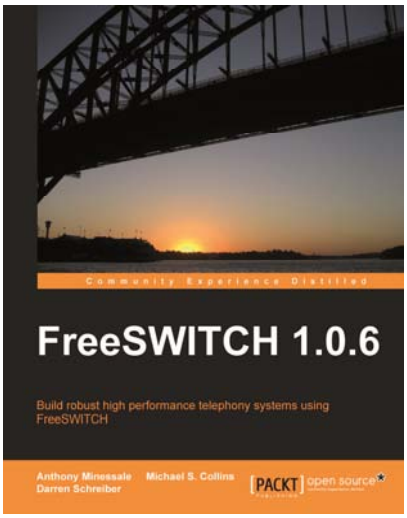
[PACKT] PUBLISHING open source community experience distilled

FreeSWITCH 1.0.6

Anthony Minessale
Michael S. Collins
Darren Schreiber



Chapter No.4
"SIP and the User Directory"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter NO.4 "SIP and the User Directory"

A synopsis of the book's content

Information on where to buy this book

About the Authors

Anthony Minessale has been involved in telephony since 2003. Prior to that, he had spent over ten years as a CTO and platform designer for various companies during the height of the Internet revolution. He has spent over half a decade working as the creator and lead developer of the FreeSWITCH open source project and several years before that as a volunteer developer for the Asterisk open source PBX, and is a noted contributor of several features on that project as well.

Anthony is the creator and owner of FreeSWITCH Solutions LLC, responsible for the popular annual VoIP conference, ClueCon, held every summer in Chicago. Anthony also works for Barracuda Networks as the director of Engineering for the CudaTEL PBX appliance, a product he and his team handcrafted to work with FreeSWITCH as the telephony engine.

I would like to thank the awesome FreeSWITCH community for their dedication to our project and the invaluable feedback they provide on a daily basis. A really big thank you to Michael Collins and Darren Schreiber for helping to put this book together. I would also like to thank Brian West and Michael Jerris for helping make FreeSWITCH even possible with all the time they devote to making it work. Finally I would like to thank the original Asterisk community who inspired us all to relentlessly push forward in open source telephony.

Michael S. Collins is a telephony and open source software enthusiast. He is a PBX veteran, having worked as a PBX technician for five years and as the head of IT for a call center for more than nine years. Michael is an active member of the FreeSWITCH community. He currently works for Barracuda Networks, Inc.

I would like to thank, first and foremost, my wife, Lisa, my daughter, Katherine, and my son, Sean, who keep me going each day. I would also like to thank the many FreeSWITCH experts around the world who are so willing to answer technical questions: Michael Jerris, Moises Silva, Raymond Chandler, Mathieu Réné and more. I would especially like to thank Brian K. West for taking me under his wing and educating me in the ways of VoIP. Finally, I would like to say thank you to Anthony Minessale for authoring an amazing piece of software and inviting me to work with the core FreeSWITCH development team.

Darren A. Schreiber is helping pioneer the way to distributed cloud telephony solutions. He is the founder of the 2600hz Project, the TCAI project, and is the author of the latest version of FreePBX, a popular open source telephony management system. He has worked in Enterprise IT for over 15 years and has lead teams in successful projects in telecom, marketing, and web-based SaaS spaces. He has been a serious telephony enthusiast from a young age and has been working extensively with VoIP technologies for the past seven years. Darren graduated from Rensselaer with a degree in Computer Science and Business Entrepreneurship.

FreeSWITCH 1.0.6

In 1999, the first shot of the telephony revolution was fired when the Asterisk PBX was released to the world. In the ensuing decade, open source telephony took the world by storm, lead by Asterisk and a host of other software packages such as OpenSER and YATE.

In 2006, an Asterisk developer named Anthony Minessale announced an ambitious project: a new telephony software engine, built from the ground up. Some thought this was crazy considering the wild success of the Asterisk platform. However, Anthony's vision was to create a telephony platform unlike any in existence—open source or proprietary. In May 2008, this new project reached a critical milestone with the release of FreeSWITCH 1.0.0.

Now that FreeSWITCH has been available for several years, some developers have migrated from Asterisk to FreeSWITCH. Others have added FreeSWITCH to an existing environment, having it work together with Asterisk, OpenSER, OpenSIPS, Kamailio, and other telephony applications.

Is FreeSWITCH right for you? The correct answer is, of course: It depends. When people ask the FreeSWITCH developers which telephony software they should use, the developers always reply with another correct answer: Use what works for your situation. To know the answer you will need to investigate further.

