CHAPTER 4

Initial Configuration of Asterisk

Perseverance is the hard work you do after you get tired of doing the hard work you already did.

—Newt Gingrich

The purpose of this chapter is to guide the user through the configuration of four channels: a Foreign eXchange Office (FXO) channel, a Foreign eXchange Station (FXS) channel, a Session Initiation Protocol (SIP) channel, and an Inter-Asterisk eXchange protocol (IAX)* channel. The purpose is not to give an exhaustive survey of all channel types or topologies, but rather to provide a base platform on which to build your telecommunications system. Further scenarios and channel configuration details can be found in Appendix D. We start by exploring the basic configuration of analog interfaces such as FXS and FXO ports with the use of a Digium Dev-Lite kit. We'll then configure two Voice over Internet Protocol (VoIP) interfaces: a local SIP channel connected to a soft phone, and a connection to Free World Dialup via IAX.

Once you've worked through this chapter, you will have a basic system consisting of many useful interfaces, and you will be ready to learn more about the *extensions.conf* file (discussed further in Chapter 5), which contains the instructions Asterisk needs to build the dialplan.

What Do I Really Need?

The asterisk character (*) is used as a wildcard in many different applications. It is the perfect name for this PBX for many reasons, one of which is the enormous number of interface types to which Asterisk can connect. These include:

- Analog interfaces, such as your telephone line and analog telephones
- Digital circuits, such as T-1 and E-1 lines
- VoIP protocols such as SIP and IAX

* Officially, the current version is IAX2, but all support for IAX1 has been dropped, so whether you say "IAX" or "IAX2," it is expected that you are talking about Version 2.

Asterisk doesn't need any specialized hardware—not even a sound card. Channel cards that connect Asterisk to analog phones or phone lines are available, but not essential. You can connect to Asterisk using the soft phones that are available for Windows, Linux, and other operating systems without using a special hardware interface. You can also use any IP phone that supports either SIP or IAX2. On the other side, if you don't connect directly to an analog phone line from your central office, you can route your calls over the Internet to a telephony service provider.

Working with Interface Configuration Files

In this chapter, we're finally going to "get our hands dirty" and start building an Asterisk configuration. For the first few sections on FXO and FXS channels, we'll assume that you have the Digium Dev-Lite kit with one FXO and one FXS interface, which allows you to connect to an analog phone line (FXO) and to an analog phone (FXS). Note that this hardware interface isn't necessary; if you want to build an IP-only configuration, you can skip to the section on configuring SIP.

The configuration we do in this chapter won't be particularly useful on its own, but it will be a kernel to build on. We're going to touch on the following files:

zaptel.conf
: Here, we'll do low-level configuration for the hardware interface. We'll set up one FXO channel and one FXS channel.

zapata.conf
: In this file, we'll configure Asterisk's interface to the hardware.

extensions.conf
: The dialplans we create will be extremely primitive, but they will prove that the system is working.

sip.conf
: This is where we'll configure the SIP protocol.

iax.conf
: This is where we'll configure incoming and outgoing IAX channels.

In the following sections, you will be editing several configuration files. You'll have to reload these files for your changes to take effect. After you edit the *zaptel.conf* file, you will need to reload the configuration for the hardware with `/sbin/ztcfg -vv` (you may omit the `–vv` if you don't need verbose output). Changes made in *zapata.conf* will require a `reload` from the Asterisk console; however, changing signaling methods requires a `restart`. You will need to perform a `reload chan_iax2.so` and a `reload chan_sip.so` after editing the *iax.conf* and *sip.conf* files, respectively.

FXO and FXS Channels

The difference between an FXO channel and an FXS channel is simply which end of the connection provides the dial tone. An FXO port does not generate a dial tone; it accepts one. A common example is the dial tone provided by your phone company. An FXS port provides both the dial tone and ringing voltage to alert the station user of an inbound call. Both interfaces provide bidirectional communication (i.e., communication that is transmitted and received in both directions simultaneously).

If your Asterisk server has a compatible FXO port, you can plug a telephone line from your telephone company (or "telco") into this port. Asterisk can then use the telco line to place and receive telephone calls. By that same token, if your Asterisk server has a compatible FXS port, you may plug an analog telephone into your Asterisk server, so that Asterisk may call the phone and you may place calls.

Ports are defined in the configuration by the signaling they use, as opposed to the physical type of port they are. For instance, a physical FXO port will be defined in configuration with FXS signaling, and an FXS port will be defined with FXO signaling. This can be confusing until you understand the reasons for it. FX_ cards are named not according to what they are, but rather according to what is connected to them. An FX*S* card, therefore, is a card that connects to a *s*tation. Since that is so, you can see that in order to do its job, an FXS card must *behave* like a central office and use FXO signaling. Similarly, an FX*O* card connects to a central *o*ffice (CO), which means it will need to behave like a station and use FXS signaling. The modem in your computer is a classic example of an FXO device.

