APPENDIX D

Configuration Files

This appendix contains a reference to the configuration files not covered in the previous appendixes. If you are looking for VoIP channel configurations, refer to Appendix A. For a dialplan reference, you'll want to use Appendix B.

A configuration file is required for each Asterisk module you wish to use. These *.conf* files contain channel definitions, describe internal services, define the locations of other modules, or relate to the dialplan. You do not need to configure all of them to have a functioning system, only the ones required for your configuration. Although Asterisk ships with samples of all of the configuration files, it is possible to start Asterisk without any of them. This will not provide you with a working system, but it clearly demonstrates the modularity of the platform.

If no *.conf* files are found, Asterisk will make some decisions with respect to modules. For example, the following steps are always taken:

- The Asterisk Event Logger is loaded, and events are logged to */var/log/asterisk/event_log*.
- Manager actions are registered.
- The PBX core is initialized.
- The RTP port range is allocated from 5,000 through 31,000.
- Several built-in applications are loaded, such as `Answer( )`, `Background( )`, `GotoIf( )`, `NoOp( )`, and `Set( )`.
- The dynamic loader is started—this is the engine responsible for loading modules defined in *modules.conf*.

This appendix starts with an in-depth look at the *modules.conf* configuration file. We'll then briefly examine all the other files that you may need to configure for your Asterisk system.

modules.conf

The *modules.conf* file controls which modules are loaded or not loaded at Asterisk startup. This is done through the use of the `load =>` or `noload =>` constructs.

This file is a key component to building a secure Asterisk installation: best practice suggests that only required modules be loaded.

The *modules.conf* file always starts with the `[modules]` header. The `autoload` statement tells Asterisk whether to automatically load all modules contained within the modules directory or to load only those modules specifically defined by `load =>` statements. We recommend you manually load only those modules you need, but many people find it easier to let Asterisk attempt to `autoload` whatever it finds in */usr/lib/asterisk/modules*. You can then exclude certain modules with `noload =>` statements.

Here's a sample *modules.conf* file:

```
[modules]
autoload=no                ; set this to yes and Asterisk will load any
                           ; modules it finds in /usr/lib/asterisk/modules

load => res_adsi.so
load => pbx_config.so      ; Requires: N/A
load => chan_iax2.so       ; Requires: res_crypto.so, res_features.so
load => chan_sip.so        ; Requires: res_features.so
load => codec_alaw.so      ; Requires: N/A
load => codec_gsm.so       ; Requires: N/A
load => codec_ulaw.so      ; Requires: N/A
load => format_gsm.so      ; Requires: N/A
load => app_dial.so        ; Requires: res_features.so, res_musiconhold.so
```

Since we assume Asterisk is built on Linux, all the module names we use end in a *.so* extension. However, this may not be the case if you have built Asterisk on a different operating system.

As of this writing, there are eight module types: *resources*, *applications*, *Call Detail Record database connectors*, *channels*, *codecs*, *formats*, *pbx modules*, and *standalone functions*. Let's take a look at each of them.

Resources

A *resource* provides a connection to a static repository of a particular type of information, such as a unique regional requirement or a library of constant elements. This information must be configurable for each system, but once loaded it doesn't need to change in the course of normal operations.

For each resource below, we have outlined the applications and features it provides to other Asterisk modules We've indicated the *.conf* file used to define the resource, where needed; if no file is listed, then a configuration file isn't required. The resource modules are:

res_adsi.so
: Configuration file: *adsi.conf*
: Provides: ADSI functions to `ADSIProg( )` and `Voicemail( )`

res_agi.so
: Provides: `DeadAGI( )`, `EAGI( )`, `AGI( )`

res_crypto.so
: Provides: Loads public and private keys located in */var/lib/asterisk/keys/*

res_features.so
: Configuration file: *features.conf*
: Provides: `ParkedCall( )`, `Park( )`

res_indications.so
: Configuration file: *indications.conf*
: Provides: `Playtones( )`, `StopPlaytones( )`

res_monitor.so
: Provides: `Monitor( )`, `StopMonitor( )`, `ChangeMonitor( )`, action `Monitor`, action `StopMonitor`, action `ChangeMonitor`

res_musiconhold.so
: Configuration file: *musiconhold.conf*
: Provides: `MusicOnHold( )`, `WaitMusicOnHold( )`, `SetMusicOnHold( )`, `StartMusicOnHold( )`, `StopMusicOnHold( )`

res_odbc.so
: Configuration file: *res_odbc.conf*
: Provides: Connectivity information to the ODBC[*] driver—the purpose is to store configuration file information in a database and retrieve that information from the database; however, a reload is required to make changes take effect

Applications

If you build an Asterisk dialplan of any size, you are going to use at least one—and more likely dozens—of applications.[†] If an application is never going to be used, it is not strictly required that it be loaded. For performance-challenged systems (or if you

[*] Open DataBase Connectivity (ODBC) is a standard by which access to a database can be provided.

[†] To be of any use, a self-contained dialplan will always require several applications. Some folks, however, use the dialplan for no other purpose than to pass control to an external application. In this case, it would be possible to have the dialplan use no application other than `AGI( )`. We're not recommending that you do this, but again, it demonstrates Asterisk's enormous flexibility.

just like to keep it lean), you may elect to load only those applications that are referenced in your dialplan.

