FreeSWITCH XML Dialplan elements

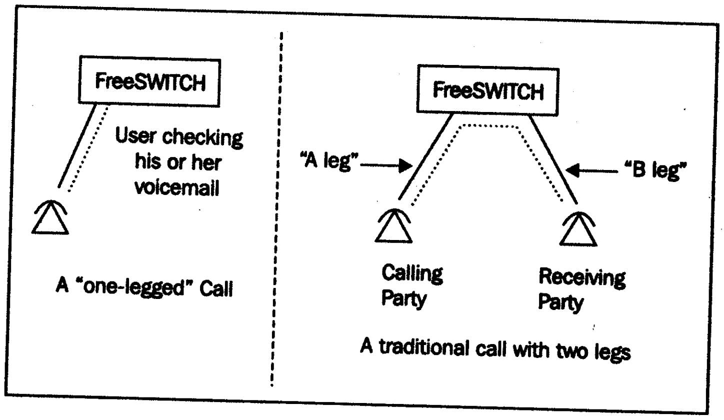
The default FreeSWITCH XML Dialplan is contained in three main files and two directories, located in `conf/dialplan/`:

- `default.xml` — The primary FreeSWITCH Dialplan configuration
- `public.xml` — Handles calls coming in to FreeSWITCH from another locati
- `features.xml` — A special context for handling specific dialing features
- `default/` — Files in this directory get included in the default context
- `public/` — Files in this directory get included in the public context

The default XML configuration has many instructions for routing calls, all of which make use of the basic building blocks of a Dialplan: contexts, extensions, conditions and actions. A context is a logical grouping of one or more extensions. An extensio contains one or more conditions that must be met. Conditions contain actions that will be performed on the call, depending on the whether the condition is met or not. Before discussing these building blocks further, though, let's revisit some of the concepts we first considered in *Chapter 3, Test Driving the Default Configuration.*

Call legs and channel variables

Phone calls to and from FreeSWITCH consist of one or more call legs. A "one-legged" connection might be something like a user dialing into his or her voicemail. A traditional call between two parties is a connection with two call legs. Recall the following illustration from *Chapter 3, Test Driving the Default Configuratio*



A call between two different telephones consists of an **A-leg** (calling or originating party) and a **B-leg** (receiving party). Each call leg is also known as a **channel**, as in an audio channel. Each channel has a set of logical attributes, what you might call a list of facts about that particular call leg. Each of these attributes is stored in a corresponding channel variable. In the previous chapter, we learned that a registered user has several channel variables defined, and these variables are included in call legs involving that user. To get an idea of just how much information is available for a call, you can call the `information` extension at *9192* following the steps below:

1. Launch `fs_cli` and issue the command `/log 6`
2. From a registered phone dial *9192*

You will see dozens of lines of information. Following is an excerpt from an `info` dump:

```
2009-12-09 13:50:40.685247 [INFO] mod_dptools.c:916 CHANNEL_DATA:
Channel-State: [CS_EXECUTE]
Channel-State-Number: [4]
Channel-Name: [sofia/internal/1001@10.15.0.94]
Unique-ID: [dd9fbb65-971e-4b65-9f2e-586bde83a6bf]
Call-Direction: [inbound]
Presence-Call-Direction: [inbound]
Channel-Presence-ID: [1001@10.15.0.94]
Answer-State: [answered]
Channel-Read-Codec-Name: [PCMU]
Channel-Read-Codec-Rate: [8000]
Channel-Write-Codec-Name: [PCMU]
Channel-Write-Codec-Rate: [8000]
Caller-Username: [1001]
Caller-Dialplan: [XML]
Caller-Caller-ID-Name: [Michael Collins]
Caller-Caller-ID-Number: [1001]
Caller-Network-Addr: [10.15.0.124]
Caller-ANI: [1001]
Caller-Destination-Number: [9192]
...
variable_sip_received_ip: [10.15.0.124]
variable_sip_received_port: [29935]
variable_sip_via_protocol: [udp]
variable_sip_authorized: [true]
variable_sip_number_alias: [1001]
```

```
variable_sip_auth_username: [1001]
variable_sip_auth_realm: [10.15.0.94]
variable_number_alias: [1001]
variable_toll_allow: [domestic,international,local]
variable_accountcode: [1001]
variable_user_context: [default]
variable_effective_caller_id_name: [Michael Collins]
variable_effective_caller_id_number: [Michael Collins]
...
```

The lines beginning with "variable_" show the values in the respective channel variables. For example, the line `variable_sip_authorized: [true]` is showing that the value of `sip_authorized` channel variable is "true". You will also notice that there are numerous other data elements such as `Unique-ID` and `Call-Direction`. These are **info application variables**. Most (but not all) of these are available as read-only values, which can be accessed just like channel variables.

