CHAPTER 10
Asterisk for the Über-Geek

The first ninety percent of the task takes ninety percent of the time, and the last ten percent takes the other ninety percent.

—The Ninety:Ten Rule

The toughest part of writing this book was not finding things to write about, but rather deciding what we would not be able to write about. Now that we've covered the basics, you are ready to be told the truth: we have not taught you anywhere near all that there is to know about Asterisk. Well, okay, perhaps five percent, but likely less.

Now please understand, this is not because we didn't want to give you our very best; it's merely because Asterisk is, well, limitless (or so we believe).

In this chapter, we want to give you a taste of some of the wonders Asterisk holds in store for you. Pretty nearly every section in this chapter could become a book in itself (and they *will* become books, if Asterisk succeeds in the way we think it is going to).

Festival

Festival is a popular open source text-to-speech engine. The basic premise of using Festival with Asterisk is that your dialplan can pass a body of text to Festival, which will then "speak" the text to the caller. Probably the most obvious use for Festival would be to have it read your email to you when you are on the road.

Getting Festival Set Up and Ready for Asterisk

There are currently two ways to use Festival with Asterisk. The first (and easiest) method—without having to patch and recompile Festival—is to add the following text to Festival's configuration file (*festival.scm*, usually located in */etc/* or */usr/share/festival/*):

```
(define (tts_textasterisk string mode)
"(tts_textasterisk STRING MODE)
Apply tts to STRING. This function is specifically designed for use in server mode so
a single function call may synthesize the string. This function name may be added to
the server safe functions."
(let ((wholeutt (utt.synth (eval (list 'Utterance 'Text string)))))
(utt.wave.resample wholeutt 8000)
(utt.wave.rescale wholeutt 5)
(utt.send.wave.client wholeutt)))
```

You may place this text anywhere in the file, as long as it is not between any other parentheses.

The second (and more traditional) way is to compile Festival with an Asterisk-specific patch (located in the *contrib/* directory of the Asterisk source).

Information on both of these methods is contained in the *README.festival* file, located in the *contrib/* directory of the Asterisk source.

For either method, you'll need to modify the Festival access list in the *festival.scm* file. Simply search for the word "localhost," and replace it with the fully qualified domain name of your server.

Both of these methods set up Festival to be able to correctly communicate with Asterisk. After setting up Festival, you should start the Festival server. You can then call the `Festival( )` application from within your dialplan.

Configuring Asterisk for Festival

The Asterisk configuration file that deals with Festival is aptly called *festival.conf*. Inside this file, you specify the hostname and port of your Festival server, as well some settings for the caching of Festival speech. For most installations (if you're going to run Festival on your Asterisk server), the defaults will work just fine.

Starting the Festival Server

To start the Festival server for debugging purposes, simply run `festival` with the `--server` argument, like this:

```
[root@asterisk ~]# festival --server
```

Once you're sure that the Festival server is running and not rejecting your connections, you can start Festival by typing:

```
[root@asterisk ~]# festival_server 2>&1 >/dev/null &
```

Calling Festival from the Dialplan

Now that Festival is configured and the Festival server is started, let's call it from within a simple dialplan:

```
exten => 123,1,Answer( )
exten => 123,2,Festival(Asterisk and Festival are working together)
```

You should always call the `Answer()` application before calling `Festival()`, to ensure that a channel is established.

As Asterisk connects to Festival, you should see output like this in the terminal where you started the Festival server:

```
[root@asterisk ~]# festival --server
server    Sun May  1 18:38:51 2005 : Festival server started on port 1314
client(1) Sun May  1 18:39:20 2005 : accepted from asterisk.localdomain
client(1) Sun May  1 18:39:21 2005 : disconnected
```

If you see output like the following, it means you didn't add the host to the access list in *festival.scm*:

```
[root@asterisk ~]# festival --server
server    Sun May  1 18:30:52 2005 : Festival server started on port 1314
client(1) Sun May  1 18:32:32 2005 : rejected from asterisk.localdomain not in access
list
```