> The older X100P card used a Motorola chipset, and the X101P (which Digium sold before completely switching to the TDM400P) is based on the Ambient/Intel MD3200 chipset. These cards are modems with drivers adapted to utilize the card as a single FXO device (the telephone interface cannot be used as an FXS port). Support for the X101P card has been dropped in favor of the TDM series of cards. Use of these cards (or their clones) is not recommended in production environments.

Determining the FXO and FXS Ports on Your TDM400P

Figure 4-1 contains a picture of a TDM400P with an FXS module and an FXO module. You can't see the colors, but module 1 is a green FXS module and module 2 is an orange/red FXO module. In the bottom-right corner of the picture is the Molex connector, where power is supplied from computer's power supply.

> Plugging an FXS port (the green module) into the PSTN may destroy the module and the card!

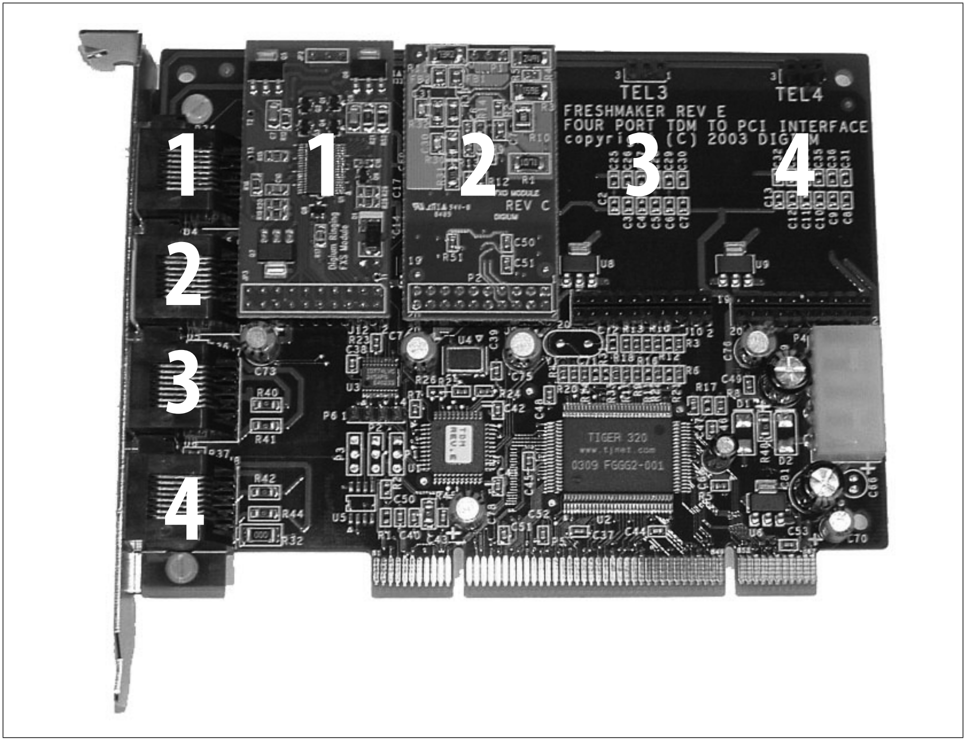


Figure 4-1. A TDM400P with an FXS module (1 across) and an FXO module (2 across)

Be sure to connect your computer's power supply to the Molex connector on the TDM400P if you have FXS modules, as it is used to generate the voltage to produce ringing on the phone. The Molex connector is not required if you have only FXO modules.

Configuring an FXO Channel

We'll start by configuring an FXO channel. First we'll configure the Zaptel hardware, and then the Zapata hardware. We'll set up a very basic dialplan, and we'll show you how to test the channel.

Zaptel Hardware Configuration

The *zaptel.conf* file located in */etc/* is used to configure your hardware. The following minimal configuration defines an FXO port with FXS signaling:

```
fxsks=2
loadzone=us
defaultzone=us
```

In the first line, in addition to indicating whether we are using FXO or FXS signaling, we specify one of the following protocols for channel 2:

- Loop start (`ls`)
- Ground start (`gs`)
- Kewlstart (`ks`)

The difference between *loop start* and *ground start* has to do with how the equipment requests a dial tone: a ground start circuit signals the far end that it wants a dial tone by momentarily grounding one of the leads; a loop start circuit uses a short to request a dial tone. Though not common for new installations, analog ground start lines still exist in many areas of the country.[*] For example, ground start lines are predominately used to reduce a condition known as "glare"[†] that is associated with loop start lines and PBXs with high call volumes. All home lines (and analog telephones/modems/faxes) in North America use loop start signaling. *Kewlstart* is in fact the same as loop start, except that it has greater intelligence and is thus better able to detect far-end disconnects.[‡] Kewlstart is the preferred signaling protocol for analog circuits in Asterisk.

