Multi home tutorial

About

This document covers information about a multi home tutorial by Michael Gende back in October, 2009 so proceed with caution.

FREESWITCH FOR DUMMIES: DUAL-HOMED HOST EXAMPLE

The name tells all.

STUFF ONE OUGHT TO KNOW BEFORE STARTING

PURPOSE OF THIS DOCUMENT

When I was getting my own FS up and running, I did a lot of Wiki reading (and still do). There's a lot of information (including a really good "Getting Started" guide) on the FreeSwitch site, but I nonetheless found myself looking here and there. So, I tried to put into one place some things I had found about the basics. While I keep this pretty straightforward, there is the assumption that one is comfortable with computer systems, networking, VoIP in general, XML, these kinds of things. This is not developer documentation, just "getting it going and trying to understand what's what." You'll get some of my opinions, keep 'em or chuck 'em, up to you. But, one will also learn some important basics about how to do some hopfully useful things, where things are, and how to get FS to act like a phone system (in this case). From there, the sky's the limit. The idea here is to provide a foothold to step up higher.

Note that this document has much in common with the more mature "Getting Started" section on the FS website. This is more tuned to dual-homed setups, thus its creation. If you find it helpful, great. Find errors, point them out to me, please, and I'll correct them.

PRE-INSTALL CONSIDERATIONS

THE BOX:

Before one starts anything, ask yourself a few questions about how you would like FS to be deployed in your situation. One way, and a simple one at that, is to add a computer (or use an existing one) to your LAN. Just a box with, say, a GB of RAM and a dual-core CPU will do it (see the FS site for more precise details on system requirements). FS comes set up to work this way without much fooling around. However, using a dual-homed host has its advantages. The largest one being lessening the traffic introduced to your LAN because a WAN interface is available. I find this a good thing and worth the extra effort. You may not.

THE OS:

FS can run on a variety of OS's (the top three being macOS, Windows, and Linux I believe). If you can work with Linux, BSD and the like, you'll be fine. The FS site has a list of ways to install on different distros, take your pick. We initially went the CentOS route, which is a good Red Hat Linux compatible distro. For a computer with only one Ethernet interface on your LAN, that would be my personal choice. However, if you use a computer with separate LAN and WAN interfaces, I'd opt for PfSense (pfsense.org). This is a very nice FreeBSD system that is already a powerful firewall (and other great stuff). If you're connecting direct to the Internet, you'll need that. Further, with PfSense the installation of the OS and Firewall, via install disk image they provide, is pretty straightforward. Once FreeBSD/PfSense is installed, one adds FS as a package (see the directions at pfsense.org) with just a couple of mouse clicks. While FS on PfSense has some handy interfaces to provide aids in administration, I tend to run and configure it from the command line just the same. I do so because I want to know and understand the FS directory structure and XML files that make things work.

SIDEBAR: Want to run FS on a dual homed host?

That's what we do. One Ethernet port is set for the LAN while the other connects direct to the Internet. We opt for a static WAN IP address. I personally suggest using a fixed IP if you're going to go this route. It will make configuring FS, as well as well as potentially registering with your SIP provider, easier in my opinion. Also, one will have less new LAN traffic with the advent of you new VoIP system as one would with a single-homed host. This is what persuaded me. However, please note the FS scripts are pretty intelligent and can sense an IP change on your WAN port, letting the system know about it. However, for clarity and ease, largely my own, I'll go on about a static IP on the WAN and a separate IP address for the LAN port of your FS box.

NOTE: Just call your cable provider or DSL provider for info on getting a static IP if you'd like one. We have both DSL and Cable at my office, it was pretty simple to set up. These Internet providers, they'll do anything for money. If you have a T1 or better, you already have a static IP. Also note: going this way (dual-homed, fixed IP) gets your hands a little dirty, and I do mean "a little", with FS. That's something I think you'll want to do to understand more about how things work. It sure helped me.

MOVING ON: A FEW IMPORTANT FS COMMANDS FOR STARTERS

At this point, I'm assuming that you'll have the computer of your choice up and running with an OS and FS installed and waiting for your commands (if you don't, pretend). When I was getting FS up and running, there were a few concepts and commands that helped me get on my feet. So, for your edification, here they are:

To Start FS: The executable is in /usr/local/freeswitch/bin/freeswitch. Invoking it long-hand (or going to the ~/bin directory and typing ./freeswitch) will start FS in the foreground, attach to your terminal, and provide output and a command line.

To Check FS Status: If the screen is full of "junk", hit enter. You should see a prompt something like this: "freeswitch@FreeSwitch.local>". From there, you can tell FS to do things and ask about status. You'll use commands like "sofia status", "sofia status profile internal", and "sofia status profile external". One can accomplish quite a lot with just those few. More on these later in the document. Or, go try 'em out.

To Stop FS: At the command line mentioned above, just enter the command "shutdown" (no quotes). Or, one could kill it if its running in the background (as it does by default in PfSense on FreeBSD). I've done so with a "kill -15" followed by the FS PID (ps uax | grep free) and things shut down fine. There's probably a better way to do that, but this works.

To Reload XML Files into FS: You'll likely be making changes to the XML files that configure FS, especially initially. Once you edit an XML file, FS has to be told about this so your changes will take effect. That can be done with the "reloadxml" command. Alternatively, one can issue a "shutdown" command and then start FS from the command line. Take your choice.

