# mod_verto

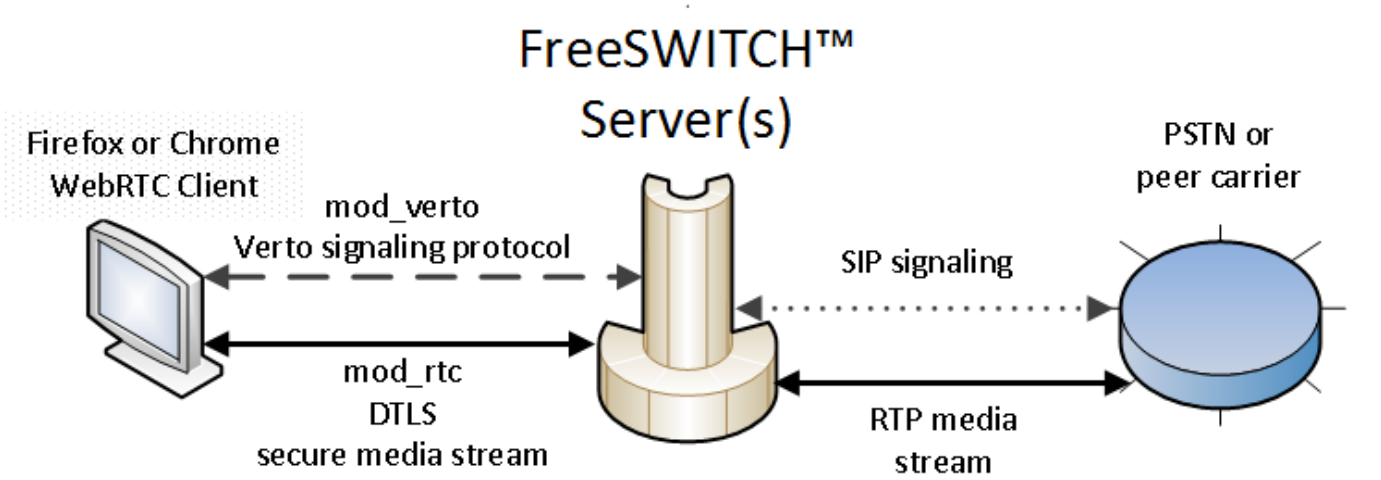
## About

**Verto** (VER-to) RTC is a FreeSWITCH endpoint that implements a subset of a JSON-RPC connection designed for use over secure websockets. The initial target is WebRTC to simplify coding and implementing calls from web browsers and devices to FreeSWITCH. This allows a web browser or other WebRTC client to originate a call using Verto into a FreeSWITCH installation and then out to the PSTN using SIP, SS7, or other supported protocol. This moves FreeSWITCH farther out onto the cutting edge of real-time communications technology while maintaining interoperability with SIP and other legacy protocols.

mod_verto is the signaling protocol. It depends on mod_rtc for secure media streaming services.

Sofia is a SIP stack used by FreeSWITCH.

Try Verto by calling the FreeSWITCH conference bridge with Chrome to hear stereo demonstration recordings or participate in our weekly conference calls to hear stereo positioning of conferees in crystal clear high-definition audio. Unfortunately, although Firefox is a good web browser, it does not support stereo WebRTC audio, therefore the stereo positioning will be lost on it.




## Features

Verto specifies a Javascript library to be used in the browser while speaking JSON to the server.

Provides a single library and signaling channel for session management, call control, text messaging, event messaging, and user data exchange.

## History

Verto was named after the Latin root word for "communication" or "dissemination of information" as seen in the modern use of the verb "ad**vert**ise". In this, Verto simplifies communications between modern devices.

## Applications

Verto performs most of the processing for its signaling protocol which relieves the web developer of a huge burden. A relatively simple Javascript enables a web browser to participate in a conference call that displays the list of participants with near-real-time updates of who is speaking; the moderator could be provided with visual controls to mute or kick a particular conferee. Because Verto is designed to work hand-in-hand with WebRTC it avoids the complications of using SIP, instead preferring a command structure designed for the purpose.

To see demonstration Javascript code look at the FreeSWITCH source code tree.