To configure a signaling method other than kewlstart, replace the `ks` in `fxsks` with either `ls` or `gs` (for loop start or ground start, respectively).

`loadzone` configures the set of indications (as configured in *zonedata.c*) to use for the channel. The *zonedata.c* file contains information about all the various sounds that a phone system makes in a particular country: dial tone, ringing cycles, busy tone, and so on. When you apply a loaded tone zone to a Zap channel, that channel will mimic the indications for the specified country. Different indication sets can be configured for different channels. The `defaultzone` is used if no zone is specified for a channel.

After configuring *zaptel.conf*, you can load the drivers for the card. `modprobe` is used to load modules for use by the Linux kernel. For example, to load the *wctdm* driver, you would run:

```
# modprobe wctdm
```

[*] Yes, there is such a thing as ground start signaling on channelized T-1s, but that has nothing to do with an actual ground condition on the circuit (which is entirely digital).

[†] When a call is initiated from one end of a circuit at the same approximate time a call is initiated from the opposite end of the circuit.

[‡] A *far-end disconnect* happens when the far end hangs up. In an unsupervised circuit, there is no method of telling the near end that the call has ended. If you are on the phone this is no problem, since you will know the call has ended and will manually hang up your end. If, however, your voicemail system is recording a message, it will have no way of knowing that the far end has terminated and will thus keep recording silence, or even the dial tone or reorder tone. Kewlstart can detect these conditions and disconnect the circuit.

If the drivers load without any output, they have loaded successfully.* You can verify that the hardware and ports were loaded and configured correctly with the use of the *ztcfg* program:

```
# /sbin/ztcfg -vv
```

The channels that are configured and the signaling method being used will be displayed. For example, a TDM400P with one FXO module has the following output:

```
Zaptel Configuration
======================


Channel map:

Channel 02: FXS Kewlstart (Default) (Slaves: 02)

1 channels configured.
```

If you receive the following error, you have configured the channel for the wrong signaling method:

```
ZT_CHANCONFIG failed on channel 2: Invalid argument (22)
Did you forget that FXS interfaces are configured with FXO signalling
and that FXO interfaces use FXS signalling?
```

To unload drivers from memory, use the `rmmod` (remove module) command, like so:

```
# rmmod wctdm
```

The *zttool* program is a diagnostic tool used to determine the state of your hardware. After running it, you will be presented with a menu of all installed hardware. You can then select the hardware and view the current state. A state of "OK" means the hardware is successfully loaded:

```
Alarms          Span
OK              Wildcard TDM400P REV E/F Board 1
```

Zapata Hardware Configuration

Asterisk uses the *zapata.conf* file to determine the settings and configuration for telephony hardware installed in the system. The *zapata.conf* file also controls the various features and functionality associated with the hardware channels, such as Caller ID, call waiting, echo cancellation, and a myriad of other options.

When you configure *zaptel.conf* and load the modules, Asterisk is not aware of anything you've configured. The hardware doesn't have to be used by Asterisk; it could very well be used by another piece of software that interfaces with the Zaptel

* It is generally safe to assume that the modules have loaded successfully, but to view the debugging output when loading the module, check the console output (by default this is located on TTY terminal 9, but this is configurable in the *safe_asterisk* script—see the previous chapter for details).

modules. You tell Asterisk about the hardware and control the associated features via *zapata.conf*:

```
[trunkgroups]
; define any trunk groups

[channels]
; hardware channels
; default
usecallerid=yes
hidecallerid=no
callwaiting=no
threewaycalling=yes
transfer=yes
echocancel=yes
echotraining=yes

; define channels
context=incoming        ; Incoming calls go to [incoming] in extensions.conf
signalling=fxs_ks       ; Use FXS signalling for an FXO channel
channel => 2            ; PSTN attached to port 2
```

The `[trunkgroups]` section is for NFAS and GR-303 connections, and it won't be discussed in this book. If you require this type of functionality, see the *zapata.conf.sample* file for more information.

The `[channels]` section determines the signaling method for hardware channels and their options. Once an option is defined, it is inherited down through the rest of the file. A channel is defined using `channel =>`, and each channel definition inherits all the options defined above that line. If you wish to configure different options for different channels, remember that the options should be configured *before* the `channel =>` definition.