What FreeSWITCH is and what it is not

FreeSWITCH is a scalable softswitch. In practical terms this means that it can do anything a traditional PBX can do and much more. It can (and does) act as the core switching software for commercial carriers. It can scale up to handle thousands of simultaneous calls. It can also scale down to act as a simple softphone for your laptop or personal computer. It can also work in a cluster of servers.

FreeSWITCH is not a proxy server. If you need proxy server functionality, then consider OpenSIPS, Kamailio, or other similar software. FreeSWITCH is a **back-to-back user agent** or **B2BUA**. In this regard, it is similar to Asterisk and other IP PBX software.

Version and licensing

At the time of this writing this book, the FreeSWITCH developers were putting the finishing touches on FreeSWITCH version 1.2. While the examples presented in this book were specifically tested with version 1.0.6, they have also been confirmed to work with the latest FreeSWITCH development versions that form the basis of version 1.2. Do not be concerned about the fact that this material does

not cover version 1.2—it certainly does. The FreeSWITCH user interface is very stable between versions; therefore, this text will be applicable for years to come.

FreeSWITCH is released under the **Mozilla Public License** (**MPL**) version 1.1. Since FreeSWITCH is a library that can be implemented in other software applications and projects, the developers felt it important to strike a balance between the extremely liberal BSD license and the so-called "viral" GPL. The MPL fits this paradigm well and allows businesses to create commercial products based on FreeSWITCH without licensing concerns.

However, what about using FreeSWITCH with GPL-based software? It should suffice if we said that the developers wanted to make sure that anyone, including proprietary and GPL-based software users, could use FreeSWITCH. The powerful event socket gives us this functionality—a simple TCP socket-based interface that allows an external program to control FreeSWITCH. Regardless of the license you may be using for your own software, you can still connect to a FreeSWITCH server without any licensing issues

What This Book Covers

Chapter 1, Architecture of FreeSWITCH gives a brief, but thorough introduction to the underlying architecture of FreeSWITCH.

Chapter 2, Building and Installation shows how to download and install FreeSWITCH on Windows and Unix-like operating systems.

Chapter 3, Test Driving the Default Configuration provides a hands-on look at the powerful and feature-rich default FreeSWITCH configuration.

Chapter 4, SIP and the User Directory offers an introduction to the concept of users and the directory as well as brief look at SIP user agents

Chapter 5, *Understanding the XML Dialplan* explains the basics of creating and editing Dialplan extensions to add advanced functionality to a FreeSWITCH install.

Chapter 6, Using the Built-In XML IVR Engine discusses how to create menus and sound phrases for interacting with callers.

Chapter 7, Building IVR Applications with Lua introduces the concept of advanced call handling using the lightweight scripting language Lua.

Chapter 8, Advanced Dialplan Concepts builds upon the foundation laid in Chapter 5 and shows how to handle more challenging routing scenarios.

Chapter 9, Controlling FreeSWITCH Externally introduces the incredibly powerful Event Socket and the Event Socket library that can be used to access and control a FreeSWITCH server.

Chapter 10, Advanced Features and Further Reading highlights some of the more powerful FreeSWITCH features like conferencing and offers some ideas on where to learn more about FreeSWITCH.

Appendix A, The FreeSWITCH Online Community gives a brief introduction to the worldwide online community and the tools used to stay in contact.

Appendix B, The History Of FreeSWITCH is a description of how FreeSWITCH came to be from one of the authors, Anthony Minessale.

4

SIP and the User Directory

In the previous chapter, we briefly introduced SIP, the Session Initiation Protocol, where we discussed how to register a telephone with FreeSWITCH. In this chapter, we will build upon that foundation and learn more about how we use SIP to connect users, both locally and around the world. SIP is a ubiquitous protocol in the VoIP landscape. In this chapter, we will:

- Learn the principle behind the FreeSWITCH user Directory
- Explore and configure the FreeSWITCH user Directory for the first time
- Learn how to connect FreeSWITCH to service providers
- Make modifications to the Dialplan and directory XML configuration
- Briefly discuss SIP profiles and User Agents

Understanding the FreeSWITCH user directory

The FreeSWITCH user directory is based on a centralized XML document, comprised of one or more `<domain>` elements. Each `<domain>` can contain either `<user>` elements or `<groups>` elements. A `<groups>` element contains one or more `<group>` elements, each of which contains one or more `<user>` elements. A small, simple example would look like the following:

```
<section name="directory">
  <domain name="example.com">
    <groups>
      <group name="default">
        <user id="1001">
          <params>
            <param name="password" value="1234"/>
```

```
            </params>
          </user>
        </group>
      </groups>
    </domain>
  </section>
```

Some more basic configurations may not have a need to organize the users in groups so it is possible to omit the `<groups>` element completely, and just insert several `<user>` elements into the top `<domain>` element.