Yet Another Way to Use Festival with Asterisk

Some people in the Asterisk community have reported good success by passing text to Festival's *text2wave* utility and then having Asterisk play back the resulting *.wav* file. For example, you might do something like this:

```
exten => 124,1,Answer( )
exten => 124,2,System(echo "This is a test of Festival" | /usr/bin/text2wave
-scale 1.5 -F 8000 -o /tmp/festival.wav)
exten => 124,3,Playback(/tmp/festival)
exten => 124,4,System(rm /tmp/festival.wav)
exten => 124,5,Hangup( )
```

This method also allows you to call other text-to-speech engines, such as the popular speech engine from Cepstral.[a] For this example, we'll assume that Cepstral is installed in */usr/local/cepstral/*:

```
exten => 125,1,Answer( )
exten => 125,2,System(/usr/local/cepstral/bin/swift -o /tmp/swift.wav
"This is a test of Cepstral")
exten => 125,3,Playback(/tmp/swift)
exten => 125,4,System(rm /tmp/swift.wav)
exten => 125,5,Hangup( )
```

a. Cepstral can be evaluated at *http://www.cepstral.com*. Cepstral is an inexpensive commercial derivative of Festival with very good-sounding voices.

Call Detail Recording

Without even being told, Asterisk assumes that you want to store CDR information. Quite a smart machine, yes?

By default, Asterisk will create a CSV* file and place it in the folder */var/log/asterisk/cdr-csv/*. To the naked eye, this file looks like a bit of a mess. If, however, you separate each line according to the commas, you will find that each line contains information about a particular call, and that the commas separate the following values:

accountcode
: Assigned if the application SetAccount( ) is used, or if configured for the channel in the channel configuration file (i.e., *sip.conf*). The account code is assigned on a per-channel basis.

src
: Received Caller*ID (string, 80 characters).

dst
: Destination extension.

dcontext
: Destination context.

clid
: Caller*ID with text (80 characters).

channel
: Channel used (80 characters).

dstchannel
: Destination channel, if appropriate (80 characters).

lastapp
: Last application, if appropriate (80 characters).

lastdata
: Last application data (arguments, 80 characters).

start
: Start of call (date/time).

answer
: Answer of call (date/time).

end
: End of call (date/time).

* A Comma Separated Values (CSV) file is a common method of formatting database-type information in a text file. You can open CSV files with a text editor, but most spreadsheet and database programs will also read them and properly parse them into rows and columns.

`duration`
: Total time in system, in seconds (integer), from dial to hangup.

`billsec`
: Total time call is up, in seconds (integer), from answer to hangup.

`disposition`
: What happened to the call (`ANSWERED`, `NO ANSWER`, `BUSY`).

`amaflags`
: What flags to use (`DOCUMENTATION`, `BILL`, `IGNORE`, etc.), specified on a per-channel basis, like *`accountcode`*. AMA flags stand for Automated Message Accounting flags, which are somewhat standard (supposedly) in the industry.

`userfield`
: A user-defined field, maximum 255 characters.

Storing CDRs in a Database

CDRs can also be stored in a database. Asterisk currently supports SQLite, PostGreSQL, MySQL, and unixODBC. The configuration details for these databases will not be covered in this book, but they are outlined in the Asterisk source code, under the *doc/* subdirectory. (For licensing reasons, *cdr_mysql* is in *asterisk-addons*.) Many people prefer to store their CDRs in a database because this makes it easier to query them for specific information, such as billing or toll fraud. We can use the CDR applications to manipulate the current CDR from the dialplan (adding information to the custom field, for example).

CDR Challenges

While Asterisk will happily store information about any calls that pass through it, it cannot store information it is not given. For example, if you have SIP devices that are allowed to reinvite, once Asterisk has finished setting up the calls, the devices will no longer need its assistance. Whether or not those devices subsequently report call detail information back to it is something Asterisk is unable to control. If CDRs are important, make sure your IP devices are not allowed to reinvite.*

Customizing System Prompts

In keeping with the seemingly limitless flexibility of Asterisk, you can also modify the system prompts. This is very simple to explain, but generally difficult to do well.

