.

```
digits = session:getDigits(5, "#", 3000);
session:consoleLog("info", "Got dtmf: ".. digits .."\n");
```

session:getState

Get the call state, i.e. "CS_EXECUTE". The call states are described in "switch_types.h".

```
state=session:getState();
```

session:getVariable

To get system variables such as ${hold_music}

```
local moh = session:getVariable("hold_music")
--[[ events obtained from "switch_channel.c"
 regards Monroy from Mexico
]]
        session:getVariable("context");
        session:getVariable("destination_number");
        session:getVariable("caller_id_name");
        session:getVariable("caller_id_number");
        session:getVariable("network_addr");
        session:getVariable("ani");
        session:getVariable("aniii");
        session:getVariable("rdnis");
        session:getVariable("source");
        session:getVariable("chan_name");
        session:getVariable("uuid");
```

session:hangup

You can hang up a session and provide an optional Hangup Cause Code Table.

```
session:hangup("USER_BUSY");
```

or

```
session:hangup(); -- default normal_clearing
```

session:hangupCause

You can find the hangup cause of an answered call and/or the reason an originated call did not complete. See Hangup Cause Code Table.

```
-- Initiate an outbound call
obSession = freeswitch.Session("sofia/192.168.0.4/1002")

-- Check to see if the call was answered
if obSession:ready() then
    -- Do something good here
else    -- This means the call was not answered ... Check for the reason
    local obCause = obSession:hangupCause()
    freeswitch.consoleLog("info", "obSession:hangupCause() = " .. obCause )
    if ( obCause == "USER_BUSY" ) then              -- SIP 486
       -- For BUSY you may reschedule the call for later
    elseif ( obCause == "NO_ANSWER" ) then
       -- Call them back in an hour
    elseif ( obCause == "ORIGINATOR_CANCEL" ) then   -- SIP 487
       -- May need to check for network congestion or problems
    else
       -- Log these issues
    end
end
```

Example call retry based on hangup cause:

Here's a Lua example in code which retries a call depending on the hangup cause. It retries max_retries1 times and alternates between 2 different gateways:

```
session1 = null;
max_retries1 = 3;
retries = 0;
ostr = "";
repeat
    retries = retries + 1;
    if (retries % 2) then ostr = originate_str1;
    else ostr = originate_str12; end
    freeswitch.consoleLog("notice", "*********** Dialing Leg1: " .. ostr .. " - Try: "..retries.."
***********\n");
    session1 = freeswitch.Session(ostr);
    local hcause = session1:hangupCause();
    freeswitch.consoleLog("notice", "*********** Leg1: " .. hcause .. " - Try: "..retries.." **********\n");
until not ((hcause == 'NO_ROUTE_DESTINATION' or hcause == 'RECOVERY_ON_TIMER_EXPIRE' or hcause ==
'INCOMPATIBLE_DESTINATION' or hcause == 'CALL_REJECTED' or hcause == 'NORMAL_TEMPORARY_FAILURE') and (retries <
max_retriesl1))
```

NOTE: originate_str1 and originate_str21 are dial strings for 2 different gateways.

session:hangupState

session:insertFile

`session:insertFile(<orig_file>, <file_to_insert>, <insertion_sample_point>)`

Inserts one file into another. All three arguments are required. The third argument is in samples, and is the number of samples into orig_file that you want to insert file_to_insert. The resulting file will be written at the sample rate of the session, and will replace orig_file.

Because the position is given in samples, you'll need to know the sample rate of the file to properly calculate how many samples are X seconds into the file. For example, to insert a file two seconds into a .wav file that has a sample rate of 8000Hz, you would use 16000 for the insertion_sample_point argument.

Note that this method requires an active channel with a valid session object, as it needs the sample rate and the codec info from the session.