We've enabled Caller ID with `usecallerid=yes` and specified that it will not be hidden for outgoing calls with `hidecallerid=no`. Call waiting is deactivated on an FXO line with `callwaiting=no`. Enabling three-way calling with `threewaycalling=yes` allows an active call to be placed on hold with a hook switch flash (discussed in Chapter 7) to suspend the current call. You may then dial a third party and join them to the conversation with another hook switch. The default is to not enable three-way calling.

Allowing call transfer with a hook switch is accomplished by configuring `transfer=yes`; it requires that three-way calling be enabled. The Asterisk echo canceller is used to remove the echo that can be created on analog lines. You can enable the echo canceller with `echocancel=yes`. The echo canceller in Asterisk requires some time to learn the echo, but you can speed this up by enabling echo training (`echotraining=yes`). This tells Asterisk to send a tone down the line at the start of a call to measure the echo, and therefore learn it more quickly.

When a call comes in on an FXO interface, you will want to perform some action. The action to be performed is configured inside a block of instructions called a *context*. Incoming calls on the FXO interface are directed to the `incoming` context with `context=incoming`. The instructions to perform inside the context are defined within *extensions.conf*.

Finally, since an FXO channel uses FXS signaling, we define it as such with `signalling=fxs_ks`.

Dialplan Configuration

The following minimal dialplan makes use of the `Echo()` application to verify that bidirectional communications for the channel are working:

```
[incoming]
; incoming calls from the FXO port are directed to this context from zapata.conf
exten => s,1,Answer()
exten => s,2,Echo()
```

Whatever you say, the `Echo()` application will relay back to you.

Dialing in

Now that the FXO channel is configured, let's test it. Run the *zttool* application and connect your PSTN line to the FXO port on your TDM400P. Once you have a phone line connected to your FXO port, you can watch the card come out of a RED alarm.

Now dial the PSTN number from another external phone (such as a cell phone). Asterisk will answer the call and execute the `Echo()` application. If you can hear your voice being reflected back, you have successfully installed and configured your FXO channel.

Configuring an FXS Channel

The configuration of an FXS channel is similar to that of an FXO channel. Let's take a look.

Zaptel Hardware Configuration

The following is a minimal configuration for an FXS channel on a TDM400P. The configuration is identical to the FXO channel configuration above, with the addition of `fxoks=1`.

Recall from our earlier discussion that the opposite type of signaling is used for FXO and FXS channels, so we will be configuring FXO signaling for our FXS channel. In the example below we are configuring channel 1 to use FXO signaling, with the kewlstart signaling protocol:

```
fxoks=1
fxsks=2
loadzone=us
defaultzone=us
```

After loading the drivers for your hardware, you can verify their state with the use of `/sbin/ztcfg -vv`:

```
Zaptel Configuration
======================


Channel map:

Channel 01: FXO Kewlstart (Default) (Slaves: 01)
Channel 02: FXS Kewlstart (Default) (Slaves: 02)

2 channels configured.
```

Zapata Hardware Configuration

The following configuration is identical to that for the FXO channel, with the addition of a section for our FXS port and of the line `immediate=no`. The context for our FXS port is `internal`, the signaling is `fxoks` (kewlstart), and the channel number is set to `1`.

FXS channels can be configured to perform one of two different actions when a phone is taken off the hook. The most common (and often expected) option is for Asterisk to produce a dial tone and wait for input from the user. This action is configured with `immediate=no`. The alternative action is for Asterisk to automatically perform a set of instructions configured in the dialplan instead of producing a dial tone, which you indicate by configuring `immediate=yes`.[*] The instructions to be performed are found in the context configured for the channel and will match the `s` extension (both of these topics will be discussed further in the following chapter).

Here's our new *zapata.conf*:

```
[trunkgroups]
; define any trunk groups

[channels]
; hardware channels
; default
usecallerid=yes
hidecallerid=no
callwaiting=no
threewaycalling=yes
transfer=yes
echocancel=yes
echotraining=yes
immediate=no
```

[*] Also referred to as the BatPhone method, and more formally known as an Automatic Ringdown or Private Line Automatic Ringdown (PLAR) circuit. This method is commonly used at rental car counters and airports.

```
; define channels
context=internal        ; Uses the [internal] context in extensions.conf
signalling=fxo_ks       ; Use FXO signalling for an FXS channel
channel => 1            ; Telephone attached to port 1

context=incoming        ; Incoming calls go to [incoming] in extensions.conf
signalling=fxs_ks       ; Use FXS signalling for an FXO channel
channel => 2            ; PSTN attached to port 2
```

Dialplan Configuration

To test our newly created Zap extension, we need to create a basic dialplan. The following dialplan contains a context called `internal`. This is the same context name that we configured in *zapata.conf* for channel 1. When we configure `context=internal` in *zapata.conf*, we are telling Asterisk where to look for instructions when a user presses digits on his telephone. In this case, the only extension number that will work is 611. When you dial 611 on your telephone, Asterisk will execute the `Echo( )` application so that when you talk into the phone whatever you say will be played back to you, thereby verifying bidirectional voice.