For each application module, we will define any resource requirements and name the applications that the module provides. Unless we have stated otherwise, the application does not require a configuration file or any other modules. The available application modules are:

app_adsiprog.so
: Requires: *res_adsi.so*
: Provides: `ADSIProg( )`

app_alarmreceiver.so
: Provides: `AlarmReceiver( )`

app_authenticate.so
: Provides: `Authenticate( )`

app_cdr.so
: Provides: `NoCDR( )`

app_chanisavail.so
: Provides: `ChanIsAvail( )`

app_chanspy.so
: Provides: `ChanSpy( )`

app_controlplayback.so
: Provides: `ControlPlayback( )`

app_curl.so
: Provides: `Curl( )`

app_cut.so
: Provides: `Cut( )`

app_db.so
: Provides: `DBget( )`, `DBput( )`, `DBdel( )`, `DBdeltree( )`

app_dial.so
: Requires: *res_features.so*, *res_musiconhold.so*
: Provides: `Dial( )`, `RetryDial( )`

app_dictate.so
: Provides: `Dictate( )`

app_directory.so
: Provides: `Directory( )`

app_disa.so
: Provides: `DISA( )`

app_dumpchan.so
: Provides: `DumpChan( )`

app_echo.so
: Provides: `Echo( )`

app_enumlookup.so
: Configuration file: *enum.conf*
: Provides: `EnumLookup( )`

app_eval.so
: Provides: `Eval( )`

app_exec.so
: Provides: `Exec( )`

app_festival.so
: Provides: `Festival( )`

app_forkcdr.so
: Provides: `ForkCDR( )`

app_getcpeid.so
: Requires: *res_adsi.so*
: Provides: `GetCPEID( )`

app_groupcount.so
: Provides: `GetGroupCount( )`, `SetGroup( )`, `CheckGroup( )`, `GetGroupMatchCount( )`

app_hasnewvoicemail.so
: Provides: `HasVoicemail( )`, `HasNewVoicemail( )`

app_ices.so
: Provides: `ICES( )`

app_image.so
: Provides: `SendImage( )`

app_lookupblacklist.so
: Provides: `LookupBlacklist( )`

app_lookupcidname.so
: Provides: `LookupCIDName( )`

app_macro.so
: Provides: `Macro( )`, `MacroExit( )`, `MacroIf( )`

app_math.so
: Provides: `Math( )`

app_md5.so
: Provides: `MD5( )`, `MD5Check( )`

app_milliwatt.so
: Provides: `Milliwatt( )`

app_mp3.so
: Provides: `MP3Player( )`

app_nbscat.so
: Provides: `NBScat( )`

app_parkandannounce.so
: Requires: *res_features.so*
: Provides: `ParkAndAnnounce( )`

app_playback.so
: Provides: `Playback( )`

app_privacy.so
: Provides: `PrivacyManager( )`

app_queue.so
: Requires: *res_features.so*, *res_monitor.so*, *res_musiconhold.so*
: Provides: `Queue( )`, `AddQueueMember( )`, `RemoveQueueMember( )`, `PauseQueueMember( )`, `UnpauseQueueMember( )`, action `Queues`, action `QueueStatus`, action `QueueAdd`, action `QueueRemove`, action `QueuePause`

app_random.so
: Provides: `Random( )`

app_read.so
: Provides: `Read( )`

app_readfile.so
: Provides: `ReadFile( )`

app_realtime.so
: Provides: `RealTime( )`, `RealTimeUpdate( )`

app_record.so
: Provides: `Record( )`

app_sayunixtime.so
: Provides: `SayUnixTime( )`, `DateTime( )`

app_senddtmf.so
: Provides: `SendDTMF( )`

app_sendtext.so
: Provides: `SendText( )`

app_setcallerid.so
: Provides: `SetCallerPres( )`, `SetCallerID( )`

app_setcdruserfield.so
: Provides: `SetCDRUserField( )`, `AppendCDRUserField( )`, action `SetCDRUserField`

app_setcidname.so
: Provides: `SetCIDName( )`

app_setcidnum.so
: Provides: `SetCIDNum( )`

app_setrdnis.so
: Provides: `SetRDNIS( )`

app_settransfercapability.so
: Provides: `SetTransferCapability( )`

app_sms.so
: Provides: `SMS( )`

app_softhangup.so
: Provides: `SoftHangup( )`

app_striplsd.so
: Provides: `StripLSD( )`

app_substring.so (deprecated)
: Provides: `SubString( )`

app_system.so
: Provides: `System( )`, `TrySystem( )`

app_talkdetect.so
: Provides: `BackgroundDetect( )`

app_test.so
: Provides: `TestClient( )`, `TestServer( )`

app_transfer.so
: Provides: `Transfer( )`

app_txtcidname.so
: Configuration file: *enum.conf*
: Provides: `TXTCIDName( )`

app_url.so
: Provides: `SendURL( )`

app_userevent.so
: Provides: `UserEvent( )`

app_verbose.so
: Provides: `Verbose( )`

app_voicemail.so
: Configuration file: *voicemail.conf*
: Requires: *res_adsi.so*
: Provides: `VoiceMail( )`, `VoiceMailMain( )`, `MailboxExists( )`, `VMAuthenticate( )`

app_waitforring.so
: Provides: `WaitForRing( )`

app_waitforsilence.so
: Provides: `WaitForSilence( )`

app_while.so
    Provides: `While()`, `ExecIf()`, `EndWhile()`

app_zapateller.so
    Provides: `Zapateller()`

Database-Stored Call Detail Records

Asterisk normally stores Call Detail Records (CDRs) in a Comma-Separated Values (CSV) file.[*] If you want CDRs to be stored in a database, you'll need to load the appropriate module and define the relevant *.conf* file.