The important thing is that each `user@domain` derived from this directory is available to all components in the system—it's a single centralized directory for storing all of your user information. If you register as a user with a SIP phone or if you try to leave a *voicemail* message for a user, FreeSWITCH looks in the same place for user data. This is important because it limits duplication of data, and makes it more efficient than it would be if each component kept track of its users separately.

This system should work well for a small system with a few users in it, but what about a large system with thousands of users? What if a user wants to connect his existing database to FreeSWITCH to provide the user directory? Well, using `mod_xml_curl` that we discussed in the first chapter, we can create a web service that gets the request for the entries in the user directory, in the same way a web page sends the results of a form submission. In turn, that web service can query an existing database of users formatted any way possible, and construct the XML records in the format that FreeSWITCH registry expects. `mod_xml_curl` returns the data to the module requesting the lookup. This means that instant, seamless integration with your existing setup is possible; your data is still kept in its original, central location.

The user directory can be accessed by any subsystem within FreeSWITCH. This includes modules, scripts, and the FSAPI interface among others. In this chapter, we are going to learn how the Sofia SIP module employs the user directory to authenticate your softphone or hardware SIP phone. If you are a developer you may appreciate some nifty things you can do with your user directory, such as adding a `<variables>` element to either the `<domain>`, the `<group>`, or the `<user>` element. In this element you can set many `<variable>` elements, allowing you to set channel variables that will apply to every call made by a particular authenticated user. This can come in very handy in the Dialplan because it allows you to make user-specific routing decisions. It is also possible to define IP address ranges using CIDR notation, which can be used to authenticate particular users based on what remote network address they connect from. This removes the need for a login and password, if your user always logs in from the same remote IP address.

Authentication versus authorization

Authentication is the process of identifying a user. **Authorization** is the process of determining the level of access of a user. Authentication answers the question, "Is this person really who he says he is?" Authorization answers the question, "What is this person allowed to do here?" When you see expressions such as "IP Auth" and "Digest Auth", remember that they are referring to the two primary ways of identifying (that is, authenticating) a user. IP authorization is based upon the user's IP address. Digest authentication is based upon the user supplying a username and password. SIP (and FreeSWITCH) can use either method. Visit `http://en.wikipedia.org/wiki/Digest_access_authentication` for a discussion of how digest authentication works.

The directory is implemented in pure XML. This is advantageous for several reasons, not the least of which is the "X" in XML: Extensible. Since XML is, by definition, extensible, the directory structure is also extensible. If we need to add a new element into the directory, we can do so simply by adding to the existing XML structure.

Working with the FreeSWITCH user directory

The default configuration has one domain with a directory of 20 users. Users can be added or removed very easily. There is no set limit to how many users can be defined on the system. The list of users is collectively referred to as the **directory**. Users can belong to one or more **groups**. Finally, all the users belong to a single *domain*. By default, the **domain** is the IP address of the FreeSWITCH server.

In the following sections we will discuss these topics:

- User features
- Adding a user
- Testing voicemail
- Groups of users

User features

Let's begin by looking at the XML file that defines a user. Locate the file `conf/directory/default/1000.xml` and open it in an editor. You should see a file like the following:

```
<include>
  <user id="1000">
    <params>
      <param name="password" value="$${default_password}"/>
      <param name="vm-password" value="1000"/>
    </params>
    <variables>
      <variable name="toll_allow"
      value="domestic,international,local"/>
      <variable name="accountcode" value="1000"/>
      <variable name="user_context" value="default"/>
      <variable name="effective_caller_id_name" value="Extension
      1000"/>
      <variable name="effective_caller_id_number" value="1000"/>
      <variable name="outbound_caller_id_name"
      value="$${outbound_caller_name}"/>
      <variable name="outbound_caller_id_number"
      value="$${outbound_caller_id}"/>
      <variable name="callgroup" value="techsupport"/>
    </variables>
  </user>
</include>
```

The XML structure of a user is simple. Within the `<include>` tags the user has the following:

- The `user` element with the `id` attribute
- The `params` element, wherein parameters are specified
- The `variables` element, wherein channel variables are defined

Even before we know what much of the specifics mean, we can glean from this file that the user `id` is `1000` and that there is both a `password` and a `vm-password`. In this case, the password parameter refers to the SIP authorization password. (We discussed this under the *Configuring a SIP phone to work with FreeSWITCH* section in Chapter 3.) The expression `$${default_password}` refers to the value contained in the global variable `default_password` which is defined in the `conf/vars.xml` file. If you surmised that `vm-password` means "voicemail password" then you are correct. This value refers to the digits that the user needs to dial when logging in to check his or her voicemail messages. The value of `id` is used both as the authorization username and the SIP username.