* Reinvites can be turned off in *sip.conf* with `canreinvite=no`. Similar functionality is controlled in *iax.conf* with `notransfer=yes`.

With over three hundred system prompts in the main distribution, and over six hundred more in the *asterisk-sounds* add on, if you're contemplating customizing all of them you'd better have either a lot of money or a lot of time on your hands.

An audio engineer is also recommended, to ensure that the recordings are normalized to –3 dB and that all prompts start and end at a zero-crossing point (with just the right amount of silence prepended and appended).

The Voice

If you are interested in The Voice of Asterisk, she is Allison Smith, and she can deliver customized recordings for you to use on your own system.

This is an extremely cool concept, as very few PBXs allow you to use the same voice in your custom recordings as is used by the system prompts.

To make use of Allison's talents, sign up at *http://thevoice.digium.com*.

Once you have the recordings, the actual implementation is easy—simply replace the files in */var/lib/asterisk/sounds/* with the ones you have created.

Alternatively, you can opt to record your own prompts and place them in a folder of your choosing. When you refer to sound files with the `Playback()` or `Background()` applications, you can refer to the full pathname of the sound file, or to any subdirectory of */var/lib/asterisk/sounds/*.

A useful way to convert your WAV files to GSM format is with the use of the *sox* application. To convert your files with *sox*, use:

```
# sox foo.wav -r 8000 foo.gsm resample -ql
```

If your WAV files are recorded in stereo, be sure to add the `–c1` flag to write the files in mono. These recordings are often made through a PC, but check out the following sidebar—some people have had better luck recording from the dialplan.

Manager

Asterisk Manager provides an API that allows external programs the ability to create, monitor and manage Asterisk.* The Manager interface is a powerful mechanism for integrating external programs of all kinds into Asterisk.

To use the Manager, you must define an account in the file */etc/asterisk/manager.conf*. This file will look something like this:

```
[general]
enabled = yes
```

* An Application Program Interface (API) is a mechanism by which a program allows other programs to take control of it. Contrast this with AGI, which allows external programs to be called from the dialplan.

Sound Recording from the Dialplan

Surprisingly, one of the easiest ways to get respectable-quality recordings is not through a PC with fancy editing software, but rather through a telephone set. There are many reasons for this, but the most important is that the telephone will tend to filter out background noise (such as white noise caused by HVAC equipment) and will record at a consistent audio level.

This little addition to your dialplan will allow you to easily create recordings, which will be placed in your system's */tmp/* folder (from there, you can rename them and move them wherever you'd like):

```
exten => _66XX,1,Wait(2)
exten => _66XX,2,Record(/tmp/prompt${EXTEN:2}:wav)
exten => _66XX,3,Wait(1)
exten => _66XX,4,Playback(/tmp/prompt${EXTEN:2})
exten => _66XX,5,Wait(2)
exten => _66XX,6,Hangup( )
```

This little snippet will allow you to dial from 6600 to 6699, and it will record prompts in the */tmp/* folder using the names *prompt00.wav* to *prompt99.wav*. After you complete recording (by pressing the # key), it will play your prompt back to you and hang up.

Be sure to move your prompts out of the */tmp/* dir to the Asterisk sounds directory. To keep the dialplan readable, rename your *promptXX* files to a meaningful names—e.g., **`mv /tmp/prompt00.wav /var/lib/asterisk/sounds/custom/welcome-message.wav`**.

```
port = 5038
bindaddr = 0.0.0.0

[oreilly]
secret = notvery
;deny=0.0.0.0/0.0.0.0
;permit=209.16.236.73/255.255.255.0
read = system,call,log,verbose,command,agent,user
write = system,call,log,verbose,command,agent,user
```

In the `[general]` section, you have to enable the service by setting the parameter `enabled = yes`. The TCP port to use will default to 5038.

For each user, you will specify the username in square brackets (`[]`), followed by the password for that user (`secret`), any IP addresses you wish to `deny` access to, any IP addresses you wish to `permit` access to, and the `read` and `write` permissions for that user.