The dialplan looks like this:

```
[internal]
exten => 611,1,Answer( )
exten => 611,2,Echo( )
```

Configuring SIP

The Session Initiation Protocol (SIP), often used in VoIP phones (either hard phones or soft phones), takes care of the setup and teardown of calls, along with any renegotiations during a call. Basically, it helps two endpoints talk to each other (if possible, *directly* to each other). SIP does not carry media; rather, it uses the Real-time Transport Protocol (RTP) to transfer the media[*] directly between phone A and phone B once the call has been set up.

SIP and RTP

SIP is an application-layer signaling protocol that uses the well-known port 5060 for communications. SIP can be transported with either the UDP or TCP transport-layer protocols. Asterisk does not currently have a TCP implementation for transporting SIP messages, but it is possible that future versions may support it (and patches to the code base are gladly accepted). SIP is used to “establish, modify, and terminate

[*] We use the term *media* to refer to the data transferred between endpoints and used to reconstruct your voice at the other end. It may also refer to music or prompts from the PBX.

multimedia sessions such as Internet telephony calls."[*] SIP does not transport media between endpoints.

RTP is used to transmit media (i.e., voice) between endpoints. RTP uses high-numbered, unprivileged ports in Asterisk (10,000 through 20,000, by default).

A common topology to illustrate SIP and RTP, commonly referred to as the "SIP trapezoid," is shown in Figure 4-2. When Alice wants to call Bob, Alice's phone contacts her proxy server, and the proxy tries to find Bob (often connecting through his proxy). Once the phones have started the call, they communicate directly with each other (if possible), so that the data doesn't have to tie up the resources of the proxy.

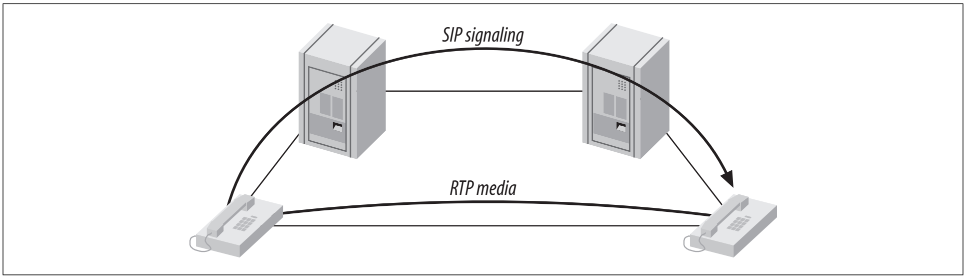


Figure 4-2. The SIP trapezoid

SIP was not the first, and is not the only, VoIP protocol in use today (others include H.323, MGCP, IAX, and so on), but currently it seems to have the most momentum with hardware vendors. The advantages of the SIP protocol lie in its wide acceptance and architectural flexibility (and, we used to say, simplicity!).

SIP Configuration

Here is a basic *sip.conf* file:

```
[general]
context=default
srvlookup=yes

[john]
type=friend
secret=welcome
qualify=yes                     ; Qualify peer is no more than 2000 ms away
nat=no                          ; This phone is not natted
host=dynamic                    ; This device registers with us
canreinvite=no                  ; Asterisk by default tries to redirect
context=internal                ; the internal context controls what we can do
```

[*] RFC 3261, SIP: Session Initiation Protocol, p. 9, Section 2.

The *sip.conf* file starts with a [general] section, which contains the channel settings and default options for all *users* and *peers* defined within *sip.conf*. You can override the default settings on a per-user/peer basis by configuring them within the user/peer definition.

Domain Name System Service records (DNS SRV records) are a way of setting up a logical, resolvable address where you can be reached. This allows calls to be forwarded to different locations without the need to change the logical address. By using SRV records, you gain many of the advantages of DNS, whereas disabling them breaks the SIP RFC and removes the ability to place SIP calls based on domain names. (Note that if multiple records are returned, Asterisk will use only the first.) DNS SRV record lookups are disabled by default in Asterisk, but it's highly recommended that you turn them on. To enable them, set srvlookup=yes in the [general] section of *sip.conf*.

Each connection is defined as a user, peer, or friend. A user type is used to authenticate incoming calls, a peer type is used for outgoing calls, and a friend type is used for both. The extension name is defined within square brackets ([]). In this case, we have defined the extension john as a friend.