Examples:

```
-- On a ulaw channel, insert bar.wav one second into foo.wav:
session:insertFile("foo.wav", "bar.wav", 8000)

-- Prepend bar.wav to foo.wav:
session:insertFile("foo.wav", "bar.wav", 0)

-- Append bar.wav to foo.wav:
session:insertFile("bar.wav", "foo.wav", 0)
```

session:mediaReady

session:originate

session:originate is deprecated, use the following construct instead:

```
new_session = freeswitch.Session("sofia/gateway/gatewayname/18001234567", session);
```

The code below is here for the sake of history only; please do not use it going forward.

```
-- this usage of originate is deprecated, use freeswitch.Session(dest, session)
new_session = freeswitch.Session();
new_session.originate(session, dest[, timeout]);
```

dest - quoted dialplan destination. For example: "sofia/internal/1000@10.0.0.1" or "sofia/gateway/my_sip_provider/my_dest_number"

timeout - origination timeout in seconds

Note: session.originate expects at least 2 arguments.

session:playAndGetDigits

Plays a file and collects DTMF digits. Digits are matched against a regular expression. Non-matching digits or a timeout can trigger the playing of an audio file containing an error message. Optional arguments allow you to transfer to an extension on failure, and store the entered digits into a channel variable.

Syntax

```
digits = session:playAndGetDigits (
          min_digits, max_digits, max_attempts, timeout, terminators,
          prompt_audio_files, input_error_audio_files,
          digit_regex, variable_name, digit_timeout,
          transfer_on_failure)
```

Arguments

min_digits	*The minimum number of digits required.*
max_digits	*The maximum number of digits allowed.*
max_attempts	*The number of times this function waits for digits and replays the prompt_audio_file when digits do not arrive.*
timeout	*The time (in milliseconds) to wait for a digit.*
terminators	*A string containing a list of digits that cause this function to terminate.*
prompt_audio_file	*The initial audio file to play. Playback stops if digits arrive while playing. This file is replayed after each timeout, up to max_attempts.*

input_error_audio_file	*The audio file to play when a digit not matching the digit_regex is received. Received digits are discarded while this file plays. Specify an empty string if this feature is not used.*
digit_regex	*The regular expression used to validate received digits.*
variable_name	***(Optional)*** *The channel variable used to store the received digits.*
digit_timeout	***(Optional)*** *The inter-digit timeout (in milliseconds). When provided, resets the timeout clock after each digit is entered, thus giving users with limited mobility the ability to slowly enter digits without causing a timeout. If not specified, digit_timeout is set to timeout.*
transfer_on_failure	***(Optional)*** *In the event of a failure, this function will transfer the session to an extension in the dialplan. The syntax is "* extension-name *[*dialplan-id *[*context*]]".*

Discussion

- This function returns an empty string when all timeouts and retry counts are exhausted.
- When the maximum number of allowable digits is reached, the function returns immediately, even if a terminator was not entered.
- If the user forgets to press one of the terminators, but has made a correct entry, the digits are returned after the next timeout.
- The session has to be answered before any digits can be processed. If you do not answer the call you, the audio will still play, but no digits will be collected.

Examples

This example causes FreeSWITCH to play *prompt.wav* and listen for between 2 and 5 digits, ending with the # key. If the user enters nothing (or something other than a digit, like the * key) *error.wav* is played, and the process is repeated another two times.

```
digits = session:playAndGetDigits(2, 5, 3, 3000, "#", "/prompt.wav", "/error.wav", "\\d+")
session:consoleLog("info", "Got DTMF digits: ".. digits .."\n")
```

This time, we require only one digit, and it must be 1, 3 or 4. If the user does not comply after three attempts, they are transferred to the operator extension in the "default" XML dial plan.

If the user presses a correct key, that digit is returned to the caller, and the "digits_received" channel variable is set to the same value.

```
digits = session:playAndGetDigits(1, 1, 3, 3000, "", "/menu.wav", "/error.wav", "[134]", "digits_received", 3,
"operator XML default")
session:consoleLog("info", "Got DTMF digits: ".. digits .."\n")
```