For each module below, we state the database type it supports, and specify the configuration file, if required. The CDR database connector modules are:

cdr_csv.so
    Provides: CSV CDR backend

cdr_custom.so
    Configuration file: *cdr_custom.conf*

    Provides: Customizable CSV CDR backend

cdr_manager.so
    Configuration file: *cdr_manager.conf*

    Provides: Asterisk Call Manager CDR backend

cdr_odbc.so[†]
    Configuration file: *cdr_odbc.conf*

    Provides: ODBC CDR backend

cdr_pgsql.so
    Configuration file: *cdr_pgsql.conf*

    Provides: PostgreSQL CDR backend

Channels

Next, let's take a look at the channel modules. For each channel module, we identify dependencies and list the capabilities the module provides. We show the configuratin file, if one is required. The available modules are:

[*] Information stored in a text file as Comma-Separated Values can be imported into pretty much any spreadsheet or database (yes, even stuff from Microsoft). This makes the CSV format extremely portable.

[†] The *cdr_odbc* connector could theoretically replace all of the other database-specific connectors—however, people may prefer to use specific connectors due to performance differences, stability issues, personal preference, backward-compatibility, and so forth. Many options are available. If you are familiar with databases, Asterisk gives you lots of choices.

chan_agent.so
: Configuration file: *agents.conf*

  Requires: *res_features.so*, *res_monitor.so*, *res_musiconhold.so*

  Provides: channel Agent, `AgentLogin( )`, `AgentCallbackLogin( )`, `AgentMonitorOutgoing( )`, action `Agents`

chan_features.so
: Provides: channel Feature

chan_iax2.so
: Configuration file: *iax.conf*, *iaxprov.conf*

  Requires: *res_crypto.so*, *res_features.so*, *res_musiconhold.so*

  Provides: channel IAX2, `IAX2Provision( )`, function `IAXPEER`, action `IAXPEERS`, action `IAXnetstats`

chan_local.so
: Provides: channel Local

chan_mgcp.so
: Configuration file: *mgcp.conf*

  Requires: *res_features.so*

  Provides: channel MGCP

chan_modem.so
: Configuration file: *modem.conf*

  Provides: channel Modem

chan_modem_aopen.so
: Requires: *chan_modem.so*

  Provides: A/Open (Rockwell Chipset) ITU-2 VoiceModem Driver

chan_modem_bestdata.so
: Requires: *chan_modem.so*

  Provides: BestData (Conexant V.90 Chipset) VoiceModem Driver

chan_modem_i4l.so
: Requires: *chan_modem.so*

  Provides: ISDN4Linux Emulated Modem Driver

chan_oss.so
: Provides: channel Console (soundcard required)

chan_phone.so
: Configuration file: *phone.conf*

  Provides: channel Phone

chan_sip.so
: Configuration file: *sip.conf*, *sip_notify.conf*

  Requires: *res_features.so*

Provides: channel SIP, `SIPDtmfMode()`, `SIPAddHeader()`, `SIPGetHeader()`, action `SIPpeers`, action `SIPshowpeer`, function `SIP_HEADER`

chan_skinny.so

Configuration file: *skinny.conf*

Requires: *res_features.so*

Provides: channel Skinny

Codecs

There are several acceptable ways to pass audio information in digital form. The formulas used to encode and decode (or compress and decompress) this information are collectively referred to as *codecs*. Most of Asterisk's codecs are provided free of license requirements; however, some (such as G.729) are encumbered by patents and thus must be licensed before they can be used.

Asterisk will load these codecs without complaint, but if you attempt to transcode a channel using an unlicensed codec, your calls will be dropped as soon as they connect.

Here, then, are the codec modules—if there are parameters that can be defined, they will be configurable in the *codecs.conf* file:

codec_a_mu.so

Provides: translator `alawtoulaw`, translator `ulawtoalaw`

codec_adpcm.so

Configuration file: *codecs.conf*

Provides: translator `adpcmtolin`, translator `lintoadpcm`

codec_alaw.so

Configuration file: *codecs.conf*

Provides: translator `alawtolin`, translator `lintoalaw`

codec_g726.so

Configuration file: *codecs.conf*

Provides: translator `g726tolin`, translator `lintog726`

codec_gsm.so

Configuration file: *codecs.conf*

Provides: translator `gsmtolin`, translator `lintogsm`

codec_ilbc.so

Configuration file: not required

Provides: translator `ilbctolin`, translator `lintoilbc`

codec_lpc10.so

Configuration file: *codecs.conf*

Provides: translator `lpc10tolin`, translator `lintolpc10`

codec_ulaw.so
  Configuration file: *codecs.conf*
  Provides: translator `ulawtolin`, translator `lintoulaw`

Formats

Formats are essentially the same as codecs, except that they relate to handling files instead of live media streams. If you are talking to someone, a codec (or two) will be employed. If you are leaving a voicemail or listening to Music on Hold, a format will be involved.