A secret is a password used for authentication. Our secret is defined as welcome. We can monitor the latency between our Asterisk server and the phone with qualify=yes, thereby determining whether the remote device is reachable. qualify=yes can be used to monitor any end device, including other Asterisk servers. By default, Asterisk will consider an extension reachable if the latency is less than 2,000 ms (2 seconds). You can configure the time Asterisk should use when determining whether or not a peer is reachable by replacing yes with the number of milliseconds.

If an extension is behind a device performing Network Address Translation (NAT), such as a router or firewall, configure nat=yes to force Asterisk to ignore the contact information for the extension and use the address from which the packets are being received. Setting host=dynamic will require the extension to register so that Asterisk knows how to reach the phone. To limit an endpoint to a single IP address or fully qualified domain name (FQDN), replace dynamic with the IP address or domain name. Note that this limits only where you place calls *to*, as the user is allowed to place calls *from* anywhere (assuming she has authenticated successfully). If you set host=static, the end device is not required to register.

We've also set canreinvite=no. In SIP, *invites* are used to set up calls and to redirect media. Any invite issued after the initial invite in the same dialog is referred to as a *reinvite*. For example, suppose two parties are exchanging media traffic. If one client goes on hold and Asterisk is configured to play Music on Hold (MoH), Asterisk will issue a reinvite to the secondary client, telling it to redirect its media stream toward the PBX. Asterisk is then able to stream music or an announcement to the on-hold client.

The primary client then issues an off-hold command in a reinvite to the PBX, which in turn issues a reinvite to the secondary party requesting that it redirect its media stream toward the primary party, thereby ending the on-hold music and reconnecting the clients.

Normally, when two endpoints set up a call they pass their media directly from one to the other. Asterisk generally breaks this rule by staying within the media path, allowing it to listen for digits dialed on the phone's keypad. This is necessary because if Asterisk cannot determine the call length, inaccurate billing can occur. Configuring `canreinvite=no` forces Asterisk to stay in the media path, not allowing RTP messages to be exchanged directly between the endpoints.

Asterisk will not issue a reinvite in any of the following situations:

- If either of the clients is configured with `canreinvite=no`
- If the clients cannot agree on a common set of codecs and Asterisk needs to perform codec conversion
- If either of the clients is configured with `nat=yes`
- If Asterisk needs to listen to Dual Tone Multi-Frequency (DTMF) tones during the call (for transfers or any other features)

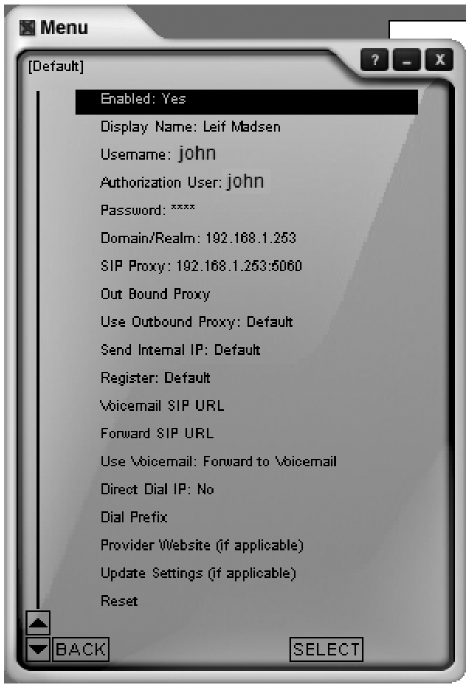
Lastly, `context=internal` specifies the location of the instructions used to control what the phone is allowed to do, and what to do with incoming calls for this extension. The context name configured in *sip.conf* matches the name of the context in *extensions.conf*, which contains the instructions. More information about contexts and dialplans will be presented in the following chapter.

If you are configuring a number of clients with similar configurations, you can place like commands under the `[general]` heading. Asterisk will use the defaults specified in the `[general]` section unless they are explicitly changed within a client's configuration block.

Client Configuration

While it would be impossible to show all the possible configurations for all the end devices that can communicate with Asterisk, we feel it beneficial to provide the configuration for at least one free soft phone, which you can use in determining if Asterisk is right for your organization. We've chosen to use X-ten's X-Lite client, which you can download from their web site (*http://www.xten.com*).

The configuration of the client is generally straightforward. The most important parts are the username and password for registration, plus the address of the Asterisk server with which you wish to register. Figure 4-3 shows a sample configuration for the X-Lite client. Be sure to modify the values of the fields to reflect your configuration.