For More Info: one can type "help" at the FS command prompt. There's a lot there; don't let it intimidate you.

THE FS FILE SYSTEM: WHERE STUFF IS FOR STARTERS:

Here's the basics on where things are. Master this and you've got a good, starting grasp on how to "get around" when configuring and maintaining your FS:

NOTE: Everything starts with /usr/local/freeswitch, so I'm just going to assume that with the tilde (~) here to save me from typing it again and again.

~/bin is where all the binaries, like the FS executable itself, resides.

~/conf is a starting point for your exploration. Some important files here, like vars.xml, and more.

~/conf/sip_profiles is very important. Here are all the SIP User Agent instructions. UAs listen for registrations of SIP Phones, etc.

~/conf/dialplan is where one sets up instructions telling FS about events and what to do when they take place.

~/conf/directory is where information on physical SIP phone extensions (and their groupings) is stored.

These directories have sub directories (which themselves have sub directories) that you'll want to get familiar with. I'll go into more detail later on. Also, I've left out where language stuff goes, and more. But, one has to start somewhere. Now, let's get to work.

SETTING THE SIP PROFILES TO USE DIFFERENT ETHERNET PORTS

INTERNAL LAN

Back to "dual-homed host setup with fixed IP on the WAN port". To make this work, one has to tell FS about where to be listening for SIP registrations, and other SIP traffic. That's done in the /usr/local/freeswitch/conf/sip_profiles directory. There, one finds what are conveniently called "sip profiles". One profile is for internal traffic (in my case called "internal.xml", for the phones on your LAN, etc) the other is for outside traffic (in my case called "external.xml", outside phone registrations, etc).

In the internal.xml file, we'll make some changes to accommodate the LAN network. Its a big file, but just a small tweak needs to be made. I'd edit the file and search for rtp-ip, sip-ip, ext-rtp-ip, and ext-sip-ip, respectively. Initially, you'll likely see "value=$${local_ip_v4}" in these four lines. Change the "value=" as below, using your own LAN IP address, for those four lines. Note that in the actual file there's a lot more stuff.

```
<param name="rtp-ip" value="192.168.0.199"/>
```

```
<param name="sip-ip" value="192.168.0.199"/>
```

Since I'm writing about a dual-homed example, we don't need a STUN server (i.e, we don't really need external rtp or sip) you can either leave the following two entries alone or set them as I have below:

```
<param name="ext-rtp-ip" value="192.168.0.199"/>
```

```
<param name="ext-sip-ip" value="192.168.0.199"/>
```

Another reminder to use the LAN IP of your computer running FS, not mine. Also remember this is a dual-homed host in this particular example. You don't need to mess with this - the the internal.xml or external.xml files - if you have a single Ethernet port connected to a LAN with a gateway to the Internet. The settings "from the factory" will do just fine "out of the box" with a single homed host (though you will likely need STUN, not covered here). But, what fun is that?

EXTERNAL WAN

So, now we've got to do the same sort of thing to the /usr/local/freeswitch/conf/sip_profiles/external.xml file too right? Actually, while you could, one does not need to. Strangely enough, and to one's great happiness, this profile will go out and get the address of the WAN side and make the proper adjustments. Again, it will also, if a dynamic IP is being used, sense the address and any changes to it. Again, I prefer the static IP and that's what I'm writing about here. The long and short of it is you don't have to change this file at this time.

GETTING YOUR LAN PHONES REGISTERED

Also, we make a similar change elsewhere. In the /usr/local/freeswitch/conf/directory area sits a file that is - I hope - called default.xml. This file also has something to do with SIP on your LAN. In this case, what end-points (phones, soft phones) FS should expect to see, and other info (such as grouping these end-points into, well, groups). So, edit that file but use your LAN IP instead of mine, just like in the profiles above:

```
<!--the domain or ip (the right hand side of the @ in the addr-->

<domain name="192.168.0.199">
```

While we're in the /usr/local/freeswitch/conf/directory area, let's consider how the SIP phones on your LAN are going to register with FS. I suggest using the pre-set 1000-1019 extension numbers - already set up for you - to configure and register your local SIP phones. If you need more, you'll know where /how to enter them in. So, while still in the default.xml file here - remember there are other default.xml files in other directories doing different things - note everything between the <groups> and </groups> tags. There are singular <group> </group> tags nested inside, setting up logical groupings of extension numbers into Sales, Billing, and Support. You can change these groupings to suite yourself, our go with what's there. Either way, you can now see where this configuration is set.