Additionally, there are a number of channel variables that are defined for this user. Most of these are directly related to the default Dialplan. The following table lists each variable and what it is used for:

Variable	Purpose
`toll_allow`	Specifies which types of calls this user can make
`accountcode`	Arbitrary value that shows up in CDR data
`user_context`	The Dialplan context that is used when this person makes a phone call
`effective_caller_id_name`	Caller ID name displayed on called party's phone when calling another registered user
`effective_caller_id_number`	Caller ID number displayed on called party's phone when calling another registered user
`outbound_caller_id_name`	Caller ID name sent to provider on outbound calls
`outbound_caller_id_number`	Caller ID number sent to provider on outbound calls
`callgroup`	Arbitrary value that can be used in Dialplan or CDR

In summary, a user in the default configuration has the following:

- A username for SIP and for authorization
- A voicemail password
- A means of allowing/restricting dialling
- A means of handling caller ID being sent out
- Several arbitrary variables that can be used or ignored as needed

Let's now add a new user to our directory.

Adding a user

Adding one or more users is a simple two-step process, which is as follows:

- Create a new XML file for the user, usually by copying an existing file
- Modify the `Local_Extension` Dialplan entry

In this example, we will create a new user for a person named "Gwen" and a username of "1100". Follow these steps:

- Open a terminal window, change directory to `conf/directory/default`
- Make a copy of `1000.xml` and name it `1100.xml`. A Linux/Unix session looks like the following:

```
#>cd /usr/local/freeswitch/conf/directory/default
#>cp 1000.xml 1100.xml
```

- Open `1100.xml` in an editor and make the following changes:
  - Replace all occurrences of "1000" with "1100"
  - Change the value of `effective_caller_id_name` to "Gwen"

The new file should look like the following:

```
<include>
  <user id="1100">
    <params>
      <param name="password" value="$${default_password}"/>
      <param name="vm-password" value="1100"/>
    </params>
    <variables>
      <variable name="toll_allow"
      value="domestic,international,local"/>
      <variable name="accountcode" value="1100"/>
      <variable name="user_context" value="default"/>
      <variable name="effective_caller_id_name" value="Gwen"/>
      <variable name="effective_caller_id_number" value="1100"/>
      <variable name="outbound_caller_id_name"
      value="$${outbound_caller_name}"/>
      <variable name="outbound_caller_id_number"
      value="$${outbound_caller_id}"/>
      <variable name="callgroup" value="techsupport"/>
    </variables>
  </user>
</include>
```

Save the file. Next, we need to edit the Dialplan entry for `Local_Extension`. Open `conf/dialplan/default.xml` in an editor and locate the following lines:

```
<extension name="Local_Extension">
  <condition field="destination_number" expression="^(10[01][0-9])$">
```

This Dialplan extension, as its name implies, routes calls to local extensions. In our case, a local extension is a phone registered to a user in our directory. Recall that FreeSWITCH comes with 20 directory users predefined, numbered 1000 through 1019. This extension corresponds to those 20 users. By default, any call made to 1000, 1001, … 1019 will be handled by the `Local_Extension` Dialplan entry. We need to add "1100" to the regular expression. Edit the `expression` value so that it looks like the following:

```
^(10[01][0-9]|1100)$
```

Save the file. (Regular expressions are discussed in greater detail in *Chapter 5*, *Understanding the XML Dialplan*.)