[Default]

Enabled: Yes
Display Name: Leif Madsen
Username: john
Authorization User: john
Password: ****
Domain/Realm: 192.168.1.253
SIP Proxy: 192.168.1.253:5060
Out Bound Proxy
Use Outbound Proxy: Default
Send Internal IP: Default
Register: Default
Voicemail SIP URL
Forward SIP URL
Use Voicemail: Forward to Voicemail
Direct Dial IP: No
Dial Prefix
Provider Website (if applicable)
Update Settings (if applicable)
Reset

Figure 4-3. X-Lite soft phone configuration screen

The display name is the string that will be used for Caller ID. The username and authorization user are used for authentication, along with the password. The domain/realm should be the IP address or FQDN of your Asterisk server. The SIP proxy is the same as the one entered for the domain/realm, but with `:5060` appended (this specifies the port number to use for SIP signaling—be sure it matches the port you have configured in *sip.conf*).

After entering all this information, verify that `Enabled` is set to `Yes`, and then close the configuration menu. X-Lite will then register to Asterisk. If X-Lite doesn't appear to register, simply restart the client. Because X-Lite is minimized to the task tray when you close the application with the X button, you will need to exit the program by right-clicking on the icon in the tray and then clicking "Exit" in the pop-up menu before restarting.

Dialplan Configuration

Many SIP phones, both soft and hard, are multi-line phones. This means they can accept multiple incoming calls at the same time. Thus, to test your X-Lite soft phone you can simply call yourself, and the call will loop back from the Asterisk server and onto line two of the client. To call yourself, dial extension 100. If your preferred client doesn't support multi-line functionality, you can use extension 611 to enter the `Echo( )` test application.

```
[internal]
exten => 100,1,Dial(SIP/john)
exten => 611,1,Echo( )
```

Configuring Inbound IAX Connections

The Inter-Asterisk eXchange (IAX) protocol is usually used for server-to-server communication; more hard phones are available that talk SIP. However, there are several soft phones that support the IAX protocol, and work is progressing on several fronts for hard phone support in firmware. The primary difference between the IAX and SIP protocols is the way media (your voice) is passed between endpoints.

With SIP, the RTP (media) traffic is passed using different ports than those used by the signaling methods. For example, Asterisk receives the signaling of SIP on port 5060 and the RTP (media) traffic on ports 10,000 through 20,000, by default. The IAX protocol differs in that both the signaling and media traffic are passed via a single port: 4569. An advantage to this approach is that the IAX protocol tends to be better suited to topologies involving NAT.

An IAX *user* is used to authenticate and handle calls coming into the PBX system. For calls going out from the PBX, Asterisk uses an IAX `peer` entry in the *iax.conf* file to authenticate with the remote end. (IAX peers will be explored in the section "Configuring Outbound IAX Connections.")

This section explores the configuration of your system for a Free World Dialup (FWD) account via IAX. Free World Dialup is a free VoIP service provider that allows you to connect to any other member of the network, regardless of physical location, for free. FWD is also connected to over 100 other networks to which you can connect for free.

Be sure to enable IAX2 support for your FWD account before you get started by visiting *http://www.fwdnet.net/index.php?section_id=112*.

This section sets up *iax.conf* and *extensions.conf* to allow you to accept calls from another FWD user. The section on outgoing IAX connections deals with placing calls.

iax.conf Configuration

In *iax.conf*, sections are defined with a name enclosed in square brackets (`[ ]`). Every *iax.conf* file needs at least one main section: `[general]`. Within the `[general]` section, you define the settings related to the use of the IAX protocol, such as default codecs and jitter buffering. You can override the default codecs you specify in the [general] section by specifying them within the user or peer definitions.

The following [general] section is the default from the *iax.conf.sample* configuration file (the same file that's installed when you perform a make samples). For more information about the options, see Appendix A.

```
[general]
bandwidth=low
disallow=lpc10
jitterbuffer=no
forcejitterbuffer=no
tos=lowdelay
autokill=yes

register => fwd_number:password@iax2.fwdnet.net

[iaxfwd]
type=user
context=incoming
auth=rsa
inkeys=freeworlddialup
```

Within the [general] section, you'll need to add a register statement. The purpose of the register statement is to tell the FWD IAX server where you are on the Internet (your IP address). When a call is placed to your FWD number, the FWD servers do a lookup in their database and forward the call to the IP address associated with the FWD number.

In the [iaxfwd] section, define the user for incoming calls with type=user. Then define where the incoming call will be handled within the dialplan, with context=incoming. To specify that the authentication for the incoming call will be done with an RSA public/private key pair, use auth=rsa. The public key is defined with inkeys=freeworlddialup. The freeworlddialup public key comes standard with Asterisk.