Manager Commands

It is important to keep in mind that the Manager interface is designed to be used by programs, not fingers. That's not to say that you can't issue commands to it directly—just don't expect a typical console interface, because that's not what Manager is for.

Commands to Manager are delivered in packages with the following syntax (lines are terminated with CRLF):

```
Action: <action type>
<Key 1>: <Value 1>
<Key 2>: <Value 2>
etc ...
<Variable>: <Value>
<Variable>: <Value>
etc...
```

For example, to authenticate with Manager (which is required if you expect to have any interaction whatsoever), you would send the following:

```
Action: login
Username: oreilly
Secret: notvery
<CRLF>
```

An extra CRLF on a blank line will send the entire package to Manager.

Once authenticated, you will be able to initiate actions, as well as see events generated by Asterisk. On a busy system, this can get quite complicated and become totally impossible to keep track of with the unaided eye.

The Flash Operator Panel

The Flash Operator Panel (FOP) is far and away the most popular example of the power of the Manager interface. FOP creates a web-based visual view of your system and allows you control of calls.

FOP is most commonly used to enable a live attendant to view the users in the system and connect calls between them. It can also be used in a call-center environment to provide CRM-triggered screen pops.*

The FOP management interface is shown in Figure 10-1. To grab a copy of FOP, head to *http://www.asternic.org*.

Call Files

Call files allow you to create calls through the Linux shell. These powerful events are triggered by depositing a *.call* file in the directory */var/spool/asterisk/outgoing/*. The actual name of the file does not matter, but it's good form to give the file a meaningful name and to end the filename with *.call*.

* Customer Relationship Management (CRM) is an interface companies use to help manage customer information and interaction.

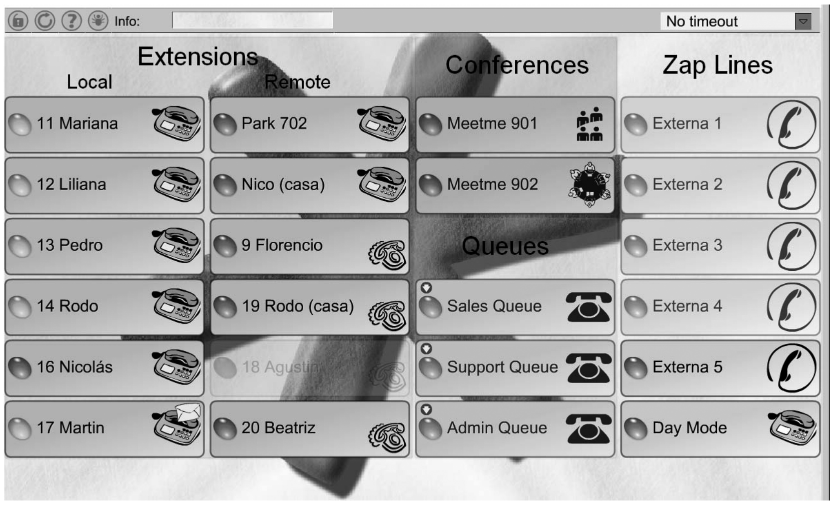

Figure 10-1. The Flash Operator Panel management interface

When a call file appears in the outgoing folder, Asterisk will almost immediately[*] act on the instructions contained therein.

Call files are formatted in the following manner. First, we define where we want to call:

```
Channel: <channel>
```

We can control how long to wait for a call to be answered (the default is 45 seconds), how long to wait between call retries, and the maximum number of retries. If `MaxRetries` is omitted, the call will be attempted only once:

```
WaitTime: <number>
RetryTime: <number>
MaxRetries: <number>
```

If the call is answered, we specify where to connect it here:

```
Context: <context-name>
Extension: <ext>
Priority: <priority>
```

Alternatively, we can specify a single application and pass arguments to it:

```
Application: Playback( )
Data: hello-world
```

Next, we set the Caller*ID of the outgoing call:

```
CallerID: Asterisk <800-555-1212>
```

Then we set channel variables, as follows:

```
SetVar: john=Zap/1/5551212
SetVar: sally=SIP/1000
```

[*] We're talking seconds or less.

and add a CDR account code:

```
Account: documentation
```

When you create a call file, do *not* do so from the spool directory. Asterisk monitors the spool aggressively and will try to grab your file before you've even finished writing it. Create call files in some other folder, and then `mv` them into the spool directory.