Here are the current Asterisk formats. Formats do not have associated configuration files:

format_g723.so
  Provides: format `g723sf`

format_g726.so
  Provides: format `g726-40`, format `g726-32`, format `g726-24`, format `g726-16`

format_g729.so
  Provides: format `g729`

format_gsm.so
  Provides: format `gsm`

format_h263.so
  Provides: format `h263`

format_ilbc.so
  Provides: format `ilbc`

format_jpeg.so
  Provides: format `jpg`

format_pcm.so
  Provides: format `pcm`

format_pcm_alaw.so
  Provides: format `alaw`

format_sln.so
  Provides: format `sln`

format_vox.so
  Provides: format `vox`

format_wav.so
  Provides: format `wav`

format_wav_gsm.so
  Provides: format `wav49`

PBX Core Modules

The PBX modules deliver the core functionality of the system. For each module, we show the services it provides, and list the configuration file, if one is required. At minimum, *config*, *functions*, and *spool* are required. *dundi*, *loopback*, and *realtime* are needed only if you are going to make use of their capabilities. The PBX core modules are:

pbx_config.so
: Configuration file: *extensions.conf*

  Provides: Loads dialplan into memory

pbx_dundi.so
: Configuration file: *dundi.conf*

  Requires: *res_crypto.so*

  Provides: `DUNDiLookup( )`

pbx_functions.so
: Configuration file: not required

  Provides: function `CDR`, function `CHECK_MD5`, function `DB`, function `DB_EXISTS`, function `ENV`, function `EVAL`, function `EXISTS`, function `FIELDQTY`, function `GROUP_COUNT`, function `GROUP_MATCH_COUNT`, function `GROUP`, function `GROUP_LIST`, function `IF`, function `ISNULL`, function `LANGUAGE`, function `LEN`, function `MD5`, function `REGEX`, function `STRFTIME`, function `SET`, function `TIMEOUT`

pbx_loopback.so
: Provides: Loopback switch

pbx_realtime.so
: Provides: Realtime switch

pbx_spool.so
: Provides: Outgoing spool support

Standalone Functions

There is currently only one standalone function available. This function operates identically to those in *pbx_functions.so*, but because it is standalone, it can be loaded (or not) completely independently of the *pbx* functions. The function is:

func_callerid.so
: Configuration file: not required

  Provides: function `CALLERID`

adsi.conf

The Analog Display Services Interface (ADSI) was designed to allow telephone companies to deliver enhanced services across analog telephone circuits. In Asterisk, you can use this file to send ADSI commands to compatible telephones. Please note that the phone must be directly connected to a Zapata channel. ADSI messages cannot be sent across a VoIP connection to a remote analog phone.

The *res_adsi.so* module is required for the `Voicemail()` application; however, the *adsi.conf* file is not necessarily used. Detailed information about ADSI is not publicly available, and documentation needs to be purchased from Telcordia.

adtranvofr.conf

Prior to Voice over IP, Voice over Frame Relay (VoFR) enjoyed brief fame as a means of carrying packetized voice. Supporting VoFR through Adtran equipment is part of the history of Asterisk.

This feature is no longer popular in the community, though, so it may be difficult to find support for it.

agents.conf

This file allows you to create and manage agents for your call center. If you are using the `Queue()` application, you may want to configure agents for the queue. The *agents.conf* file is used to configure the AGENT channel driver.

The `[general]` section in *agents.conf* currently contains only one parameter. The `persistentagents=yes` parameter tells Asterisk to save the status of agents who use the callback feature of queues in the local Asterisk database. A logged-in remote agent will then remain logged in across a reboot (unless removed from the database through some other means).

The following parameters, which are specified in the `[agents]` section, are used to define agents and the way the system interacts with them. The settings apply to all agents, unless otherwise specified in the individual agent definitions:

`ackcall`
: Accepts the arguments `yes` and `no`. If set to `yes`, requires a callback agent to acknowledge login by pressing the `#` key after logging in. This works in conjunction with the `AgentCallbackLogin()` application.

`autologoff`
: Accepts an argument (in seconds) defining how long an agent channel should ring for before the agent is deemed unavailable and logged off.

`group`
: Defines the groups to which an agent belongs, specified with integers. Specify that an agent belongs to multiple groups by separating the integers with commas.

`musiconhold => *class*`
: Accepts a Music on Hold class as its argument. This setting applies to all agents.

`updatecdr`
: Accepts the arguments `yes` and `no`. Used to define whether the source channel in the CDRs should be set to agent/agent_id to determine which agent generated the calls.

`wrapuptime`
: Accepts an argument (in milliseconds) specifying the amount of time to wait after an agent has finished a call before that agent can be considered available to answer another call.

The remaining parameters are also specified in the `[agents]` section, but they are global to the *chan_agent* channel driver and thus cannot be defined on a per-agent basis:

`createlink`
: Accepts the arguments `yes` and `no`. Inserts the name of the created recording in the CDR user field.

`custom_beep`
: Accepts a filename as its argument. Can be used to define a custom notification tone to signal to an always-connected agent that there is an incoming call.

`recordagentcalls`
: Accepts the arguments `yes` and `no`. Defines whether or not agent calls should be recorded.

`recordformat`
: Defines the format to record files in. The argument specified should be `wav`, `gsm`, or `wav49`. The default recording format is `wav`.

`savecallsin`
: Accepts a filesystem path as its argument. Allows you to override the default path of */var/spool/asterisk/monitor/* with one of your choosing.

> Since the storage of calls will require a large amount of hard drive space, you will want to define a strategy to handle storing and managing these recordings.
>
> This location should probably reside on a separate volume; one with very high performance characteristics.