Also in this file, near the top, you'll see a pre-processor directive to go a directory below and include any .XML files found there. Leave this file, and cd down to the ./default directory. Just to be clear, we're going into directory /usr/local/freeswitch/conf/directory/default. Here's 1015.xml, one of the several you'll find there:

```
<include>
  <user id="1015">
    <params>
      <param name="password" value="$${default_password}"/>
      <param name="vm-password" value="1015"/>
    </params>
    <variables>
      <variable name="toll_allow" value="domestic,international,local"/>
      <variable name="accountcode" value="1015"/>
      <variable name="user_context" value="default"/>
      <variable name="effective_caller_id_name" value="Extension 1015"/>
      <!-- The value Extension 1015 can be personalized to your name, for instance, if you'd like -->
      <variable name="effective_caller_id_number" value="1015"/>
      <!-- Again, one could type in your main DID phone number instead of just 1015 if desired -->
      <variable name="outbound_caller_id_name" value="$${outbound_caller_name}"/>
      <variable name="outbound_caller_id_number" value="$${outbound_caller_id}"/>
      <variable name="callgroup" value="techsupport"/>
    </variables>
  </user>
</include>
```

That's the whole XML file. Now, these can get more complicated, but this is right there and ready to go. While the variables pretty much tell you what they do, the point here is that if you need to modify something about a LAN SIP phone's "personal info", you come to /usr/local/freeswitch/conf/directory/default and look at the *.xml files here. And/or go "up" one level and look in the default.xml file there. NOTE: On my system, there's another default.xml file here. Keep in mind that while the names are the same, where they are has a lot to do with their function, so associate the place with the name. It gets easier with time.

CHANGING THE PASSWORD FOR SIP REGISTRATION ON YOUR LAN

Last thing: You'll want to change the default password that your LAN SIP phones (or external phones for that matter) will use when authenticating. Leaving the one set up "out of the box" is a security risk since anyone could know it. So, make up your own. Edit the /usr/local/freeswitch/conf/vars.xml file. Near the top you'll see:

```
<X-PRE-PROCESS cmd="set" data="default_password=1234"/>

<!-- Did you change it yet? -->
```

Change the default_password parameter to some other four numbers that appeal to you. Now, only SIP phones that know the secret can register. Having done all this, set up your LAN SIP phones. Assign each a number between 1000 and 1019 for starters because these are already configured. Point your SIP phones the the IP address we've hard coded in the profiles (192.168.0.199 in my example). Be sure to use the new password you just set up.

APPLYING YOUR CHANGES AND CHECKING YOUR WORK

Once you've made all these changes, either reloadxml the FS or simply stop and start it (don't know how? See "Stuff I Ought to Know Before Starting" section). To check, after the reload or restart, issue this command at the FS command line:

```
freeswitch@FreeSwitch.local> sofia status profile internal
```

You'll get something like the following if all is well:

```
API CALL [sofia(status profile internal)] output:
=================================================================================================
Name                    internal
Domain Name             N/A
DBName                  sofia_reg_internal
Pres Hosts
Dialplan                XML
Context                 public
Challenge Realm         auto_to
RTP-IP                  192.168.0.199
Ext-RTP-IP              192.168.0.199
SIP-IP                  192.168.0.199
Ext-SIP-IP              192.168.0.199
URL                     sip:mod_sofia@192.168.0.199:5060
BIND-URL                sip:mod_sofia@192.168.0.199:5060
HOLD-MUSIC              local_stream://moh
OUTBOUND-PROXY          N/A
CODECS                  G7221@32000h,G7221@16000h,G722,PCMU,PCMA,GSM
TEL-EVENT               101
DTMF-MODE               rfc2833
CNG                     13
SESSION-TO              0
MAX-DIALOG              0
NOMEDIA                 false
LATE-NEG                false
PROXY-MEDIA             false
AGGRESSIVENAT           false
STUN-ENABLED            true
STUN-AUTO-DISABLE       false
CALLS-IN                2
FAILED-CALLS-IN         0
CALLS-OUT               3
FAILED-CALLS-OUT        0

Registrations:
=================================================================================================

Call-ID:        e6c864e9c4a3d@192.168.0.80
User:           1013@192.168.0.199
Contact:        "user" <sip:1013@192.168.0.80:5062;transport=udp;fs_nat=yes;fs_path=sip%013%40192.168.0.80%062%
3Btransport%3Dudp>
Agent:          Grandstream GXP2000 1.1.6.46
Status:         Registered(UDP-NAT)(unknown) EXP(2009-09-23 16:37:10)
Host:           FreeSwitch.local
IP:             192.168.0.80
Port:           5062
Auth-User:      1013
Auth-Realm:     192.168.0.199

Call-ID:        88ee646b068da@192.168.0.41
User:           1012@192.168.0.199
Contact:        "user" <sip:1012@192.168.0.41:5062;transport=udp;user=phone;fs_nat=yes;fs_path=sip%012%
192.168.0.41%3A5062%3Btransport%udp%3Buser%3Dphone>
Agent:          Grandstream GXP2000 1.1.5.15
Status:         Registered(UDP-NAT)(unknown) EXP(2009-09-23 16:37:12)
Host:           FreeSwitch.local
IP:             192.168.0.41
Port:           5062
Auth-User:      1012
Auth-Realm:     192.168.0.199

Call-ID:        45ee6e1b083da@192.168.0.57
User:           1017@192.168.0.199
Contact:        "user" <sip:1017@192.168.0.57:5062;transport=udp;fs_nat=yes;fs_path=sip%17%40192.168.0.57%62%
```

```
3Btransport%3Dudp>
Agent:          Grandstream GXP2000 1.1.5.15
Status:         Registered(UDP-NAT)(unknown) EXP(2009-09-23 16:37:15)
Host:           FreeSwitch.local
IP:             192.168.0.57
Port:           5062
Auth-User:      1017
Auth-Realm:     192.168.0.199

=================================================================================================
```

Note that everywhere there's an IP address (RTP_IP, SIP_IP, EXT_RTP_IP, EXT_SIP_IP), its the LAN Ethernet address on our FS box or the address of a LAN SIP phone (under Registrations). That's what we want. The internal.xml profile is listening ONLY on the LAN.