Dialplan Configuration

Handling an incoming call in the *extensions.conf* file is simple. First, create a context called incoming (the same context name configured for the iaxfwd user in *iax.conf*). The context is followed by a Dial( ) statement that will dial the SIP extension created earlier in this chapter. Replace the number 10001 with that of your FWD account:

```
[incoming]
exten => 10001,1,Dial(SIP/john)
```

Configuring Outbound IAX Connections

While an IAX *user* receives inbound calls; an IAX *peer* is used to place outbound calls. This section will set up *iax.conf* and *extensions.conf* so that you can place calls.

iax.conf Configuration

The following entry in *iax.conf* can be used to place a call on the FWD network:

```
[iaxfwd]
type=peer
host=iax2.fwdnet.net
username=<fwd-account-number>
secret=<fwd-account-password>
qualify=yes
disallow=all
allow=ulaw
allow=gsm
allow=ilbc
allow=g726
```

A peer is defined with `type=peer`. Use `host` to configure the server through which you will place calls (`iax2.fwdnet.net`). Your FWD account number and password will be used for authentication to the FWD network and are defined respectively with `username` and `secret`.

You can use the `qualify=yes` statement to occasionally check that the remote server is responding. The response time (latency) can be viewed from the Asterisk console with `iax2 show peers`. By default, a peer is considered unreachable after 2000 ms (2 seconds). You can customize the time period by replacing `yes` with the number of milliseconds.

The available codecs and the order of preference can be defined on a per-peer basis. `disallow=all` is used to reset any codec settings set previously. You can then `allow` the codecs you support and set their preference (from top to bottom), using the syntax `allow=`*`codec`*.

Use the `iax2 show registry` command from the Asterisk CLI to verify that you've registered successfully.

Dialplan Configuration

Let's define a section in *extensions.conf* so that we can place a call to the FWD echo test application. As in previous configurations, we will create a context, followed by the instructions to connect to the FWD echo test. Use either your telephone attached to the FXS port or your SIP phone to place the call by dialing 613.

```
[internal]
exten => 613,1,Dial(IAX2/iaxfwd/613)
```

,ch04.20428 Page 75 Wednesday, August 31, 2005 4:55 PM

Debugging

Several methods of debugging are available in Asterisk. Once you've connected to the console, you can enable different levels of verbosity and debugging output, as well as protocol packet tracing. We'll take a look at the various options in this section. (The Asterisk console is discussed in more detail in Appendix E.)

Connecting to the Console

To connect to the Asterisk console, you can either start the server in the console directly (in which case you will not be able to exit out of the console without killing the Asterisk process), or start Asterisk as a daemon and then connect to a remote console.

To start the Asterisk process directly in the console, use the console flag:

```
# /usr/sbin/asterisk -c
```

To connect to a remote Asterisk console, start the daemon first, then connect with the `–r` flag:

```
# /usr/sbin/asterisk
# /usr/sbin/asterisk -r
```

If you are having a problem with a specific module not loading, or a module causing Asterisk to not load, start the Asterisk process with the `–c` flag to monitor the status of modules loading. For example, if you attempt to load the OSS channel driver (which allows the use of the `CONSOLE` channel), and Asterisk is unable to open */dev/dsp*, you will receive the following error on startup:

```
WARNING[32174]: chan_oss.c:470 soundcard_init: Unable to open /dev/dsp: No such file
or directory
  == No sound card detected -- console channel will be unavailable
  == Turn off OSS support by adding 'noload=chan_oss.so' in /etc/asterisk/modules.
conf
```

Enabling Verbosity and Debugging

Asterisk can output debugging information in the form of `WARNING`, `NOTICE`, and `ERROR` messages. These messages will give you information about your system, such as registrations, status and progression of calls, and various other useful bits of information. Note that `WARNING` and `NOTICE` messages are not errors; however, `ERROR` messages should be investigated. To enable various levels of verbosity, use `set verbose` followed by a numerical value. Useful values range from 3 to 10. For example, to set the highest level of verbosity, use:

```
# set verbose 10
```

You can also enable core debugging messages with `set debug` followed by a numerical value. To enable `DEBUG` output on the console, you may need to enable it in the *logger.conf* file by adding `debug` to the `console =>` statement, as follows:

```
console => warning,notice,error,event,debug
```

Useful values for `set debug` range from 3 to 10. For example:

```
# set debug 10
```

Conclusion

If you've worked through all of the sections in this chapter, you will have configured a pair of analog interfaces, a local SIP channel connected to a soft phone, and a connection to Free World Dialup via IAX2. These configurations are quite basic, but they give us functional channels to work with. We will make use of them in the following chapters, while we learn to build more useful dialplans.