DUNDi

If there were any concerns that Mark Spencer was in danger of running out of good ideas, Distributed Universal Number Discovery (DUNDi) ought to lay such thoughts to rest. DUNDi is poised to be as revolutionary as Asterisk. The DUNDi web site (*http://www.dundi.com*) says it best: "DUNDi™ is a peer to peer system for locating Internet gateways to telephony services. Unlike traditional centralized services (such as the remarkably simple and concise *ENUM*[*] standard), DUNDi is fully distributed with no centralized authority whatsoever."

How Does DUNDi Work?

Think of DUNDi as a large phone book that allows you to ask peers if they know of an alternative VoIP route to an extension number or PSTN telephone number.

For example, assume you are connected to the *DUNDi-test* network (a free and open network that terminates calls to traditional PSTN numbers). You ask your friend Bob if he knows how to reach 1-800-555-1212, a number for which you have no direct access. Bob replies, "I don't know how to reach that number, but let me ask my peer Sally."

Bob asks Sally if she knows how to reach the requested number, and she responds with, "You can reach that number at *IAX2/dundi:very_long_password@hostname/extension*." Bob then stores the address in his database and passes on to you the information about how to reach 1-800-555-1212 via VoIP, allowing you an alternative method of reaching the same destination through a different network.

Because Bob has stored the information he found, he'll be able to provide it to any peers who later request the same number from him, so the lookup won't have to go any further. This helps reduce the load on the network and increases response times for numbers that are looked up often. (However, it should be noted that DUNDi creates a rotating key, and thus stored information is valid for a limited period of time.)

DUNDi performs lookups dynamically, either with a `switch =>` statement in your *extensions.conf* file or with the use of the `DUNDiLookup()` application. DUNDi is available only in Asterisk Version 1.2 or higher.

[*] *http://www.faqs.org/rfc/rfc2916.txt.*

You can use the DUNDi protocol in a private network as well. Say you're the Asterisk administrator of a very large enterprise installation and you wish to simplify the administration of extension numbers. You could use DUNDi in this situation, allowing multiple Asterisk boxes (presumably located at each of the company's locations and peered with one another) to perform dynamic lookups for the VoIP addresses of extensions on the network.

Configuring Asterisk for Use with DUNDi

There are three files that need to be configured for DUNDi: *dundi.conf*, *extensions.conf*, and *iax.conf*.* The *dundi.conf* file controls the authentication of peers who we allow to perform lookups through our system. This file also manages the list of peers to whom we might submit our own lookup requests. Since it is possible to run several different networks on the same box, it is necessary to define a different section for each peer, and then configure the networks in which that peer is allowed to perform lookups. Additionally, we need to define which peers we wish to use to perform lookups.

The General Peering Agreement

The General Peering Agreement (GPA) is a legally binding license agreement that is designed to prevent abuse of the DUNDi protocol. Before connecting to the *DUNDi-test* group, you are required to sign a GPA. The GPA is used to protect the members of the group and to create a "trust" between the members. It is a requirement of the *DUNDi-test* group that your complete and accurate contact information be configured in *dundi.conf*, so that members of your peer group can contact you. The GPA can be found in the *doc/* subdirectory of the Asterisk source.