`urlprefix`
: Accepts a string as its argument. The string can be formed as a URL and is appended to the start of the text to be added to the name of the recording.

The final parameter is used to define agents. As in the *zapata.conf* file, configuration parameters are inherited from above the `agent =>` definition. Agents are defined with the following format:

```
agent => agent_id,agent_password,name
```

For example, we can define agent Happy Tempura with the agent ID 1000 and password 1234, as follows.

```
agent => 1000,1234,Happy Tempura
```

Be aware that an *agents.conf* file is a complement to the queue configuration process. The most critical configuration file for your queues is *queues.conf*. You can configure a very basic queue without *agents.conf*.

alarmreceiver.conf

The `AlarmReceiver()` application is not approved by Underwriter's Laboratory (UL) and should not be used as the primary or sole means of receiving alarm messages or events. This application is not guaranteed to be reliable, so don't depend on it unless you have extensively tested it. Use of this application without extensive testing may place your life and/or property at risk.

The *alarmreceiver.conf* file is used by the `AlarmReceiver()` application, which allows Asterisk to accept alarms using the SIA (Ademco) Contact ID protocol. When a call is received from an alarm panel, it should be directed to a context that calls the `AlarmReceiver()` application. In turn, `AlarmReceiver()` will read the *alarmreceiver.conf* configuration file and perform the configured actions as required. All parameters are specified under the `[general]` heading.

The sample configuration file will contain the current settings for this application and is very well documented.

alsa.conf

The *alsa.conf* file is used to configure Asterisk to use the Advanced Linux Sound Architecture (ALSA) to provide access to a sound card, if desired. You can use this file to configure the CONSOLE channel, which is most commonly used to create an overhead paging system (although, as with any other channel, there are all kinds of creative ways this can be used). Keep in mind that the usefulness of the ALSA channel by itself is limited due to its lack of a user interface.*

* Yes, we are aware that the user interface to the channel interface is the Asterisk CLI; however, this is not usable as a telephone and therefore does not meet the criteria of an interface from the perspective of a telephone user.

asterisk.conf

The *asterisk.conf* file defines the locations for the configuration files, the spool directory, and the modules, as well as a location to write log files to. The default settings are recommended unless you understand the implications of changing them. The *asterisk.conf* file is generated automatically when you run the `make samples` command, based on information it collects about your system. It will contain a `[directories]` section such as the following:

```
[directories]
astetcdir => /etc/asterisk
astmoddir => /usr/lib/asterisk/modules
astvarlibdir => /var/lib/asterisk
astagidir => /var/lib/asterisk/agi-bin
astspooldir => /var/spool/asterisk
astrundir => /var/run
astlogdir => /var/log/asterisk
```

Additionally, you can specify an `[options]` section, which will allow you to define startup options (command-line switches) in the configuration file. The following example shows the available options and the command-line switches that they effectively enforce:

```
[options]
verbose=<value>           ; starting verbosity level (-v)
debug=yes|no|<val>        ; turn debugging on or off (or value in 1.2) (-d)
nofork=yes|no             ; don't fork a background process (-f)
console=yes|no            ; load the Asterisk console (-c)
highpriority=yes|no       ; run with high priority (-p)
initcrypto=yes|no         ; initialize crypto at start (-i)
nocolor=yes|no            ; disable ANSI colors on the console (-n)
dumpcore=yes|no           ; dump a core file on failure (-g)
quiet=yes|no              ; run quietly (-q)
cache_record_files=yes|no ; cache files recorded with Record( ) in an alternative
                          ; directory in conjunction with record_cache_dir
record_cache_dir=<dir>    ; directory in which to cache files recorded with
                          ; Record ( ) until completion
execincludes=yes|no       ; enable support of #exec includes in configuration
                          ; files (off by default)
```

cdr.conf

The *cdr.conf* file is used to enable call detail record logging to a database. Storing call records is useful for all sorts of purposes, including billing, fraud prevention, QoS evaluations, and more. *cdr.conf* contains some general parameters that are not specific to any particular database, but rather indicate how Asterisk should handle the passing of information to the database. All options are under the `[general]` heading of the *cdr.conf* file:

`batch`
: Accepts the arguments `yes` and `no`. Allows Asterisk to write data to a buffer instead of writing to the database at the end of every call, to reduce load on the system.

> Note that if the system dies unexpectedly when this option is set to `yes`, data loss may occur.

`enable`
: Accepts the arguments `yes` and `no`. Specifies whether or not to use CDR logging. If set to `no`, this will override any CDR module explicitly loaded. The default is `yes`.

`safeshutdown`
: Accepts the arguments `yes` and `no`. Setting `safeshutdown` to `yes` will prevent Asterisk from shutting down completely until the buffer is flushed and all information is written to the database. If this parameter is set to `no` and you shut down Asterisk with information still residing in the buffers, that information will likely be lost.

`scheduleronly`
: Accepts the arguments `yes` and `no`. If you are generating a massive volume of CDRs on a system that is pushing them to a remote database, setting `scheduleronly` to `yes` may be of benefit. Since the scheduler cannot start a new task until the current one is finished, slow CDR writes may adversely affect other processes needing the scheduler. This setting will instruct Asterisk to handle CDR writes in a new thread, essentially assigning a dedicated scheduler to this function. In normal operation, this would yield very little benefit.

`size`
: Accepts an integer as its argument. Defines the number of CDRs to accumulate in the buffer before writing to the database. The default is 100.