Reminder: If you need to match the * key in the regular expression, you will have to quote it twice:

```
digits = session:playAndGetDigits(2, 5, 3, 3000, "#", "/sr8k.wav", "", "\\d+|\\*");
```

session:preAnswer

Pre answer the session:

```
session:preAnswer();
```

session:read

Play a file and get digits.

```
digits = session:read(5, 10, "/sr8k.wav", 3000,
"#");
session:consoleLog("info", "Got dtmf: ".. digits .."\n");
```

session:read has 5 arguments: <min digits> <max digits> <file to play> <inter-digit timeout> <terminators>

This shows how you can figure out which terminator was pressed, if any:

```
session:setVariable("read_terminator_used", "")
digits = session:read(5, 10, "/sr8k.wav", 3000, "*#")
terminator = session:getVariable("read_terminator_used")
session:consoleLog("info", "Got dtmf: " .. digits .. " terminator: " .. terminator .. "\n")
```

session:ready

- checks whether the session is still active (true anytime between call starts and hangup)
- also session:ready will return false if the call is being transferred. Bottom line is you should always be checking session:ready on any loops and periodicly throughout your script and exit asap if it returns false.

See #session:hangupCause for more detail on if NOT ready.

```
while (session:ready() == true)
do

   -- do something
here

end
```

session:recordFile

syntax is ended_by_silence = session:recordFile(file_name, max_len_secs, silence_threshold, silence_secs)

silence_secs is the amount of silence to tolerate before ending the recording.
ended_by_silence is 0 if recording was ended by something else, eg an input callback getting dtmfs

```
function onInputCBF(s, _type, obj, arg)
    local k, v = nil, nil
    local _debug = true
    if _debug then
        for k, v in pairs(obj) do
            printSessionFunctions(obj)
            print(string.format('obj k-> %s v->%s\n', tostring(k), tostring(v)))
        end
        if _type == 'table' then
            for k, v in pairs(_type) do
                print(string.format('_type k-> %s v->%s\n', tostring(k), tostring(v)))
            end
        end
        print(string.format('\n(%s == dtmf) and (obj.digit [%s])\n', _type, obj.digit))
    end
    if (_type == "dtmf") then
        return 'break'
    else
        return ''
    end
end

recording_dir = '/tmp/'
filename = 'myfile.wav'
recording_filename = string.format('%s%s', recording_dir, filename)

if session:ready() then
    session:setInputCallback('onInputCBF', '');
    -- syntax is session:recordFile(file_name, max_len_secs, silence_threshold, silence_secs)
    max_len_secs = 30
    silence_threshold = 30
    silence_secs = 5
    test = session:recordFile(recording_filename, max_len_secs, silence_threshold, silence_secs);
    session:consoleLog("info", "session:recordFile() = " .. test )
end
```

session:sayPhrase

Play a speech phrase macro.

```
session:sayPhrase(macro_name [,macro_data] [,language]);
```

- macro_name - (string) The name of the say macro to speak.
- macro_data - (string) Optional. Data to pass to the say macro.
- language - (string) Optional. Language to speak macro in (ie. "en" or "fr"). Defaults to "en".

To capture events or DTMF, use it in combination with session:setInputCallback

Example:

```
function key_press(session, input_type, data, args)
  if input_type == "dtmf" then
    session:consoleLog("info", "Key pressed: " .. data["digit"])
    return "break"
  end
end
if session:ready() then
  session:answer()
  session:execute("sleep", "1000")
  session:setInputCallback("key_press", "")
  session:sayPhrase("voicemail_menu", "1:2:3:#", "en")
end
```

When used with setInputCallback, the return values and meanings are as follows:

- true or "true" - Causes prompts to continue speaking.
- Any other string value interrupts the prompt.

session:sendEvent

session:setAutoHangup

By default, lua script hangs up when it is done executing. If you need to run the next action in your dialplan after the lua script, you will need to setAutoHangup to false. The default value is true.

```
session:setAutoHangup(false)
```

session:setHangupHook

In your lua code, you can use setHangupHook to define the function to call when the session hangs up.

```
function myHangupHook(s, status, arg)
    freeswitch.consoleLog("NOTICE", "myHangupHook: " .. status .. "\n")
    -- close db_conn and terminate
    db_conn:close()
    error()
end

blah="w00t";
session:setHangupHook("myHangupHook", "blah")
```