To see a demonstration of Verto in action, visit Verto Communicator WebRTC application

## Configuration

mod_verto implements a FreeSWITCH endpoint as a subset of a JSON-RPC connection designed for use over websockets.

mod_verto is the signaling protocol. It depends on mod_rtc for media streaming services.

mod_verto was introduced into the version 1.5 development branch of FreeSWITCH and is undergoing heavy development in the Master branch. Please see the Community page for ways to participate in the development and testing of this neat protocol.

## Certificates

WebRTC and WSS (secure web sockets) began life with security as a primary design goal, not an afterthought. In order for any WebRTC implementation to function you must install properly configured certificate(s) and have the correct A record(s) in your DNS pointing to your FreeSWITCH domain. The hacks previously available for SIP with force-registration-domain and related force- settings simply can not work in the WSS security model. This is a Good Thing.



Only proceed with the following instructions after you have properly installed your WSS certificates. Follow the directions on the WebRTC page under Install Certificates.

## verto.conf

Create this sample file conf/autoload_configs/verto.conf.xml

**verto.conf.xml**

```
<configuration name="verto.conf" description="HTML5 Verto Endpoint">
  <settings>
    <param name="debug" value="10"/>
    <param name="enable-presence" value="false"/>
    <param name="detach-timeout-sec" value="120"/>
  </settings>
  <profiles>
    <profile name="mine">
      <param name="bind-local" value="0.0.0.0:8081"/>
      <param name="bind-local" value="0.0.0.0:8082" secure="true"/>
      <param name="secure-combined" value="/usr/local/freeswitch/certs/wss.pem"/>
      <param name="secure-chain" value="/usr/local/freeswitch/certs/wss.pem"/>
      <param name="userauth" value="true"/>
      <param name="context" value="public"/>
      <param name="dialplan" value="XML"/>
      <param name="mcast-ip" value="239.1.1.1"/>
      <param name="mcast-port" value="1337"/>
      <param name="local-network" value="localnet.auto"/>
      <param name="apply-candidate-acl" value="wan.auto"/>
      <param name="rtp-ip" value="$${local_ip_v4}"/>
      <!--  <param name="ext-rtp-ip" value=""/> -->
      <param name="outbound-codec-string" value="opus,vp8"/>
      <param name="inbound-codec-string" value="opus,vp8"/>
      <param name="timer-name" value="soft"/>
      <!-- <param name="force-register-domain" value="$${domain}"/> -->
    </profile>
  </profiles>
</configuration>
```

## User Directory

You can specify verto-context and verto-dialplan on a per-user basis in the user directory config files.

This is a portion of an example user named 1000

**conf/directory/default/1000.xml**

```
<include>
  <user id="1000">
    <params>
      <param name="password" value="$${default_password}"/>
      <param name="vm-password" value="1000"/>
      <param name="verto-context" value="public"/>
      <param name="verto-dialplan" value="XML"/>
    </params>
  </user>
</include>
```

## Permissions

Permission to use Verto signaling can be granted to the entire user directory by entering the following lines into directory/default.xml

**User Directory**

```
<domain name="$${domain}">
  <params>
    <param name="jsonrpc-allowed-methods" value="verto"/>
    <param name="jsonrpc-allowed-event-channels" value="demo,conference,presence"/>
  </params>
</domain>
```

Permission to use Verto signaling can be granted to an individual user by entering the following lines into directory/default/1000.xml (for example)

**Individual Extension Permissions**

```
<user id="1000">
  <params>
    <param name="jsonrpc-allowed-methods" value="verto"/>
    <param name="jsonrpc-allowed-event-channels" value="demo,conference,presence"/>
  </params>
</user>
```

## Conference

To enable the livearray functionality that provides near-real-time updates on conference events, add the following to conf/autoload_configs/conference.conf.xml

**conference.conf.xml**

```
<param name="conference-flags" value="livearray-sync"/>
```

This would enable the kind of event reporting that allows a web page to display who is talking on the conference at the moment, for example.