`time`
: Accepts an integer (in seconds) as its argument. Sets the number of seconds before Asterisk flushes the buffer and writes the CDRs to the database, regardless of the number of records in the buffer (as defined by `size`). The default is 300 seconds (5 minutes).

cdr_manager.conf

The *cdr_manager.conf* file simply contains a `[general]` heading and a single option, `enabled`, which you can use to specify whether or not the Asterisk Manager API generates CDR events. If you want CDR events to be generated, you will need the following lines in your *cdr_manager.conf* file:

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

The Manager API will then output CDR events containing the following fields:

```
Event: Cdr
AccountCode:
Source:
Destination:
DestinationContext:
CallerID:
Channel:
DestinationChannel:
LastApplication:
LastData:
StartTime:
AnswerTime:
EndTime:
Duration:
BillableSeconds:
Disposition:
AMAFlags:
UniqueID:
UserField:
```

cdr_odbc.conf

Asterisk can store CDR data in a local or remote database via the ODBC interface. The *cdr_odbc.conf* file contains the information Asterisk needs to connect to the database. The *cdr_odbc.so* module will attempt to load the *cdr_odbc.conf* file, and if information is found for connecting to a database, the CDR data will be recorded there.

If you are going to use a database for storing CDR data, you will have to select *one* of the many that are available. Asterisk does not like having multiple CDR databases to connect to, so do not have extra *cdr_.conf* files hanging about your Asterisk configuration directory.

cdr_pgsql.conf

Asterisk can store CDR data in a PostgreSQL database via the *cdr_pgsql.so* module. When the module is loaded the necessary information will be read from the *cdr_pgsql.conf* file, and Asterisk will connect to the PostgreSQL database to write and store CDR data.

cdr_tds.conf

Asterisk can also store CDR data to a FreeTDS database (including MS SQL) with the use of the *cdr_tds.so* module. The configuration file *cdr_tds.conf* is read once the module is loaded. Upon a successful connection, CDR data will be written to the database.

codecs.conf

Most codecs do not have any configurable parameters—they are what they are, and that's all they are.

Some codecs, however, are capable of behaving in different ways. This primarily means that they can be optimized for a particular goal, such as cutting down on latency, making best use of a network, or perhaps delivering high quality.

The *codecs.conf* file is fairly new in Asterisk, and as of this writing it allows configuration of Speex parameters only. The settings are self-explanatory, as long as you are familiar with the Speex protocol (see *http://www.speex.org*).

codecs.conf also allows you to configure Packet Loss Concealment (PLC). You need to define a `[plc]` section and indicate `genericplc => true`. This will cause Asterisk to attempt to interpolate any packets that are missed. (Enabling this functionality will incur a small performance penalty.)

dnsmgr.conf

This file is used to configure whether Asterisk should perform DNS lookups on a regular basis, and how often those lookups should be performed.

dundi.conf

The DUNDi protocol is used to dynamically look up the VoIP address of a phone number on a network, and to connect to that number. Unlike the ENUM standard, DUNDi has no central authority. The *dundi.conf* file contains DUNDi extensions used to control what is advertised; it also contains the peers to whom you will submit lookup requests and from whom you will accept lookup requests. The DUNDi protocol was explored in Chapter 10.

enum.conf

The Electronic Numbering (ENUM) system is used in conjunction with the Internet's DNS system to map E.164 ITU standard (ordinary telephone) numbers to email addresses, web sites, VoIP addresses, and the like. An ENUM number is created in DNS by reversing the phone number, separating each digit with a period, and appending *e164.arpa* (the primary DNS zone). If you want Asterisk to perform ENUM lookups, configure the domain(s) in which to perform the lookups within the *enum.conf* file. In addition to the official *e164.arpa* domain, you can have Asterisk perform lookups in the publicly accessible *e164.org* domain.

extconfig.conf

Asterisk can write configuration data to and load configuration data from a database using the external configuration engine (also known as *realtime*). This enables you to map external configuration files (static mappings) to a database, allowing the information to be retrieved from the database. It also allows you to map special runtime entries that permit the dynamic creation and loading of objects, entities, peers, and so on without a reload. These mappings are assigned and configured in the *extconfig.conf* file, which is used by both *res_odbc* and *realtime*.

extensions.conf

At the center of every good universe is a dialplan. The *extensions.conf* file is the means by which you tell Asterisk how you want calls to be handled. The dialplan contains a list of instructions that, unlike traditional telephony systems, is entirely customizable. The dialplan is so important that rather than defining it in this appendix, we have dedicated all of Chapters 5 and 6, as well as Appendix B, to this topic. Go forth, read, and enjoy!

features.conf

features.conf, the file formally known as *parking.conf*, contains configuration information related to call parking and call transfers. Call parking configuration options include:

- The extension to dial to park calls (`parkext =>`)
- The extension range to park calls in (`parkpos =>`)
- Which context to park calls in (`context =>`)
- How long a call can remain parked for before ringing the extension that parked it (`parkingtime =>`)
- The sound file played to the parked caller when the call is removed from parking (`courtesytone =>`)
- ADSI parking announcements (`asdipark=yes|no`)

In addition to the call parking options, in this file you can configure the button mappings for blind transfers, attended transfers, one-touch recording, disconnections, and the pickup extension (which allows you to answer a remotely ringing extension).

festival.conf

The Festival text-to-speech engine allows Asterisk to read text files to the end user with a computer-generated voice. Festival is covered in Chapter 10.

iax.conf

Similar to *sip.conf*, the *iax.conf* file is where you configure options related to the IAX protocol. Your end devices and service providers are also configured here. *iax.conf* is covered in detail in Appendix A.

iaxprov.conf

This file is used by Asterisk to allow the system to upgrade the firmware on an IAXy device.

indications.conf

The *indications.conf* file is used to tell Asterisk how to generate the various telephone sounds common in different parts of the world—a dial tone in England sounds very different from a dial tone in Canada, but your Asterisk system will be pleased to make the sounds you want to hear. This file consists of a list of sounds a telephone system might need to produce (dial tone, busy signals, and so forth), followed by the frequencies used to generate those sounds.