Do the same thing for the external profile:

```
freeswitch@FreeSwitch.local> sofia status profile external
```

...and get output similar to the above, but with no registered sip phones as yet:

```
API CALL [sofia(status profile external)] output:
=================================================================================================
Name                    external
Domain Name             N/A
DBName                  sofia_reg_external
Pres Hosts
Dialplan                XML
Context                 public
Challenge Realm         auto_to
RTP-IP                  yy.yy.yy.yy
Ext-RTP-IP              yy.yy.yy.yy
SIP-IP                  yy.yy.yy.yy
Ext-SIP-IP              yy.yy.yy.yy
URL                     sip:mod_sofia@yy.yy.yy.yy:5080
BIND-URL                sip:mod_sofia@yy.yy.yy.yy:5080
HOLD-MUSIC              local_stream://moh
OUTBOUND-PROXY          N/A
CODECS                  PCMU,PCMA,GSM
TEL-EVENT               101
DTMF-MODE               rfc2833
CNG                     13
SESSION-TO              0
MAX-DIALOG              0
NOMEDIA                 false
LATE-NEG                false
PROXY-MEDIA             false
AGGRESSIVENAT           false
STUN-ENABLED            true
STUN-AUTO-DISABLE       false
CALLS-IN                1
FAILED-CALLS-IN         0
CALLS-OUT               1
FAILED-CALLS-OUT        0

Registrations:
=================================================================================================
=================================================================================================
```

Here, we have only the external Ethernet static-ip address showing up. Again, that's as it should be. This profile is listening only to the WAN and not interfering with the LAN. Note that in my case, I don't have any registrations on this profile. That's OK, but I could set things up for phones to register from the WAN if that was desirable (which in our case, it is not as yet). As we'll see, these two will work together well for out-going and in-coming calls. Read on.

REGISTERING WITH A SIP PROVIDER

So that we can make and receive calls, we'll need to register with a SIP/PSTN provider. Getting one is outside the scope of this document, but check out the Internet, many are available. You'll have to have FS register with your "gateway provider" so things can work. Here's how we did it: In /usr/local/freeswitch/conf/sip_profiles/external, I created a file called urbancom.xml (happens to be the name of my provider). As long as this file is in that directory, FS will use it to register with my SIP/PSTN provider, making calling "in and out" possible. Here's what's inside:

```
<include>
  <gateway name="xxx.xxx.xxx.xxx">
     <!-- Remember, you'll need this name for your "out-bound calls" extension -->
     <param name="username" value="8155551212"/>
     <!-- Natch', your own username, ours is like a NPANXXTN -->
     <param name="password" value="somekindapassw0rd"/>
     <!-- Natch', your own password -->
     <param name="expire-seconds" value="60"/>
     <!-- Time out value, 60 works for us -->
     <param name="register" value="true"/>
     <!-- You want to register, so "true" -->
     <param name="register-transport" value="udp"/>
     <!-- You'll probably use udp, but ask your provider -->
     <param name="retry-seconds" value="30"/>
     <!-- Retry parameter, 30 seconds works for us -->
     <param name="caller-id-in-from" value="false"/>
     <!-- Use the callerid of an inbound call in the from field on outbound calls via this gateway. We don't,
up to you. -->
     <param name="contact-params" value="tport=5060"/>
     <!-- Port 5060 is customary, but again, ask the provider -->
     <param name="ping" value="25"/>
     <!-- We like to send a ping just to keep some traffic traversing and keep us registered -->
  </gateway>
</include>
```

That's the whole file. The "name" has to be something DNS can translate or an actual IP address. Clearly, I used the latter. Username and password come from the provider, as do the transport value (could be TCP, ask the provider) and the TPORT value (5060 is typical, but ask the...you get the message). I like the ping value, kinda making sure the link is kept alive in the absence of other traffic.

Now, we want to set something to happen "call-wise". That is, when our LAN phones register with FS and dial a phone number, something good will transpire. Here's one way to get that done: We'll add an "extension" - not to be confused with a physical telephone extension - to what's called a "context". Contexts are just containers for extensions. These context extensions are really directions to FS, telling it how to react when certain things take place. Our "thing" in this case is a registered sip phone dialing a 10 digit number, trying to call someone on the outside.