General configuration

The `[general]` section of *dundi.conf* contains parameters relating to the overall operation of the DUNDi client and server:

```
; DUNDi configuration file
;
[general]
;
department=IT
organization= toronto.example.com
locality=Toronto
stateprov=ON
country=CA
email=support@toronto.example.com
phone=+19055551212
;
```

* The *dundi.conf* and *extensions.conf* files must be configured. We have chosen to configure *iax.conf* for our address advertisement on the network, but DUNDi is protocol-agnostic—thus *sip.conf*, *h323.conf*, or *mgcp.conf* could be used instead.

```
; Specify bind address and port number.  Default is 4520
;bindaddr=0.0.0.0
port=4520
entityid=FF:FF:FF:FF:FF:FF
ttl=32
autokill=yes
;secretpath=dundi
```

The entity identifier defined by `entityid` should generally be the Media Access Control (MAC) address of an interface in the machine. The entity ID defaults to the first Ethernet address of the server, but you can override this with `entityid`, as long as it is set to the MAC address of *something* you own. The MAC address of the primary external interface is recommended. This is the address that other peers will use to identify you.

The Time To Live (`ttl`) field defines how many peers away we wish to receive replies from and is used to break loops. Each time a request is passed on down the line because the requested number is not known, the value in the TTL field is decreased by one, much like the TTL field of an ICMP packet. The TTL field also defines the maximum number of seconds we are willing to wait for a reply.

When you request a number lookup, an initial query (called a `DPDISCOVER`) is sent to your peers requesting that number. If you do not receive an acknowledgment (`ACK`) of your query (`DPDISCOVER`) within 2,000 ms (enough time for a single transmission only), and `autokill` is equal to `yes`, Asterisk will send a `CANCEL` to the peers. (Note that an acknowledgment is not necessarily a reply to the query; it is just an acknowledgment that the peer has received the request.) The purpose of `autokill` is to keep the lookup from stalling due to hosts with high latency. In addition to the `yes` and `no` options, you may also specify the number of milliseconds to wait.

The *pbx_dundi* module creates a rotating key and stores it in the local Asterisk database (AstDB). The key name `secret` is stored in the `dundi` family. The value of the key can be viewed with the `database show` command at the Asterisk console. The database family can be overridden with the `secretpath` option.

Creating mapping contexts

The *dundi.conf* file defines DUNDi contexts that are mapped to dialplan contexts in your *extensions.conf* file. DUNDi contexts are a way of defining distinct and separate directory service groups. The contexts in the mapping section point to contexts in the *extensions.conf* file, which control the numbers that you advertise. When you create a peer, you need to define which mapping contexts you will allow this peer to search. You do this with the `permit` statement (each peer may contain multiple `permit` statements). Mapping contexts are related to dialplan contexts in the sense that they are a security boundary for your peers.

Phone numbers must be advertised in the following format:

```
<country_code><area_code><prefix><number>
```

For example, a complete North American number could be advertised as 14165551212.

All DUNDi mapping contexts take the form of:

```
dundi_context => local_context,weight,technology,destination[,options]]
```

The following configuration creates a DUNDi mapping context that we will use to advertise our local phone numbers to the *DUNDi-test* group. Note that this should all appear on one line:

```
dundi-test => dundi-local,0,IAX2,dundi:${SECRET}@toronto.example.com/${NUMBER},
nounsolicited,nocomunsolicit,nopartial
```

In this example, the mapping context is `dundi-test`, which points to the `dundi-local` context within *extensions.conf* (providing a listing of phone numbers to reply to). Numbers that resolve to the PBX should be advertised with a ***`weight`*** of zero (directly connected). Numbers higher than 0 indicate an increased number of hops or paths to reach the final destination. This is useful when multiple replies for the same lookup are received at the end that initially requested the number—a ***`weight`*** with a lower number will be the preferred path.

If we can reply to a lookup, our response will contain the method by which the other end can connect to the system. This includes the technology to use (such as IAX2, SIP, H323, and so on), the username and password with which to authenticate, which host to send the authentication to, and finally the extension number.

Asterisk provides some shortcuts to allow us to create a "template" with which we can build our responses. The following channel variables can be used to construct the template:

`${SECRET}`
: Replaced with the password stored in the local AstDB

`${NUMBER}`
: The number being requested

`${IPADDR}`
: The IP address to connect to

It is generally safest to statically configure the hostname, rather than making use of the `${IPADDR}` variable. The `${IPADDR}` variable will sometimes reply with an address in the private IP space, which is unreachable from the Internet.