Accessing channel variables

Within the Dialplan, variables are accessed with a special notation: `${variable_name}`. Consider the following example:

```
<action application="log" data="INFO The value in sip_authorized is
  '${sip_authorized}' "/>
```

This action would print out a log message to the FreeSWITCH command such as the following:

```
2009-12-09 14:32:48.904383 [INFO] mod_dptools.c:897 The value in sip_
authorized is 'true'
```

Accessing the read-only values is much the same. Each of these values has a corresponding channel variable name. For example:

```
<action application="log" data="INFO The value of Unique-ID is
  '${uuid}' "/>
```

This will print a log line on the FreeSWITCH command line such as the following:

```
2009-12-09 14:46:31.695458 [INFO] mod_dptools.c:897 The value of Unique-
ID is '169ae42e-29f5-4e1c-9505-8ee6ef643081'
```

A complete list of `info` application variables and their corresponding channel variable names can be found at the following address: `http://wiki.freeswitch.org/wiki/Channel_Variables#variable_xxxx`.

In the preceding example, the extension will log some information to the FreeSWITCH command line depending upon what the user dialed. If the user dials *9101*, the action is executed and the log displays, "You dialed 9101". If the user dials anything other than 9101, then the anti-action is executed and the log displays, "You did NOT dial 9101".

Most extensions you create (and indeed, in the default Dialplan) will have many actions but few anti-actions. In most cases, actions execute Dialplan *applications* which in turn may accept *arguments*. In the preceding example, the `log` application is executed and the `data` attribute contains the argument passed to it.

How Dialplan processing works

Understanding the Dialplan is easier if you can visualize what happens when a call comes in. Often, we hear expressions like "the call traverses the Dialplan" or "the call hits the Dialplan". What exactly does that mean? Let's walk through the processing of a call, so that we can really understand what XML Dialplan is doing.

The Dialplan has two phases: parsing and executing. The Dialplan parser looks for extensions to execute. When it finds a matching extension, it then adds the actions (or anti-actions) to a list of tasks. When the parser finishes looking for extensions, the execution phase begins, and the actions in the task list are performed.

A good way to see all of this in action is to watch the FreeSWITCH console in debug mode while making a test phone call. Launch `fs_cli`, make a test call to 9196 (music on hold), and then hang up the phone. Scroll back in your terminal and look for a line that looks something like the following:

```
2009-12-09 22:23:16.727746 [INFO] mod_dialplan_xml.c:408 Processing Test
User->9196 in context default
```

This is the start of the Dialplan processing. A telephone whose user is named "Test User" has dialed 9196. (Your console will display the name of the user associated with the phone from which you dialed.) The lines following begin with "Dialplan:" and are debug messages, showing which extensions matched and which ones did not. The first extension parsed is called "unloop". It is an important extension, but is not very interesting for our Dialplan discussion. Look down to the next extension that gets parsed. In our example, *call the debug* output is as follows:

```
Dialplan: sofia/internal/1001@10.15.0.91 parsing [default->tod_example]
continue=true
Dialplan: day of week[4] =~ 2-6 (PASS)
Dialplan: hour[22] =~ 9-18 (FAIL)
Dialplan: sofia/internal/1001@10.15.0.91 Date/Time Match (FAIL) [tod_
example] break=on-false
```

The extension `tod_example` (time of day example) is shown being parsed. These debug lines correspond to the `tod_example` extension found in `conf/dialplan/default.xml`:

```
<extension name="tod_example" continue="true">
  <condition wday="2-6" hour="9-18">
    <action application="set" data="open=true"/>
  </condition>
</extension>
```

This extension simply checks the time of the day and the day of the week. If the call is made on a weekday (Monday through Friday) during business hours, (9 AM to 6 PM) then it sets the channel variable `open` to "true". This call was made on a Wednesday at 10:23 PM. Therefore, it passed the `wday` (day of week) test but not the `hour` (hour of day) test. Had the call been made between 9 AM and 6 PM then both conditions would have been met, and the set application would have been added to the task list. Notice that the `tod_example` extension has `continue="true"`. This means that the Dialplan will continue parsing extensions even if `tod_example` matches.

The parser continues trying to match extensions, most of which fail:

```
Dialplan: sofia/internal/1001@10.15.0.91 parsing [default->global-
intercept] continue=false
Dialplan: sofia/internal/1001@10.15.0.91 Regex (FAIL) [global-intercept]
destination_number(9196) =~ /^886$/ break=on-false
Dialplan: sofia/internal/1001@10.15.0.91 parsing [default->group-
intercept] continue=false
Dialplan: sofia/internal/1001@10.15.0.91 Regex (FAIL) [group-intercept]
destination_number(9196) =~ /^\*8$/ break=on-false
Dialplan: sofia/internal/1001@10.15.0.91 parsing [default->intercept-ext]
continue=false
Dialplan: sofia/internal/1001@10.15.0.91 Regex (FAIL) [intercept-ext]
destination_number(9196) =~ /^\*\*(\d+)$/ break=on-false
Dialplan: sofia/internal/1001@10.15.0.91 parsing [default->redial]
continue=false
Dialplan: sofia/internal/1001@10.15.0.91 Regex (FAIL) [redial]
destination_number(9196) =~ /^870$/ break=on-false
Dialplan: sofia/internal/1001@10.15.0.91 parsing [default->global]
continue=true
Dialplan: sofia/internal/1001@10.15.0.91 Regex (FAIL) [global] ${call_
debug}(false) =~ /^true$/ break=never
Dialplan: sofia/internal/1001@10.15.0.91 Regex (FAIL) [global] ${sip_
has_crypto}() =~ /^(AES_CM_128_HMAC_SHA1_32|AES_CM_128_HMAC_SHA1_80)$/
break=never
```