To make this work, we add the following "extension" to the "context" in the /usr/local/freeswitch/conf/dialplan/default.xml file. Don't confuse this default.xml with the one in /usr/local/freeswitch/conf/directory, different files doing different things. Enter the following in as the first extension in the file, note the <extension> tags that delimit the text you're inputting. What's the point? Well, this paragraph lets FS know what to do when an outgoing call is dialed. Note the <action application> tag. This directs FS to use a specific gateway, the one we just set up, to "get the call out".

```
<! -- Mike stole from elsewhere: Dial any 10 digit number (2223334444) or 1+10 number (12223334444) here -->
   <extension name="UrbanOut">
     <! -- Call this what you want, above. Can be anything that makes sense to you. -->
     <condition field="destination_number" expression="^(1{0,1}\d{10})$">
     <!-- The above translates to: "a 10 digit number, with or without a preceeding 0 or 1" -->
       <action application="set" data="effective_caller_id_number=18155551212"/>
       <!-- Change the above effective_caller_id_number to your own DID. -->
       <!-- If your provider does not provide ringback (180 or 183) you may simulate ringback by uncommenting
the following line. -->
       <!-- action application="ringback" /-->
       <action application="bridge" data="sofia/gateway/xxx.xxx.xxx.xxx/$1"/>
       <!-- Change the above to your own gateway name, between "/gateway/" and "/$1" -->
     </condition>
   </extension>
```

NOTE: One can know if the gateway is registered or not, plus other very useful information, by using the following command at the FS command line:

```
freeswitch@FreeSwitch.local> sofia status
```

Don't forget to hit "enter". That will get you something like the following:

```
API CALL [sofia(status)] output:

    Name                Type                       Data                           State
=================================================================================================
   internal           profile          sip:mod_sofia@192.168.0.199:5060      RUNNING (0)
   external           profile            sip:mod_sofia@yy.yy.yy.yy:5080      RUNNING (0)
   xxx.xxx.xxx.xxx    gateway           sip:8155551212@xxx.xxx.xxx.xxx       REGED
   default              alias                                  internal      ALIASED
   192.168.0.199        alias                                  internal      ALIASED
=================================================================================================
2 profiles 2 aliases
```

Note that my gateway (check the "Type" column) is in state REGED, meaning it is registered with my provider and all is well with the world. If it isn't, check with your provider. If they're nice, they'll look at your registration attempt and provide clues as to what's not right. Hey, its a start.

While were at it, also note that the internal profile is up and running. That's what we messed with in the first place to direct calls to the gateway. Lo, the beautiful symmetry of it all. Further, the internal profile is listening on the LAN address (192.168.0.199 happens to be ours) while the external listens to the outside world on a different Ethernet static IP (it isn't our real one so don't try to hack it) as well as a different UDP port (a logical port for UDP, 5080 for the outside, 5060 for the inside here). All make sense? The alias types are just that, different names for the internal and external profiles. Used for some FS shorthand we won't talk about at the moment.

LETS MAKE SOME CALLS (OUT-GOING)

At this point, your FS has SIP phones registered locally, a SIP/PSTN gateway that it is registered to, and SIP profiles listening for work to do. Your SIP phone - there are many types so I assume you know how to configure and use your device - should itself show as registered to the FS. So, go ahead. Make that call out. I called my cellphone the first time. If all is well (if your output from above looks like it should then all IS well) your outside phone is ringing. Rejoice.

IN-COMING CALLS, RINGING A GROUP

OK, that was fun. But, one would want people to be able to call FS from the outside too. It would also be nice if FS would ring my SIP phone so I could take the call. As a matter of fact, at least in my office, we like all the phones to ring and whoever is available takes the call. One way to set up incoming calls in this way (ring all extensions in a group) is the following:

First, one has to have a group to ring. Forget, for a second, that there are groups already setup in the /usr/local/freeswitch/conf/directory. There is a feature of FS that allows one, from the SIP phone itself, to join a group that FS can then use. Go to each SIP phone in turn and dial extension 8101 (or *8101 on my PfSense version). 81 means "I'm putting this phone into a group". The last two digits tell what group, 01 in this case. You'll hear a tone letting you know it worked. Hang up and go join the next phone to group 01 until you've done 'em all.

NOTE: Want to delete a phone from the group (it will still be able to call out)? Dial extension 8001 (or *8001 in my case on PfSense) and that particular phone will be deleted from the group. Keep in mind that once you are in the group, that group will persist in FS. You can re-register, restart FS or even reboot the computer FS is running on. You have to take action to get an extension out of the group (which is a good thing).

Also note that if you'd like to see how 8001, 8101, or any other "logic" works in FS on your LAN, edit the /usr/local/freeswitch/conf/dialplan/default.xml file. Search for 80, for instance. This file contains "contexts" which themselves containing "extensions". Here, and extension is logic for taking some action when FS sees a particular event take place (like someone dialing 8201). We are at the heart of FS in this file, don't forget about it or what its for.

Now, we have our group, 01 in this case (could be anything between 00 and 99). Dial extension 8201 (or, again, *8201 depending) and all phones joined to group 01 will ring. Fun stuff. Now, let's get FS to use that group to ring the same group. To do so, go to the /usr/local/freeswitch/conf/dialplan/public directory. I created a file that would be read first by using the file name 00_inbound_did.xml. The name is significant, especially the leading 00s. In this case, its the only file here but one could have many. Anyway, enter the following in that file, correcting for your own private info:

```
<include>
  <extension name="Urban1212">
  <!-- Call it what you'd like. -->
    <condition field="destination_number" expression="^(8155551212)$">
    <!-- When FS sees the incoming number above - please use your own DID - it will engage the actions below -->
      <action application="set" data="domain_name=192.168.0.199"/>
      <!-- again, use the domain for your LAN from earlier in this document -->
      <action application="transfer" data="*8201 XML default"/>
      <!-- Here's the group ring. I need the *, you may not. But, ringing 8201, just like from your phone, will
ring the whole group -->
      <!-- The "*8201 XML default" string means: find extension *8201, its in the default.xml file in the
dialplan directory. -->
    </condition>
  </extension>
</include>
```

OK, having done the above, *reloadxml* or stop and start your FS. Then, dial in from a cellphone or other outside line. All phone in your group should ring. The DID of your calling phone should show. Now, we've got something useful going.