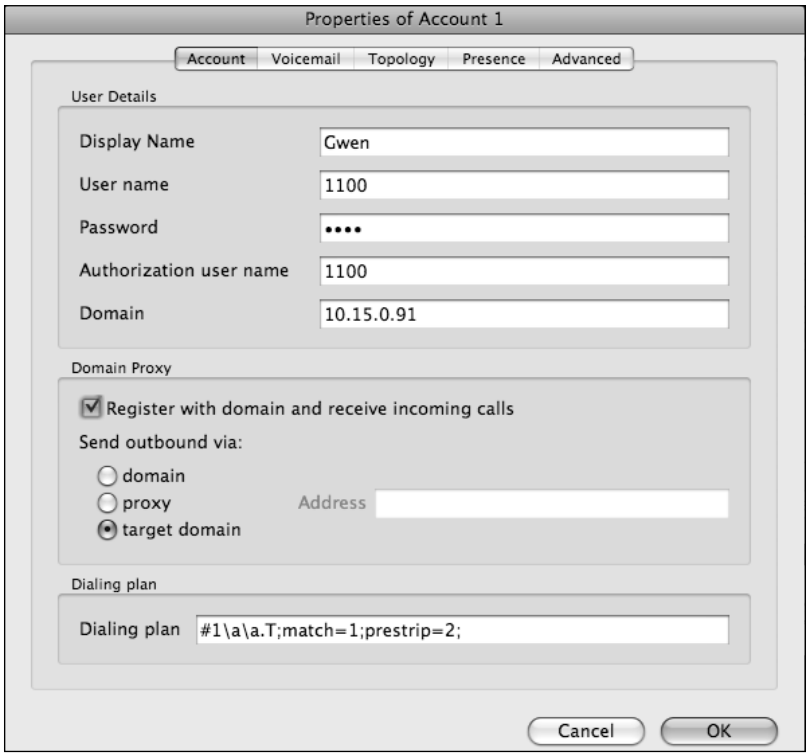
The last thing we need to do is reload the XML configuration. Launch `fs_cli` and issue the `reloadxml` command as follows:

```
freeswitch@internal> reloadxml
+OK [Success]

freeswitch@internal> 2009-11-20 16:47:36.986620 [INFO] mod_enum.c:808
ENUM Reloaded
2009-11-20 16:47:36.986620 [INFO] switch_time.c:661 Timezone reloaded 530
definitions
```

Linux/Unix users can save time by opening two terminal windows. Run `fs_cli` in one window and your editor in the other. For advanced handling of multiple windows check out the GNU Screen utility. For more information check: `http://www.gnu.org/software/screen`.

Our new extension has been defined, and we should now be able to register a SIP phone to user 1100. Using the methods described in *Chapter 3*, *Test Driving the Default Configuration*, register a SIP phone to user 1100. An X-Lite configuration looks like this:



The registered phone can now make outbound calls, and can receive inbound calls from those who dial 1100.

To see which SIP phones are registered issue this command at the FreeSWITCH command line: `sofia status profile internal`.

Now that we have successfully added a user, let's test a common feature: voicemail.

Testing voicemail

Each user in the directory has a voice "mailbox" where others can leave voice messages. By default, unanswered calls to a user will go to the user's voicemail after 30 seconds. Make a test call to confirm that everything is working. Dial the destination extension and let it ring. After about 30 seconds the voicemail system will answer; record a message of at least three seconds (the minimum message length), and then hang up. (If you have only one phone for testing, then try dialing your own extension.) The user's phone will now have a message-waiting indicator. An X-Lite softphone with a message waiting looks like the following image:

Notice the envelope icon and the red telephone icon. These indicate a new message waiting and a missed call, respectively.

Save time when leaving a voice message by pressing #, to skip past the user's outbound greeting.

Retrieving the message is also simple: Dial *98 or *4000*. The voicemail system will guide you through logging in and listening to new or saved messages. A typical session would look like the following:

4000

"Welcome to your voicemail. Please enter your ID, followed by the pound sign."

1100#

"Please enter your password, followed by the pound sign."

1100#

"You have one new message."

When a user has a new message, the system will automatically play it along with the date and time that the message was left. The default voicemail menus are configured as follows:

Main Menu

- 1—Listen to new messages
- 2—Listen to saved messages
- 5—Options menu (recorded name, greeting, and so on)
- #—Exit voicemail

While Listening to a Message

- 1—Replay message from the beginning
- 2—Save message
- 4—Rewind
- 6—Fast-forward
- 0—Pause playback

After Listening to a Message

1—Replay message from the beginning
2—Save message
4—Send to e-mail (requires configuration)
7—Delete message
8—Undelete message

Feel free to try out some of these options. Log in to your voicemail and record an outbound greeting. By default you can record up to ten different greetings; however, most users only record a single greeting. Typically we will use greeting number one.

All of the voicemail options are customizable. Look in the file `conf/autoload_configs/voicemail.conf.xml`. You can edit the default voicemail profile or even create your own custom voicemail profiles.

Now that we have voicemail working, we can concentrate on one other useful feature: groups of users.