Other possibilities to exit the script (and throwing an error)

```
return "exit";
or
return "die";
or
s:destroy("error message");
```

session:setInputCallback

```
function my_cb(s, type, obj, arg)
   if (arg) then
      io.write("type: " .. type .. "\n" .. "arg: " .. arg .. "\n");
   else
      io.write("type: " .. type .. "\n");
   end
   if (type == "dtmf") then
      io.write("digit: [" .. obj['digit'] .. "]\nduration: [" .. obj['duration'] .. "]\n");
   else
      io.write(obj:serialize("xml"));
      e = freeswitch.Event("message");
      e:add_body("you said " .. obj:get_body());
      session:sendEvent(e);
   end
end
blah="w00t";
session:answer();
session:setInputCallback("my_cb", "blah");
session:streamFile("/tmp/swimp.raw");
```

When used outside of streaming a file to a channel the return values "true" or "undefined" are accepted as true(which continues the audio stream I believe), anything else will be evaluated as false(which would stop the stream).

TODO: Additional return values can be located in the file ./src/switch_ivr.c around line 3359 with the option for "speed". Could not find out every option effect yet.

Return value effect.

Notice: Every return values are should be **STRING**.

Return value	**Effect**(When streaming audio)
"speed"	Unknown
"volume"	Unknown
"pause"	Stop audio util get the next input. After get the another input, then it play continally.
"stop"	Unkown
"truncate"	Unkown
"restart"	Unkown
"seek"	Unkown
"true"	Wait until the audio finish.
"false"	Stop the audio, immediately.
None/NULL	Don't return None/NULL. It makes type error problem. Especially python.

session:setVariable

Set a variable on a session:

```
session:setVariable("varname", "varval");
```

session:sleep

```
session:sleep(3000);
```

- **This will allow callbacks to DTMF to occur** and session:execute("sleep", "5000") won't.

session:speak

```
session:set_tts_params("flite", "kal");
session:speak("Please say the name of the person you're trying to contact");
```

session:say

Plays pre-recorded sound files for things like numbers, dates, currency, etc. Refer to Misc. Dialplan Tools say for info about the say application.

Arguments: <lang><say_type><say_method>

Example:

```
session:say("12345", "en", "number", "pronounced");
```

session:streamFile

Stream a file endless to the session

```
session:streamFile("/tmp/blah.wav");
```

Stream a file endless to the session starting at sample_count?

```
session:streamFile("/tmp/blah.wav", <sample_count>);
```

session:transfer

Transfer the current session. The arguments are extensions, dialplan and context.

```
session:transfer("3000", "XML", "default");
```

execution of your lua script will immediately stop, make sure you set session:setAutoHangup(false) if you don't want your call to disconnect

If instead you do session:execute("transfer", "3000 XML default") then the execution of the LUA script continues even though the call is mostly out of your control now, and bridges most likely will fail.

session:unsetInputCallback

session:waitForAnswer

freeswitch.IVRMenu

```
local menu = freeswitch.IVRMenu(
   main,                -- ?IVRMenu: the top level menu or nil if this is the top level one
   name,                -- ?string:  the name of the menu
   greeting_sound,      -- ?string:  the menu prompt played the first time the menu is played
   short_greeting_sound, -- ?string:  the short version of the menu prompt played on subsequent loops
   invalid_sound,       -- ?string:  the sound to play when an invalid entry/no entry is made
   exit_sound,          -- ?string:  played when the menu is terminated
   transfer_sound,      -- ?string:  the sound to play on transfer
   confirm_macro,       -- ?string:  phrase macro name to confirm input
   confirm_key,         -- ?string:  the key used to confirm a digit entry
   tts_engine,          -- ?string:  the TTS engine to use for this menu
   tts_voice,           -- ?string:  the TTS voice to use for this menu
   confirm_attempts,    -- ?int:     number of times to prompt to confirm input before failure
   inter_timeout,       -- ?int:     inter-digit timeout
   digit_len,           -- ?int:     max number of digits
   timeout,             -- ?int:     number of milliseconds to pause before looping
   max_failures,        -- ?int:     threshold of failures before hanging up
   max_timeouts)        -- ?int:     threshold of timeouts before hanging up
```

Instantiates a new IVRMenu object. To see a full example, go to the Lua IVR Menu Example page.