Now, this is pretty powerful stuff. You could create another file in /usr/local/freeswitch/conf/dialplan/public and call it 01_inbound_did.xml. There one could set up another group (or single phone) to be rung if one has another DID. Much is possible. But, we'll keep to getting things functional and operational.

MOVING ON TO MORE ADVANCED TOPICS

VOICE MAIL (OR "HOW TO ACCESS SOME THINGS THAT FS IS PREPARED TO DO")

If one has a registered phone (at this point in our document, only possible on the internal profile) then you have a voicemail extension at your disposal too. To check if an extension is in fact registered with your FS, go to your console screen and type:

```
freeswitch@FreeSwitch.local> sofia status profile internal
```

As in the scection on APPLYING YOUR CHANGES AND CHECKING YOUR WORK, one sees output about the internal profile itself, followed by all registered extensions for the profile.

Now, we're going to use the FS dialplan file to figure out how to access voicemail. You'll likely want to do things like create a message for people to hear when they get VM. Also, you will want to hear what messages are left there, deleting some, leaving others. How? We'll answer that question; in the process learning more about FS dialplans, contexts, extensions, and how to understand and use them.

NOTE: You may recall a line in the REGISTERING WITH A SIP PROVIDER section stating in passing that a dial plan contained "contexts". Contexts themselves are containters for instruction sets called "extensions". Don't confuse the term "extension" here with a SIP phone or other device (which has an extension number) and these instruction blocks within contexts.

A CLOSER LOOK AT YOUR DEFAULT DIAL PLAN

Let's edit the /usr/local/freeswitch/conf/dialplan/default.xml file. Near the top of the file, one notes a <context=default> tag. All the way at the bottom of the file, one sees the closing </context> tag. In between these tags, one can see many groupings, deliniated with the <extension="somename"> and </extension> tags. Each of these extensions is actually a reaction to an event. The event being serviced is seen in the <condition> tag set, inside each extension. Usually, that condition is a number that someone has dialed. That could be an outside caller coming into your FS via your gateway provider, or a registered phone on your LAN doing the dialing. When FS "sees" a number, coming or going, it looks at the dial plans, matching that number to each extension's <condition>. FS then performs the <actions> within the matching extension.

NOTE: If the pattern matching syntax within the "expresion=" of the <condition> tag makes no sense, please have a look at "regular expressions" ("regex" for short) and how they work. There's a great regex primer on the freeswitch.org site that's well worth finding and reading it you need it. However, that's outside the scope of this document. Nonetheless, you'll need some familiarity with regex's to understand conditions.

So, now that you're in the file and have the gist of what's going on here, search for the string "voicemail". In mine, courtesy of PfSense (pfsense.org), I find five extensions dealing with voicemail. We'll talk about each a little bit. Once finished, you'll know a lot more about voice mail on my FS. Yours will likely be similar. Here's my five, right from ~/conf/dialplan/default.xml on my system:

```
<!-- voicemail operator extension -->
    <extension name="operator">
      <condition field="destination_number" expression="^\*operator$|^0$">
        <action application="set" data="transfer_ringback=$${hold_music}"/>
        <action application="transfer" data="1000 XML features"/>
      </condition>
    </extension>
```

The "operator" extension above is invoked by the string *operator or a dialed 0. On PfSense, that author likes to use a "*" with special extensions. Others do not. You can set it how you like it. If you have an extension like my "operator", go to a registered phone and hit 0. Note that you'll need a "real" extension 1000 on your LAN to be able to use this. We'll provide a way to put this to work later.

```
<!-- voicemail main2 extension -->
    <extension name="vmain2">
      <condition field="destination_number" expression="^vmain2$|^\*97$|^\*4000$">
        <action application="answer"/>
        <action application="sleep" data="1000"/>
        <action application="voicemail" data="check default ${domain_name}"/>
      </condition>
    </extension>
```

In extension "vmain2", one can type *97, *4000 or a literal vmain2 to access the <action> here. Looking at the last <action> tag, one notes voicemail being invoked to check a user's VM contents. You'll need your user ID and password to gain access. Try this one out, if you have it, and follow the prompts.

```
<!-- voicemail main extension -->
    <extension name="vmain">
      <condition field="destination_number" expression="^vmain$|^\*98$">
        <action application="answer"/>
        <action application="sleep" data="1000"/>
        <action application="voicemail" data="check default ${domain_name} ${sip_from_user}"/>
      </condition>
    </extension>
```

Much like vmai2, you can type *98 (or set it how you'd like it) for this VM entry. Note the <action> here works thus: if you call from your own phone extension, you'll only need the password to gain entry. The invocation of voicemail here uses the number you are dialing from as the user.

```
<extension name="send_to_voicemail_5digits">
      <condition field="destination_number" expression="^\*99(\d{5})$">
        <action application="answer"/>
        <action application="sleep" data="1000"/>
        <action application="set" data="dialed_extension=$1"/>
        <action application="export" data="dialed_extension=$1"/>
        <action application="voicemail" data="default ${domain_name} ${dialed_extension}"/>
      </condition>
    </extension>
```