To receive livearray data in JSON format, also enable the `livearray-json-status flag:`

**conference.conf.xml**

```
<param name="conference-flags" value="livearray-sync|livearray-json-status"/>
```

# Dialplan

To test if a call originated from a Verto client, test in your dialplan for the variable `Caller-Source` to equal mod_verto in a condition. There is also the variable `Other-Leg-Source` if the current leg is using SIP or SS7 and the connected leg might be a Verto call.

```
<extension name="verto call" continue="true">
    <condition field="caller-source" expression="mod_verto">
        <action application="log" data="INFO ***** Verto WebRTC Call ***** "/>
    </condition>
</extension>
```

# Bridging from WebRTC (mod_verto) to PSTN/ITSPs

WebRTC is slow to establish media. So we need to provide SDP asap.

We can do this by answering and sending some silent packets, instead of waiting for normal call setup:

Dialplan:

```
<extension name="itsp_send_call">
<condition field="destination_number" expression="^(\d+)$">
<action application="answer"/>
<action application="playback" data="silence_stream://2000"/>
<action application="set" data="effective_caller_id_number=333444555"/>
<action application="set" data="effective_caller_id_name=ciaociaociao"/>
<action application="bridge" data="{absolute_codec_string=pcmu}sofia/gateway/itsp/$1"/>
</condition>
</extension>
```

# Examples

To call a Verto endpoint from FreeSWITCH™ you have 3 choices:

## Choice 1

`Usage: ${verto_contact ${dialed_user}@${dialed_domain}}`

Use this format of Verto dial string alone in a bridge command to call a known Verto endpoint, or append this template to the end of the sip dial-string currently in your directory/default.xml to include Verto endpoints on all outbound calls, or add it to an individual user's dial-string in the user directory, such as directory/default/1000.xml for example.

**Verto Dial Strings**

```
<!-- Add Verto to an extension in your dialplan to call WebRTC users directly -->
<action application="bridge" data="${verto_contact ${dialed_user}@${dialed_domain}}"/>

<!-- Append the Verto dial-string to the generic dial-string in directory/default.xml
     to call WebRTC users along with SIP users simply by calling the user/xxxx endpoint -->
<param name="dial-string" value="{^^:sip_invite_domain=${dialed_domain}:presence_id=${dialed_user}
@${dialed_domain}}${sofia_contact(*/${dialed_user}@${dialed_domain})},${verto_contact ${dialed_user}
@${dialed_domain}}"/>

<!-- Create a custom dial-string for individual Verto users (e.g. in directory/default/1000.xml)
     to call this particular user via Verto instead of SIP -->
<params>
  <param name="dial-string" value="${verto_contact ${dialed_user}@${dialed_domain}}"/>
</params>
```

## Choice 2

`Usage: verto.rtc/foo@bar.com`

Use this in the bridge command the same way that you currently call "user/1000" for example. This is a nested originate, so don't put this form in the user directory dial-string because it will add a level of nesting in the code and duplicate effort.

If you already happen to know the uuid of the channel you can also call verto.rtc/u:<uuid> but verto_contact already looks up the connection uuid for you.

### Choice 3

```
Usage: verto_contact api func
```

Use this form in an API call to originate a connection to a Verto user.

## Client Side Samples

This sample code that is in use on the Verto Demonstration Site is under development and subject to change.

Fisheye contains JavaScript sample code for your perusal under freeswitch/html5/verto in the source tree.

These could stop working at any time, but anyone  running Firefox can press Ctrl-U to view the page source and discover these very same links.

Client sample code is located in our source code repository tree

### Verto Live Array

From a post to the freeswitch-users mailing list dated 2016.03.28 12:02 EDT the following JavaScript code snippet reveals the basics of how to get raw conference events using $.verto.livearray() from the Verto library:

**$.verto.livearray()**

```
var la = new $.verto.liveArray(verto, data.pvtData.laChannel, data.pvtData.laName,
  {subParams: {callID: dialog ? dialog.callID : null}, "onChange": function (obj, args) {

  }});
  la.onChange = function (obj, args) {
  console.error("The change", args);
  };
```

You can build on this template in your own JS code.

## Installation

At this moment (July 2014) a bunch of handwork is involved in getting mod_verto working after building everything from source from the latest master tree. I wanted it to run in my LAN from a Mac desktop to the Freeswitch server, both behind the same NAT with a self signed certificate… not the easiest way. This assumes that you've installed Freeswitch, freeswitch-music and freeswitch-sounds. Many thanks to bkw for all the info.