Groups of users

Larger installations frequently need the ability to dial multiple telephones. For example, a department in a company might have several users, all of whom are responsible for answering calls to that department. At the same time, they each have their own extension number, so they may individually receive calls. FreeSWITCH has a directory feature that allows users to be grouped together. A user can belong to multiple groups.

Some PBX systems employ an advanced form of inbound call routing called **ACD** or **Automatic Call Distribution**. Call groups are not used for this kind of application. Although it is beyond the scope of this publication, FreeSWITCH users wanting advanced functionality are encouraged to investigate FIFO queues. See `http://wiki.freeswitch.org/wiki/Mod_fifo` for more information.

Groups are defined in the file `conf/directory/default.xml`. Open the file and locate the `groups` node. Notice that there are four groups already defined. They are as follows:

- Default—All users in the directory
- Sales—1000 to 1004
- Billing—1005 to 1009
- Support—1010 to 1014

The latter three groups are merely arbitrarily defined groups that can be modified or removed as needed. The `default` group, though, is a bit more interesting. It contains every user in the directory. (Use with caution!) Let's add a new group and then examine how groups work:

1. Add the following lines inside the groups node:

```
<group name="custom">
  <users>
    <user id="1000" type="pointer"/>
    <user id="1100" type="pointer"/>
  </users>
</group>
```

2. If you have two or more telephones registered then use their extension numbers instead of 1000 and 1100. Save the file.
3. Launch `fs_cli` and issue the `reloadxml` command.

Confirm that the new custom group has been added by using the `group_call` command. Your output should be similar to the following:

```
freeswitch@internal> group_call custom
[presence_id=1000@10.15.0.91]error/user_not_registered,[presence_
id=1100@10.15.0.91]sofia/internal/sip:1100@10.15.0.136:31354;rinstance=5e
39ede358ffc0c8
```

What significance does this chunk of apparently random gibberish hold? The `group_call` command is used to create a *dialstring* for calling multiple telephones. In our example, user 1000 is not registered and therefore would not receive a call. (Hence the "error" of `user_not_registered`.) However, user 1100 is indeed registered. If a user in a group is not registered, then when the group is called, that user is effectively ignored. Before we can call our new group we need to add it to the Dialplan, as follows:

1. Open `conf/dialplan/default.xml` and locate the `group_dial_billing` extension shown as follows:

```
<extension name="group_dial_billing">
  <condition field="destination_number" expression="^2002$">
    <action application="bridge"
      data="group/billing@${domain_name}"/>
  </condition>
</extension>
```

2. Insert the following new lines after the `</extension>` tag of the `group_dial_billing` extension:

```
<extension name="group_dial_custom">
  <condition field="destination_number" expression="^2003$">
    <action application="bridge"
      data="group/custom@${domain_name}"/>
  </condition>
</extension>
```

3. Save the file.
4. Launch `fs_cli` and issue the `reloadxml` command.
5. Test your group by dialing *2003*. All the extensions in your group should ring.

When all of the phones in a group are ringing, the first one to answer will "win" and receive the call. All the other phones will stop ringing.

We have seen how we can connect telephones to FreeSWITCH, as well as the many features they have. Now let's discuss how to make phone calls outside the local FreeSWITCH server.

Connecting to the world with gateways

The counterpart to having a user register to your FreeSWITCH server is to have your server register as a user on a remote server. This is accomplished using **gateways**. A gateway is quite simply an outbound registration to another SIP server. Telephone service providers use very large servers to provide SIP *trunks* to their subscribers. In FreeSWITCH, we can use a gateway to connect to a SIP provider. We can also use a gateway to connect to another SIP server, such as another FreeSWITCH server or any SIP-compliant IP-PBX.

Setting up a new gateway

A gateway simply connects to a SIP server just like a SIP phone connects to FreeSWITCH. As such, a gateway configuration bears some resemblance to a SIP phone configuration. Like a SIP phone registering to FreeSWITCH, a gateway has some minimum requirements. They are as follows:

- Username and password
- Server address or IP, and port

These values are supplied by the service provider. Occasionally there are other parameters, like a proxy server and port. If you already have an account with a SIP provider then you can use it for your gateway. In this example, we will use an account from `iptel.org`.

Visit `http://www.iptel.org` to sign up for a free SIP account.