Above, a handy extension to transfer a caller to a five-digit extension, if you have them. Mine are all four digit, 1000-1019, the defaults. You could change this to handle any extension lenth you like.

```
<extension name="send_to_voicemail_4digits">
      <condition field="destination_number" expression="^\*99(\d{4})$">
        <action application="answer"/>
        <action application="sleep" data="1000"/>
        <action application="set" data="dialed_extension=$1"/>
        <action application="export" data="dialed_extension=$1"/>
        <action application="voicemail" data="default ${domain_name} ${dialed_extension}"/>
      </condition>
    </extension>
```

Finally, an extension so that anyone taking a call can transfer that party to any four digit extension they'd like (much like the 5 digit above this). Just hit "tranfer" on your phone, and then enter *991013 to go to 1013's voicemail without ringing his/her phone. Nice. Remember, if your FS isn't configured as above, you can do so with the extension as you see them here, or by modifiying them to suit yourself.

Now that you can get around in this particular default.xml, have a look at other .xml files in the /usr/local/freeswitch/conf/dialplan directory. Note that we've been here before in the IN-COMING CALLS, RINGING A GROUP section of this document. There, we were directing in-coming calls to the proper default.xml extension via the ~/conf/dialplan/public subdirectory. Hopefully, this is all beginning to make sense as a whole.

So, back to Voicemail. Now, you can access VM from your registered phone, set up your own voice prompt, and administer VM in general. How did we figure that out? By reading the "instructions" in the default dial plan. There's more to know, and a lot more that can be done. The idea is to build on what you've learned up to this point. Now, let's tie that in with calls coming into the system to make our FS even more useful.

INCOMING CALLS RING A GROUP AND GOES TO VM AFTER NO ANSWER

This time, I'm just going to "lay out" some files that are accessed, in order, when an incoming call hits FS. Actually, FS isn't looking at the literal file, it has read them into the running application (assuming you've told it to). If not, FS can't "see" them so be sure you've reloadxml'd or shutdown/started FS. So, have a look and you'll see the flow FS follows to find out what to do. Let's say someone on the "outside" dials us in Illinois at 8158381212.

First, the call "comes in" and FS wants to know where to send it. Since this is a public incoming call, it looks for a match in ~/conf/dialplan/public/ and finds the match - in my case - in 01_incoming_did.xml:

```
<include>
       <extension name="Urban1212">
       <!-- Call it what you'd like. -->
         <condition field="destination_number" expression="^(8155551212)$">
         <!-- When FS sees the incoming number above  please use your own DID  it will engage the actions below
-->
           <action application="set" data="domain_name=192.168.0.199"/>
           <!-- again, use the domain for your LAN from earlier in this document -->
           <action application="transfer" data="8201 XML default"/>
           <!-- Go to this extension, 8201, to find more instructions. -->
         </condition>
       </extension>
</include>
```

Notice above that I changed the transfer extension from *8201 (IN-COMING CALLS, RINGING A GROUP) to a new extension, 8201 (no preceeding *), seen below. I did that so I could add the <actions> that I needed and also demonstrate another way to ring groups, as you will see:

Next, FS looks in ~/conf/dialplan/default.xml, just as it was told to, above, for the 8201 extension. Here it is:

```
<!-- Mike Added: Incoming Urban Calls -->
    <extension name="UrbanIn">
      <condition field="destination_number" expression="^(8201)$">
         <action application="set" data="call_timeout=20"/>
         <action application="set" data="domain_name=192.168.0.199"/>
         <action application="set" data="continue_on_fail=true"/>
         <action application="set" data="hangup_after_bridge=true"/>
         <action application="bridge" data="group/support@${domain_name}"/>
         <action application="answer"/>
         <action application="sleep" data="1000"/>
         <action application="voicemail" data="default 192.168.0.199 1013"/>
      </condition>
    </extension>
```

In this extension, I have the included actions do a few things. I set a timeout value to 20. That means if no one answers in that amount of time (from when the bridge is invoked later on), FS will skip down to the next action, invoking voicemail (via dialplan default.xml) for extension 1013.

Also notice that instead of bridging to *8201, which I could have still done, I opted to use one of the groups already set up for me in ~/conf/directory/default.xml (There's a lot of default.xml files. But, being in different directories, they do completely different things. Actually, one could call them anything you'd like. For the moment, I'm sticking to the names as installed).

Here's a snippet of ~/conf/directory/default.xml. I simply located the <group name> set equal to "support", and added the users I wanted to that group. Note that the same users can be in different groups if you'd like:

```
<group name="support">
      <users>
        <user id="1010" type="pointer"/>
        <user id="1012" type="pointer"/>
        <user id="1013" type="pointer"/>
        <user id="1017" type="pointer"/>
      </users>
</group>
```

Assuming you've set up extension 1013 (the physical phone's voicemail session) with a message about everybody being busy and please leave us a call back number, you're good to go (remember to reloadxml or shutdown/start if you've changed XML files). My phone's (admittedly an older Grandstream GXP-2000) message light blinked red as soon as I called in and left one (I'm 1013). Using the appropriate numbers from A CLOSER LOOK AT YOUR DIAL PLAN, you can now get your VM and deal with it as you'd like.