Methods

menu:bindAction

```
menu:bindAction(action, -- string(menu-exit|menu-sub|menu-exec-app|menu-say-phrase|menu-play-sound|menu-
back|menu-top)
                arg,    -- ?string: the arg to execute or nil if action is one of string(menu-top|menu-
back|menu-exit)
                bind)   -- string:  the pattern to bind to
```

Binds an action to the menu. See the documentation for IVR Menu.

Example

```
menu:bindAction("menu-exit", nil, "*")
menu:bindAction("menu-back", nil, "5")
menu:bindAction("menu-exec-app", "transfer 888 XML default", "7")
```

menu:execute

```
menu:execute(session,  -- Session: the session on which to execute this menu
             name)     -- ?string: the name of the menu
```

Executes the instantiated menu on the given session.

Example

```
menu:execute(session, "lua_demo_ivr")
```

Non-Session API

These methods are generic in that they do not apply to a session or an event. For example, printing data to the FreeSWITCH console is neither event- nor session-specific.

```
-----------------------------------------------------
-- SETUP "ESL.so" FOR LUA TO COMMUNICATE TO FREESWITCH
--    apt-get install lua5.2
--    git clone https://freeswitch.org/stash/scm/fs/freeswitch.git /usr/src/freeswitch.git
--    cd /usr/src/freeswitch.git
--    ./bootstrap.sh
--    ./configure
--    make -j
--    cd /usr/src/freeswitch.git/libs/esl
--    make luamod
--    mkdir -p /usr/local/lib/lua/5.2
--    cp lua/ESL.so /usr/local/lib/lua/5.2/ESL.so
-----------------------------------------------------
require("ESL")
--connect to freeswitch esl
local con = ESL.ESLconnection("127.0.0.1", "8021", "ClueCon");
--if successful connection give notice in FS log and lua script
if con:connected() == 1 then
    con:api("log","notice Lua ESL connected")
    print("connected!")
end
--send version command
version_response_body = con:api("version")
--log version number to freeswitch logfile
con:api("log","notice " .. version_response_body)
--print version to Lua output
print("Your FS version is: " .. version_response_body)
```

freeswitch.API

```
api = freeswitch.API();
-- get current time in milliseconds
time = api:getTime()
```

When calling a Lua script from the dialplan you always have the session object. However, Lua can also be called from the CLI. In either case, it is possible to execute API commands from within Lua by creating an API object:

```
api = freeswitch.API();
reply = api:executeString("version");
```

In the above snippet, the Lua variable *reply* would receive the version number from FreeSWITCH.

You can do more intricate things as well, like this:

```
api = freeswitch.API();
sofia = api:executeString("sofia status");
```

The Lua variable *sofia* would contain the total output of the **sofia status** command.

freeswitch.bridge

```
session1 = freeswitch.Session("sofia/internal/1001%192.168.1.1");
session2 = freeswitch.Session("sofia/internal/1002%192.168.1.1");
freeswitch.bridge(session1, session2);
```

freeswitch.consoleCleanLog

```
freeswitch.consoleCleanLog("This Rocks!!!\n");
```

freeswitch.consoleLog

Log something to the freeswitch logger. Arguments are loglevel, message.

```
freeswitch.consoleLog("info",   "lua rocks\n");
freeswitch.consoleLog("notice", "lua rocks\n");
freeswitch.consoleLog("err",    "lua rocks\n");
freeswitch.consoleLog("debug",  "lua rocks\n");
freeswitch.consoleLog("warning","lua rocks\n");
```

freeswitch.Dbh

Get an ODBC or core (sqlite) database handle from FreeSWITCH and perform an SQL query through it.

Advantage of this method is that it makes use of connection pooling provided by FreeSWITCH which gives a nice increase in speed when compared to creating a new TCP connection for each LuaSQL env:connect().