By default (and without an *indications.conf* file), Asterisk will use the tones common in North America. You can change the default country for your system by specifying the two-letter country code in the `[general]` section. Supported country codes are listed in the *indications.conf.sample* file located in */usr/src/asterisk/configs*. If you have the required information, your country can easily be added. Here's what the configuration for North America looks like:

```
[general]
country=us
;
[us]
description = United States / North America
ringcadance = 2000,4000
dial = 350+440
busy = 480+620/500,0/500
ring = 440+480/2000,0/4000
congestion = 480+620/250,0/250
callwaiting = 440/300,0/10000
dialrecall = !350+440/100,!0/100,!350+440/100,!0/100,!350+440/100,!0/100,350+440
record = 1400/500,0/15000
info = !950/330,!1400/330,!1800/330,0
```

logger.conf

The *logger.conf* file specifies the type and verbosity of messages logged to the various log files in the */var/log/asterisk/* directory. It has two sections, `[general]` and `[logfile]`.

[general]

Settings under the `[general]` section are used to customize the output of the logs (and can safely be left blank, as the defaults serve most people very well). However, if you love to customize such things, read on.

You can define exactly how you want your timestamps to look through the use of the `dateformat` parameter:

```
dateformat=%F %T
```

The Linux man page for `strftime(3)` lists all of the ways you can do this.

If you want to append your system's hostname to the names of the log files, set `appendhostname=yes`. This can be useful if you have a lot of systems delivering log files to you.

If for some reason you do not want to log events from your queues, you can set `queue_log=no`.

If generic events do not interest you, instruct Asterisk to omit them from the by setting `event_log=no`.

[logfiles]

The `[logfiles]` section defines the types of information you wish to log. There are multiple ranks for the various bits of information that will be logged, and it can be desirable to separate log entries into different files. The general format for lines in the `[logfiles]` section is *`filename`* `=>` *`levels`*, where *`filename`* is the name of the file to save the logged information to and *`levels`* are the types of information you wish to save.

Using `console` for the *`filename`* is a special exception that allows you to control the type of information sent to the Asterisk console.

A sample `[logfiles]` section might look like this:

```
[logfiles]
console => notice,warning,error
messages => notice,warning,error
```

You can specify logging of the following types of information:

`debug`
: Enabling debugging gives far more detailed output about what is happening in the system. For example, with debugging enabled, you can see what DTMF tones the users entered while accessing their voicemail boxes. Debugging information should be logged only when you are actually debugging something, as it will create massive log files very rapidly.

`verbose`
: When you connect to the Asterisk console and set a verbosity of `3` or higher, you'll see output on the console showing what Asterisk is doing. You can save this output to a log file by adding a line such as `verbose_log => verbose` to your *logger.conf* file. Note that a high amount of verbosity can quickly eat up hard drive space.

`notice`
: A *notice* is used to inform you of minor changes to the system, such as when a peer changes state. It is normal to see these types of messages, and the events they indicate generally have no adverse effects on the server.

`warning`
: A *warning* happens when Asterisk attempts to do something and is unsuccessful. These types of errors are usually not fatal, but they should be investigated, especially if a lot of them are seen.

`error`
: *Errors* are often related to Out of Memory errors. They generally indicate serious problems that may lead to Asterisk to crashing or freezing.

manager.conf

The Asterisk Manager interface is an API that external programs can use to communicate with and control Asterisk, much as you would do from the Asterisk console.

> The Manager gives programs the ability to run commands and request information from the Asterisk server. However, it is not very secure—its authentication mechanism uses plain-text passwords, and all connected terminals receive all events. The Asterisk Manager should be used only on a trusted local area network, or locally on the box. The `permit` and `deny` constructs allow you to restrict access to certain extensions or subnets.

Many of the available graphical interfaces to Asterisk—such as the Flash Operator Panel—use the Manager to pull data and determine the status of applications. The *manager.conf* file defines the way programs authenticate with the Manager.

The Manager commands (which you can list by typing **`show manager commands`** at the Asterisk console) have varying degrees of privilege. You can control the read and write permissions for these commands with the use of the `read` and `write` options in the *manager.conf* file.

Here's a sample *manager.conf* file:

```
[general]
enabled = no
port = 5038
bindaddr = 0.0.0.0
```

```
[magma]
secret = welcome
deny=0.0.0.0/0.0.0.0
permit= 192.168.1.0/255.255.255.0
read = system,call,log,verbose,command,agent,user
write = system,call,log,verbose,command,agent,user
```

meetme.conf

MeetMe is one of the more remarkable applications in Asterisk. This rather simple concept has proven to be extremely expensive to implement in every other PBX, but what seems like a big deal to them is simple to Asterisk. Whether by using a dedicated server, or through the use of a service, Asterisk now delivers this functionality as a standard application.