ADDING ANOTHER DID

Let's say that you would like to have more than one direct inward dial number for your home or business. We saw in the section on REGISTERING WITH A SIP PROVIDER how to have one DID registered. To have more than one, naturally, you will have to have another <gateway> set up as that one is. However, if you want to use the same provider, we'll have to change the syntax of the XML file in /usr/local/freeswitch/conf/sip_profiles/external. Have a look:

my ~/conf/sip_profiles/external/urban.xml file for registering and using one DID:

```
<include>
  <gateway name="xxx.xxx.xxx.xxx">
     <!-- Remember, you'll need this name for your "out-bound calls" extension -->
     <param name="username" value="8155551212"/>
     <!-- Natch', your own username, ours is like a NPANXXTN -->
     <param name="password" value="somekindapassw0rd"/>
     <!-- Natch', your own password -->
     <param name="expire-seconds" value="60"/>
     <!-- Time out value, 60 works for us -->
     <param name="register" value="true"/>
     <!-- You want to register, so "true" -->
     <param name="register-transport" value="udp"/>
     <!-- You'll probably use udp, but ask your provider -->
     <param name="retry-seconds" value="30"/>
     <!-- Retry parameter, 30 seconds works for us -->
     <param name="caller-id-in-from" value="false"/>
     <!-- Use the callerid of an inbound call in the from field on outbound calls via this gateway. We don't, 
up to you. -->
     <param name="contact-params" value="tport=5060"/>
     <!-- Port 5060 is customary, but again, ask the provider -->
     <param name="ping" value="25"/>
     <!-- We like to send a ping just to keep some traffic traversing and keep us registered -->
  </gateway>
</include>
```

Here's the same file, registering with the same provider, but now we have two registrations, one for each DID. Check the comments in the XML code below:

```
<include>
  <gateway name="urban1212">
        <!-- Any name that makes sense -->
        <param name="username" value="8155551212"/>
        <param name="password" value="somekindapassw0rd"/>
        <param name="realm" value="xxx.xxx.xxx.xxx"/>
        <!-- The gateway name from the "One DID" example. -->
        <param name="expire-seconds" value="60"/>>
        <!--/// do not register if value="false"///-->
        <param name="register" value="true"/>
        <param name="register-transport" value="udp"/>
        <param name="retry-seconds" value="30"/>
        <!--Use the callerid of an inbound call in the from field on outbound calls via this gateway -->
        <param name="caller-id-in-from" value="false"/>
        <param name="contact-params" value="tport=5060"/>
        <!--send an options ping every x seconds, failure will unregister and/or mark it down-->
        <param name="ping" value="25"/>
  </gateway>
  <gateway name="urban1234">
        <!-- Any name that makes sense, but not the one from above. The gateway names must be unique. -->
        <param name="username" value="8155551234"/>
        <param name="password" value="anotherpassword1"/>
        <param name="realm" value="xxx.xxx.xxx.xxx"/>
        <!-- The gateway name from the "One DID" example. This is the same as above, but its OK here. -->
        <param name="expire-seconds" value="60"/>>
        <!--/// do not register if value="false"///-->
        <param name="register" value="true"/>
        <param name="register-transport" value="udp"/>
        <param name="retry-seconds" value="30"/>
        <!--Use the callerid of an inbound call in the from field on outbound calls via this gateway -->
        <param name="caller-id-in-from" value="false"/>
        <param name="contact-params" value="tport=5060"/>
        <!--send an options ping every x seconds, failure will unregister and/or mark it down-->
        <param name="ping" value="25"/>
  </gateway>
</include>
```

Notice that I had to use a different gateway name in this example for each DID registration, even though the same provider is registering both DIDs. This was done using the <param name="realm" value=xxx.xxx.xxx.xxx"/> tag. Of course, use your own gatway IP address. If you just double the entries of the single DID example, only one will actually register. You could use the syntax of the two DID example for one DID. As you can see, there's more than one way to do things.

IMPORTANT: If you only make the changes above - assuming you have a working system with the "one DID" setup - you'll find that you can not make outgoing calls anymore (but incoming calls still work). Why? Remember in REGISTERING WITH A SIP PROVIDER, we told FS what to do with out-going calls. That is, to use your newly registered gatwway. In the "two DID" example, we've changed the name of that gateway. Had to, can't use the same one twice. So, pick one of your DID gateway names above and update the extension in ~/conf/dialplan/default.xml for out-going calls (I use the urban1212 gateway, see below).

```
<!-- Mike Stole and Added: Dial any 10 digit number (2223334444) or 1+10 number (12223334444) here -->
    <extension name="UrbanOut">
     <condition field="destination_number" expression="^(1{0,1}\d{10})$">
       <action application="set" data="effective_caller_id_number=18155551212"/>
       <!-- If your provider does not provide ringback (180 or 183) you may simulate ringback by uncommenting
the following line. -->
       <!-- action application="ringback" /-->
       <action application="bridge" data="sofia/gateway/urban1212/$1"/>
     </condition>
    </extension>
```

CHEAT SHEET (OR MORE STUFF FS IS READY TO DO)

AND YET MORE TO COME