To add a new gateway, follow these steps:

1. Create a new XML file in `conf/sip_profiles/external`. This example will use `iptel.org.xml`. Add the following lines, inserting the proper values for your provider:

```
<include>
  <gateway name="iptel">
  <param name="username" value="MY_USER_NAME"/>
  <param name="password" value="MY_PASSWORD"/>
  <param name="realm" value="iptel.org"/>
  <!-- iptel.org requires a 'proxy' parameter -->
  <param name="proxy" value="sip.iptel.org"/>
  </gateway>-->
</include>
```

2. Save the file and then launch `fs_cli`.
3. Issue the command `/log 6` to decrease the verbosity of debug messages.
4. Simply reloading the XML configuration will not add the new gateway. Issue the following command: `sofia profile external restart reloadxml`. The output will look as follows:

```
freeswitch@internal> sofia profile external restart reloadxml

Reload XML [Success]

restarting: external

freeswitch@internal> 2009-11-21 16:29:23.509986 [INFO] mod_enum.
c:808 ENUM Reloaded

2009-11-21 16:29:23.511578 [INFO] switch_time.c:661 Timezone
reloaded 530 definitions

2009-11-21 16:29:24.118566 [NOTICE] sofia_reg.c:85 UN-Registering
iptel

2009-11-21 16:29:24.713768 [NOTICE] sofia.c:1218 Waiting for
worker thread
```

```
2009-11-21 16:29:24.713768 [NOTICE] sofia_glue.c:3690 deleted
gateway example.com
2009-11-21 16:29:24.713768 [NOTICE] sofia_glue.c:3690 deleted
gateway iptel
2009-11-21 16:29:24.713768 [NOTICE] sofia_reg.c:2237 Added gateway
'iptel' to profile 'external'
2009-11-21 16:29:24.713768 [NOTICE] sofia_reg.c:2237 Added gateway
'example.com' to profile 'external'
2009-11-21 16:29:24.713768 [NOTICE] sofia.c:3149 Started Profile
external [sofia_reg_external]
2009-11-21 16:29:25.736445 [NOTICE] sofia_reg.c:333 Registering
iptel
```

5. Confirm that the gateway is registered properly. Issue the command `sofia status`. The output should look similar to the following:

```
freeswitch@internal> sofia status
                        Name          Type                                Data      State
=================================================================================================
                external       profile      sip:mod_sofia@10.15.0.91:5080      RUNNING   (0)
             example.com       gateway            sip:joeuser@example.com      NOREG
                   iptel       gateway         sip:MY_USER@sip.iptel.org       REGED
                internal       profile      sip:mod_sofia@10.15.0.91:5060      RUNNING   (0)
            internal-ipv6      profile           sip:mod_sofia@[::1]:5060      RUNNING   (0)
              10.15.0.91         alias                           internal      ALIASED
=================================================================================================
3 profiles 1 alias
```

The gateway's state should be `REGED`. If it says something else, like `FAIL_WAIT`, then most likely there is a configuration problem. Confirm your settings and try again.

Restarting a profile will disconnect all active calls that are currently routed through that profile. An alternate command to add a newly created gateway without restarting the entire profile is:
`sofia profile external rescan reloadxml`

Now that our gateway is added, we need to modify the Dialplan so that we can make and receive calls.

Making calls

We will make a simple Dialplan entry that sends calls out our new gateway. Our new extension will accept the digit 9 and then the digit 1, followed by exactly ten more digits representing the telephone number to be dialed. (In a production environment there are many other possible strings, which can even be alphanumeric. Some of these will be considered in *Chapter 5*, *Understanding the XML Dialplan*.)

To get started with making outbound calls, add your new extension to the Dialplan, following these steps:

1. Create a new file in `conf/dialplan/default` named `01_custom.xml`.
2. Add the following text to the file:

```
<include>
  <extension name="Dial Out Custom Gateway">
    <condition field="destination_number"
      expression="^9(1\d{10})$">
      <action application="bridge"
        data="sofia/gateway/custom/$1"/>
    </condition>
  </extension>
</include>
```

3. Save the file. Launch `fs_cli` and issue the `reloadxml` command.

The new extension is now ready to be tested. From a phone that is registered to FreeSWITCH, dial 9 plus a ten-digit phone number. For example, dial *9*, *1-800-555-1212*. It may take a moment for the call to be established. Confirm that audio is flowing in both directions and that each party can hear the other. If audio is flowing in only one direction, then most likely there is a problem with the NAT device on your local network.

Receiving calls

Generally, when you register your gateway with a SIP provider, the provider allows you to receive calls. (Telephones that register with FreeSWITCH are an example of this.) By default, FreeSWITCH treats incoming calls as inherently untrusted, even if they come from the opposite end of a registered gateway. These calls come into the "public" Dialplan context. From there they can be discarded or routed as needed. Let's set up a simple Dialplan entry to handling inbound calls to our `iptel.org` account.