First things first; it is mandatory that you use mod_verto with the client over https on a complete domainname without certificate warnings. Otherwise it WILL NOT work. This means you need the parts of one set of ca cert/key/cert in a few places; FreeSWITCH and the web server (Apache probably) as well as the client need to know about them.

You will need a wss.pem file in the right place. You also need the crt in the tls directory. Under here is a script (of course you can create certs yourself, there's lots of howtos), but before that, some tips.

- The required directories certs & tls do not exist by default, they need to be created (the script fixes that)

- The documentation says you will need a chain certificate. Because you are your own ca, you don't need the chain certificate.

When running the script you must enter the fields correctly when asked (FQDN for example). Run as root, this assumes CONFDIR is /etc/freeswitch, change FQDN as well. You can change the defaults in the .sh scripts used also if you don't want to enter them during execution.

```
#!/bin/bash
CONFDIR=/etc/freeswitch
FQDN=host.domain
mkdir $CONFDIR/certs
mkdir $CONFDIR/tls
chown freeswitch:freeswitch $CONFDIR/certs $CONFDIR/tls
wget http://files.freeswitch.org/downloads/ssl.ca-0.1.tar.gz
tar zxfv ssl.ca-0.1.tar.gz
cd ssl.ca-0.1/
perl -i -pe 's/md5/sha1/g' *.sh
perl -i -pe 's/2048/2048/g' *.sh
./new-root-ca.sh
./new-server-cert.sh $FQDN
./sign-server-cert.sh $FQDN
cat $FQDN.crt $FQDN.key > $CONFDIR/certs/wss.pem
cat $FQDN.crt > $CONFDIR/tls/dtls-srtp.crt
chown freeswitch:freeswitch $CONFDIR/certs/wss.pem $CONFDIR/tls/dtls-srtp.crt
```

After this, make sure you have installed and enabled at least the following modules (conf/autoload_configs/modules.conf.xml): mod_verto, mod_rtc, mod_opus

You can also follow this link on how to generate your certs: http://datacenteroverlords.com/2012/03/01/creating-your-own-ssl-certificate-authority/

Enable the following lines in conf/directory/default.xml:

```
    <param name="jsonrpc-allowed-methods" value="verto"/>
    <param name="jsonrpc-allowed-event-channels" value="demo,conference,presence"/>
```

Enable the following line in conf/autoload_configs/conference.conf.xml:

```
<param name="conference-flags" value="livearray-sync"/>
```

Now, copy the certificate authority file you've created (/root/ssl.ca-0.1/ca.crt) to your client and install it. Also, add the host.domain to your hosts file or DNS to resolve it, if that isn't ready yet.

Apache on the server has to know about the certs too, assuming you've installed it and enabled ssl, change this in default-ssl (probably in /etc/apache2 /sites-available) and restart it:

```
SSLCertificateFile    $CONFDIR/certs/wss.pem
SSLCertificateKeyFile $CONFDIR/certs/wss.pem
```

As stated before, in my setup everything was behind the same NAT, and with a stun/turn server on the internet this causes trouble. You need to install your own stun/turn server behind the NAT. This is required because using at least one server is mandatory in WebRTC. My choice is https://code.google.com/p/rfc5766-turn-server/ . Look on the site how to do it.

Then, create a symlink on the server in the apache directory to the mod_verto demo for the client in the Freeswitch sources:

```
ln -s /usr/src/freeswitch/html5/verto/demo /var/www/verto
```

You will need to change the stun and turn server settings in verto-min.js in there too, just remove the defaults (including credentials) and change them to host.domainname:3478

Now, if everything is done correctly you can use chrome to surf to:

https://host.domain/verto/ and then you should be able to call the default conference at 3000 or 9386 for funny prompts.

If not; take a look at the Freeswitch console. If you can’t find, enable debugging in verto.conf.xml.

You can test if the SSL certs are correctly installed in the webserver with:

```
openssl s_client -connect host.domain:443
```

## See Also

Verto is not just for call signaling