It works as follows:

```
local dbh = freeswitch.Dbh("dsn","user","pass") -- when using ODBC (deprecated)
-- OR --
local dbh = freeswitch.Dbh("core:my_db") -- when using sqlite (deprecated, if you have to use this to make it
work you should upgrade your FS installation)
-- OR --
local dbh = freeswitch.Dbh("sqlite://my_db") -- sqlite database in subdirectory "db"
-- OR --
local dbh = freeswitch.Dbh("odbc://my_db:uname:passwd") -- connect to ODBC database

assert(dbh:connected()) -- exits the script if we didn't connect properly

dbh:test_reactive("SELECT * FROM my_table",
                  "DROP TABLE my_table",
                  "CREATE TABLE my_table (id INTEGER(8), name VARCHAR(255))")

dbh:query("INSERT INTO my_table VALUES(1, 'foo')") -- populate the table
dbh:query("INSERT INTO my_table VALUES(2, 'bar')") -- with some test data

dbh:query("SELECT id, name FROM my_table", function(row)
  stream:write(string.format("%5s : %s\n", row.id, row.name))
end)

dbh:query("UPDATE my_table SET name = 'changed'")
stream:write("Affected rows: " .. dbh:affected_rows() .. "\n")

dbh:release() -- optional
```

- *freeswitch.Dbh(odbc://my_db:uname:passwd)* gets an ODBC db handle from the pool.
- *freeswitch.Dbh("sqlite://my_db")* gets a core db (sqlite) db handle from the pool (this automatically creates the db if it didn't exist yet).
- *dbh:connected()* checks if the handle is still connected to the database, returns true if connected, false otherwise.
- *dbh:test_reactive("test_sql", "drop_sql", "reactive_sql")* performs test_sql and if it fails performs drop_sql and reactive_sql (handy for initial table creation purposes)
- *dbh:query("query", function())* takes the query as a string and an optional Lua callback function that is called on each row returned by the db.
    - The callback function is passed a table representation of the current row for each iteration of the loop.
      Syntax of each row is: { ["column_name_1"] = "value_1", ["column_name_2"] = "value_2" }.
    - If you (optionally) return a number other than 0 from the callback-function, you'll break the loop.
- *dbh:affected_rows()* returns the number of rows affected by the last run INSERT, DELETE or UPDATE on the handle. It does not respond to SELECT operations.
- *dbh:release()* (optional) releases the handle back to the pool so it can be re-used by another thread. This is also automatically done when the dbh goes out of scope and is garbage collected (for example when your script returns). Useful for long-running scripts to release the connection sooner.

Take a look here for some examples.

freeswitch.email

Send an email with optional (converted) attachment.

Note that for this to work you have to have an MTA installed on your server, you also need to have 'mailer-app' configured in your switch.conf.xml.

You can use freeswitch.email as follows:

`freeswitch.email(to, from, headers, body, file, convert_cmd, convert_ext)`

- *to* (mandatory) a valid email address
- *from* (mandatory) a valid email address
- *headers* (mandatory) for example "subject: you've got mail!\n"
- *body* (optional) your regular mail body
- *file* (optional) a file to attach to your mail
- *convert_cmd* (optional) convert file to a different format before sending
- *convert_ext* (optional) to replace the file's extension

Example:

```
freeswitch.email("receiver@bar.com",
                 "sender@foo.com",
                 "subject: Voicemail from 1234\n",
                 "Hello,\n\nYou've got a voicemail, click the attachment to listen to it.",
                 "message.wav",
                 "mp3enc",
                 "mp3")
```

then the following system command will be executed before attaching "message.mp3" to the mail and send it on its way:

`mp3enc message.wav message.mp3`

freeswitch.Event

This is firing a custom event my::event.

```
local event = freeswitch.Event("custom", "my::event");
event:addHeader("My-Header", "test");
event:fire();
-- Send an event MESSAGE to a receiver
function FSMan:fire(nameHeader, header, body)
    local myEvent = freeswitch.Event("MESSAGE");
    nameHeader = Utils:trim(nameHeader); header = Utils:trim(header); body = Utils:trim(body);
    if (nameHeader == false ) then nameHeader="Generic_Name_Header" end
    if (header == false) then header="Header_Generic" end
    if (body == false) then body="Body_Generic" end
    myEvent:addHeader(nameHeader, header);
    myEvent:addBody(body);
    myEvent:fire();
end
```

freeswitch.EventConsumer

Consumes events from FreeSWITCH.