1. Create a new file in `conf/dialplan/public` named `01_iptel.xml`.
2. Add these lines to the file, using your account name as follows:

```
<include>
  <extension name="iptel-inbound">
    <condition field="destination_number"
    expression="^(MY_IPTEL_USERNAME)$">
      <action application="set" data="domain_name=$${domain}"/>
      <action application="bridge" data="1000 XML default"/>
    </condition>
  </extension>
</include>
```

3. Save the file. Launch `fs_cli` and execute the `reloadxml` command.

Inbound calls will now be routed to extension 1000. You can route calls to any valid extension, including all the extensions we tested in *Chapter 3, Test Driving the Default Configuration*.

Making calls without a gateway

Sometimes it is not necessary to use a gateway. For example, not all services require digest authorization. An example of this is the FreeSWITCH public conference server. In fact, the default Dialplan contains an extension for dialling the conference: *9888*. (Actually, there are several different conference "rooms" on the public FreeSWITCH conference server.) Let's look at this extension. Open `conf/dialplan/default.xml` in an editor and locate the `freeswitch_public_conf_via_sip` extension. Note the `bridge` line:

```
<action application="bridge" data="sofia/${use_profile}/$1@conference.
freeswitch.org"/>
```

The value in `${use_profile}` defaults to "internal" (as defined in `conf/vars.xml`). When a user dials *9888* the dialstring that is sent out is actually as follows:

```
sofia/internal/888@conference.freeswitch.org
```

Notice that there is no mention of a gateway. Instead, FreeSWITCH simply sends the call out to the `internal` SIP profile. In other words, the local FreeSWITCH server sends a call to `conference.freeswitch.org` without actually authorizing it. This is possible because the server at `conference.freeswitch.org` does not require authorization for incoming calls. (This is where the gateway comes in—if the target server issues a challenge then the gateway will respond to that challenge with authorization credentials, namely the username and password.)

Not all SIP providers explicitly require digest authorization of calls; some perform IP authorization instead. In those cases you do not need to create a gateway. Instead, simply send the call out a SIP profile. Usually the internal SIP profile is sufficient for these kinds of calls.

SIP profiles and user agents

Before we finish our discussion of SIP and the user directory, it would be good to touch upon a subject that some users initially find a bit daunting: SIP profiles. In the strictest sense of the word, a SIP profile in FreeSWITCH is a *User Agent*. In practical terms, this means that each SIP profile "listens" on a particular IP address and port number. The `internal` profile listens on port 5060, and the `external` profile listens on port 5080. Not only does the profile listen but it can respond as well. For example, when a phone sends a SIP REGISTER packet to FreeSWITCH (at port 5060), the `internal` profile "hears" the registration request and acts accordingly. The files in `conf/sip_profiles/` are ones which determine how the profiles behave. Many of the parameters in these profiles are to customize how FreeSWITCH handles various SIP traffic scenarios. In most cases the defaults are reasonable and should work. In other cases, though, you may find that because of the peculiarities in various VoIP phones and vendors, you will need to make adjustments.

Lastly, do not let the profile names `internal` and `external` be a source of confusion. Each profile is simply a user agent that is streamlined for a specific purpose. The `internal` profile is optimized to handle telephone registrations and calls between registered phones. The `external` profile is optimized for outbound gateway connections and several NAT traversal scenarios.

For a deeper discussion of user agents and the concept of a back-to-back user agent (B2BUA) see `http://en.wikipedia.org/wiki/Back-to-back_user_agent`.

Summary

In this chapter, we discussed the following:

- How FreeSWITCH collects users into a directory
- How FreeSWITCH uses a VoIP protocol, SIP, to connect users to each other, and to the world
- SIP is similar to e-mail in that it has users and domains
- Employing various user features like voicemail
- Adding a new user and modifying the Dialplan accordingly
- Connecting to the outside world with gateways
- SIP profiles and user agents

In this chapter, we made some minor modifications to the default XML Dialplan, and we learned how to set up users and domains within the XML user directory. Now that we have a general understanding of how these modifications work, we will continue to build upon this foundation. We will now begin to form a much more detailed understanding of FreeSWITCH as we further explore the XML Dialplan module, the default and most commonly used call routing engine available in FreeSWITCH.

Where to buy this book

You can buy Oracle FreeSWITCH 1.0.6from the Packt Publishing website: `https://www.packtpub.com/freeswitch-1-0-5-build-robust-high-performance-telephony-systems/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information:
www.PacktPub.com/freeswitch-1-0-5-build-robust-high-performance-telephony-systems/book