Defining DUNDi peers

DUNDi peers are defined in the *dundi.conf* file. Peers are identified by the unique layer-two MAC address of an interface on the remote system. The *dundi.conf* file is where we define what context to search for peers requesting a lookup and which peers we want to use when doing a lookup for a particular network.

```
[00:00:00:00:00:00] ; Remote Office
model = symmetric
host = montreal.example.com
inkey = montreal
outkey = toronto
include = dundi-test
permit = dundi-test
qualify = yes
dynamic=yes
```

The remote peer's identifier (MAC address) is enclosed in square brackets ([]). The inkey and outkey are the public/private key pairs that we use for authentication. Key pairs are generated with the *astgenkey* script, located in the *./asterisk/contrib/scripts/* source directory. Be sure to use the –n flag so that you don't have to initialize passwords every time you start Asterisk:

```
# cd /var/lib/asterisk/keys
# /usr/src/asterisk/contrib/scripts/astgenkey -n toronto
```

The resulting keys, *toronto.pub* and *toronto.key*, will be placed in your */var/lib/asterisk/keys/* directory. The *toronto.pub* file is the public key, which you should post to a web server so that it is easily accessible for anyone with whom you wish to peer. When you peer, you can give your peers the HTTP-accessible public key, which they can then place in their */var/lib/asterisk/keys/* directories.

After you have downloaded the keys, you must reload the *res_crypto.so* and *pbx_dundi.so* modules in Asterisk:

```
*CLI> reload res_crypto.so
    -- Reloading module 'res_crypto.so' (Cryptographic Digital Signatures)
    -- Loaded PRIVATE key 'toronto'
    -- Loaded PUBLIC key 'toronto'

*CLI> reload pbx_dundi.so
    -- Reloading module 'pbx_dundi.so' (Distributed Universal Number Discovery
(DUNDi))
  == Parsing '/etc/asterisk/dundi.conf': Found
```

Then, create the dundi user in the *iax.conf* file to allow connections into your Asterisk system. When a call is authenticated, the extension number being requested is passed to the dundi-local context in the *extensions.conf* file, where the call is then handled by Asterisk.

Allowing remote connections

Here is the user definition for the dundi user:

```
[dundi]
type=user
dbsecret=dundi/secret
context=dundi-local
disallow=all
allow=ulaw
allow=g726
```

Instead of using a static password, Asterisk regenerates passwords every 3,600 seconds (1 hour). The value is stored in */dundi/secret* of the Asterisk database and advertised using the `${SECRET}` variable defined within the mapping context lines in *dundi.conf*. You can see the current keys for all peers, including your local public and private keys, by performing a **show keys** at the Asterisk CLI.

The `context` entry `dundi-local` is where authorized callers are sent in *extensions.conf*. From there, we can manipulate the call just as we would in the dialplan of any other incoming connection.

Configuring the dialplan

The *extensions.conf* file handles what numbers you advertise and what you do with the calls that connect to them. The `dundi-local` context performs double duty:

- It controls the numbers we advertise, referenced by the `dundi` mapping context in *dundi.conf*.
- It controls what to do with the call, referenced by the `dundi` user in *iax.conf*.

You have the power of dialplan pattern matching to advertise ranges of numbers and to control the incoming calls. In the following dialplan, we are only advertising the number +1-416-555-1212, but pattern matching could just as easily have been employed to advertise a range of numbers or extensions:

```
[dundi-local]
exten => 14165551212,1,NoOp(dundi-local: Number advertisement and incoming)
exten => 14165551212,n,Answer( )
exten => 14165551212,n(call),Dial(SIP/1000)
exten => 14165551212,n,Voicemail(u1000)
exten => 14165551212,n,Hangup( )
exten => 14165551212,n(call)+101,Voicemail(b1000)
exten => 14165551212,n,Hangup( )
```

Conclusion

That's pretty much all this chapter is going to teach you, but it's nowhere near all there is to learn. Hopefully, you are starting to get an idea of how big this Asterisk thing really is.

In the next chapter, we're going to try and predict the future of telecom, and we'll discuss how (and why) we believe that Asterisk is well positioned to play a starring role.