Usage (single event subscription):

```
con = freeswitch.EventConsumer("<event_name>"[,"<subclass type>"]);

-- pop() returns an event or nil if no events
con:pop()

-- pop(1) blocks until there is an event
con:pop(1)

-- pop(1,500) blocks for max half a second until there is an event
con:pop(1,500)
```

Usage (multiple specific event subscriptions):

```
con = freeswitch.EventConsumer();
con:bind("RELOADXML");
con:bind("SHUTDOWN");
con:bind("CUSTOM", "multicast::event");

-- pop() returns an event or nil if no events
con:pop()

-- pop(1) blocks until there is an event
con:pop(1)

-- pop(1,500) blocks for max half a second until there is an event
con:pop(1,500)
```

Examples:

```
con = freeswitch.EventConsumer("all");
session = freeswitch.Session("sofia/default/dest@host.com");
while session:ready() do
    session:execute("sleep", "1000");
    for e in (function() return con:pop() end) do
        print("event\n" .. e:serialize("xml"));
    end
end
-- or
while session:ready() do
    for e in (function() return con:pop(1,1000) end) do
        print("event\n" .. e:serialize("xml"))
    end
end
-- You may subscribe to specific events if you want to, and even subclasses
con = freeswitch.EventConsumer("CUSTOM");
con = freeswitch.EventConsumer("CUSTOM","conference::maintenance");
-- wait for a specific event but continue after 500 ms
function poll()
    -- create event and listener
    local event = freeswitch.Event("CUSTOM", "ping::running?")
    local con = freeswitch.EventConsumer("CUSTOM", "ping::running!")
    -- add text ad libitum
    event:addHeader("hi", "there")
    -- fire event
    event:fire()
    -- and wait for reply but not very long
    if con:pop(1, 500) then
        print("reply received")
        return true
    end
    print("no reply")
    return false
end
```

freeswitch.getGlobalVariable

Retrieves a global variable

```
my_globalvar = freeswitch.getGlobalVariable("varname")
```

freeswitch.msleep

Tells script to sleep for a specified number of milliseconds.
NOTE: Do **not** use this on a session-based script or bad things will happen.

```
-- Sleep for 500 milliseconds
freeswitch.msleep(500);
```

freeswitch.Session

Create a new session.

```
local session = freeswitch.Session("sofia/10.0.1.100/1001");
session:transfer("3000", "XML", "default");
```

Create a new session with execute_on_answer variable set.

```
local session = freeswitch.Session("[execute_on_answer=info notice]sofia/10.0.1.100/1001");
```

stream:write

API commands

You can write FreeSWITCH API commands *in Lua* by using the lua FreeSWITCH API command to run a script and pass the arguments in, then whatever you write with the stream object is what you get as a reply to that command. For example, given a script in the scripts directory called hello.lua with the following content:

```
stream:write("hello world")
```

Running 'lua hello.lua' from the FreeSWITCH console would return back "hello world".

Or calling it from the dialplan like so:

```
 <action application="set" data="foo=${lua(hello.lua)}"/>
```

Would set the channel variable "foo" to "hello world".

Web page interaction (via mod_xml_rpc)

```
--
-- lua/api.lua
--
-- enable mod_xml_rpc and try http://127.0.0.1:8080/api/lua?lua/api.lua in your webbrowser
--
stream:write("Content-Type: text/html\n\n");
stream:write("<title>FreeSWITCH Command Portal</title>");
stream:write("<h2>FreeSWITCH Command Portal</h2>");
stream:write("<form method=post><input name=command size=40> ");
stream:write("<input type=submit value=\"Execute\">");
stream:write("</form><hr noshade size=1><br>");

command = env:getHeader("command");

if (command) then
   api = freeswitch.API();
   reply = api:executeString(command);

   if (reply) then
      stream:write("<br><B>Command Result</b><br><pre>" .. reply .. "</pre>\n");
   end
end

env:addHeader("cool", "true");
stream:write(env:serialize() .. "\n\n");
```

Example: Call Control

```
--
-- call control lua script
--
dialA = "sofia/gateway/fs1/9903"
dialB = "user/1001"
legA = freeswitch.Session(dialA)
dispoA = "None"
while(legA:ready() and dispoA ~= "ANSWER") do
    dispoA = legA:getVariable("endpoint_disposition")
    freeswitch.consoleLog("INFO","Leg A disposition is '" .. dispoA .. "'\n")
    os.execute("sleep 1")
end -- legA while
if( not legA:ready() ) then
    -- oops, lost leg A handle this case
    freeswitch.consoleLog("NOTICE","It appears that " .. dialA .. " disconnected...\n")
else
    legB = freeswitch.Session(dialB)
    dispoB = "None"
    while(legA:ready() and legB:ready() and dispoB ~= "ANSWER") do
        if ( not legA:ready() ) then
            -- oops, leg A hung up or got disconnected, handle this case
            freeswitch.consoleLog("NOTICE","It appears that " .. dialA .. " disconnected...\n")
        else
            os.execute("sleep 1")
            dispoB = legB:getVariable("endpoint_disposition")
            freeswitch.consoleLog("NOTICE","Leg B disposition is '" .. dispoB .. "'\n")
        end -- inner if legA ready
    end -- legB while
    if ( legA:ready() and legB:ready() ) then
        freeswitch.bridge(legA,legB)
    else
        -- oops, one of the above legs hung up, handle this case
       freeswitch.consoleLog("NOTICE","It appears that " .. dialA .. " or " .. dialB .. " disconnected...\n")
    end
end -- outter if legA ready
```

Special Case: env object

When lua is called as the hangup hook there will be a special **env** object that contains all the channel variables from the channel that just disconnected.

Add an extension to test this feature:

```
<extension name="lua-env-hangup-hook-test">
  <condition field="destination_number" expression="^(1234)$">
    <action application="answer"/>
    <action application="set" data="api_hangup_hook=lua hook-test.lua"/>
    <action application="set" data="my_custom_var=foobar"/>
    <action application="sleep" data="10000"/>
    <action application="hangup"/>
  </condition>
</extension>
```

Then add freeswitch/scripts/hook-test.lua:

```
-- hook-test.lua
-- demonstrates using env to look at channel variables in hangup hook script

-- See everything
dat = env:serialize()
freeswitch.consoleLog("INFO","Here's everything:\n" .. dat .. "\n")

-- Grab a specific channel variable
dat = env:getHeader("uuid")
freeswitch.consoleLog("INFO","Inside hangup hook, uuid is: " .. dat .. "\n")

-- Drop some info into a log file...
res = os.execute("echo " .. dat .. " >> /tmp/fax.log")
res = os.execute("echo YOUR COMMAND HERE >> /tmp/fax.log")

-- If you created a custom variable you can get it also...
dat = env:getHeader("my_custom_var")
freeswitch.consoleLog("INFO","my_custom_var is '" .. dat .. "'\n")
```

Watch the FS console and dial 1234 and then hangup. You'll see all your channel variables!

Known issues

- lua ssl module - If you are using lua ssl module it might crash freeswitch, for me using it in Lua scripts was no issue, but if I was using lua ssl in hangup hook it was crashing freeswitch constantly. Comment from Mike: "Openssl has a global initializer and no way to keep it from only being run once. We run that in FreeSWITCH, then the luassl module re-runs it (corrupting the global openssl structures). There is no scoped way to keep it from doing that without writing a custom ssl lua wrapper specifically for FreeSWITCH. Writing this is a possibility but not something we plan on doing."

See also

- Lua examples
- mod_lua