MeetMe conferences can be created either dynamically, with the `d` flag in the `Dial()` application, or statically in the *meetme.conf* file. The format for creating conference rooms is as follows:

```
conf => conference_number[,pin][,administrator_pin]
```

All conferences must be defined under the `[rooms]` section header.

```
[rooms]
conf => 4569
conf => 5060,54377017
conf => 3389,4242,1337
conf => 333,,2424
```

mgcp.conf

The Media Gateway Control Protocol (MGCP) has only primitive support in Asterisk. This is likely due to the fact that SIP has stolen the limelight from every other VoIP protocol (except IAX, of course). Because of this, you should attempt to use Asterisk's MCGP channel in a production environment only if you are prepared to perform extensive testing, are willing to pay to have features and patches implemented within your time frames, and have in-house expertise with the protocol.

Having said that, we are not prepared to pronounce MGCP dead. SIP is not yet the panacea it has been touted as, and MGCP has proven itself to be very useful in carrier backbone environments. Many believe MGCP will fill a niche or void that has not yet been discovered, and we remain interested in it.

modem.conf

The *modem.conf* file is used by Asterisk to communicate with ISDN-BRI interfaces through the ISDN4Linux driver. Since ISDN4Linux lacks many core ISDN features, it is not generally used. For BRI, the most popular add-on seems to be *chan_capi*, available from *http://www.junghanns.net*.

musiconhold.conf

The *musiconhold.conf* file is used to configure different classes of music and their locations for use in Music on Hold applications. Asterisk makes use of the *mpg123* application to play music to channels. You can specify arguments for a class, allowing you to use an external application to stream music either locally or over a network. Recently, native Music on Hold has been implemented, allowing Asterisk to play music without any external processes. If the file is available in the same format as the codec of the active channel, no transcoding will occur.

osp.conf

The Open Settlement Protocol (OSP) is officially documented in ETSI TS 101 321, a European Telecommunication Standards Institute (ETSI) document that came out of the work of the TIPHON working group. As far as we can tell, OSP is another attempt to apply old-style telecom thinking to disruptive technologies.

oss.conf

The *oss.conf* file is used to configure Asterisk to use the Open Sound System (OSS) driver to allow communications with the sound card via the CONSOLE channel. Note that ALSA is now the preferred interface for the CONSOLE channel.

phone.conf

The *phone.conf* file is used to configure a Quicknet PhoneJACK card. The PhoneJACK card seems to provide something like an FXS interface, in that you can plug an analog telephone into it and pass calls through Asterisk.

privacy.conf

The *privacy.conf* file is used to control the maximum number of tries a user has to enter his 10-digit telephone number in the `PrivacyManager()` application. The `PrivacyManager()` application determines if a Caller ID is set for the incoming call. If the user fails to enter his 10-digit number within the number of tries configured in *privacy.conf*, the call is sent to priority `n` + 101 (if it exists). If the Caller ID is set, the application does nothing.

queues.conf

Asterisk provides basic call center functionality via its queueing system, but those who are using it in more mission-critical environments often report that their solutions required customization. You can do this customization in the *queues.conf* file.

The `[general]` section of *queues.conf* contains settings that will apply to all queues. Currently, the only parameter that is supported is `persistentmembers`. If this parameter is set to `yes`, a member that is added to the system via the `AddQueueMember()` application will be stored in the AstDB, and therefore retained across a restart.

You can define a queue by placing its name inside of square brackets (`[]`). Within each queue, the following parameters are available:

`musiconhold`
: This parameter allows you to configure which Music on Hold class (configured in *musiconhold.conf*) to use for the queue.

`announce`
: When a call is presented to a member of the queue, the prompt specified by `announce` will be played to that agent before the caller is connected. This can be useful for agents who are logged into more than one queue. You can specify either the full path to the file, or a path relative to */var/lib/asterisk/sounds/*.

`strategy`
: Asterisk can use six strategies to distribute calls to agents:

  `ringall`
  : The queue rings every available agent and connects the call to whichever agent answers first (this is the default).

  `roundrobin`
  : The queue cycles through the agents until it finds one who is available to take the call. `roundrobin` does not take into account the workload of the agents. Also, because `roundrobin` always starts with the first agent in the queue, this strategy is suitable only in an environment where you want your higher-ranked agents to handle all calls unless they are busy, in which case the lower-ranked agents may get a call.

  `leastrecent`
  : The call is presented to the agent who has not been presented a call for the longest period of time.

  `fewestcalls`
  : The call is presented to the agent who has received the least amount of calls. This strategy does not take into account the actual agent workloads; it only considers the number of calls they have taken (for example, an agent who has had 3 calls that each lasted for 10 minutes will be preferred over an agent who has had 5 calls each lasting 2 minutes).

`random`
: As its name suggests, the random strategy chooses an agent at random. In a small call center, this strategy may prove to be the most fair.

`rrmemory`
: The queue cycles through each agent, keeping track of which agent last received a call (this strategy is known as *round-robin memory*). This ensures that call presentation cycles through the agents as fairly as possible.

`servicelevel`
: In a call center, the service level represents the maximum amount of time a caller should ideally have to wait before being presented to an agent. For example, if `servicelevel` is set to 60 and the service level percentage is 80%, that means 80% of the calls that came into the queue were presented to an agent in less than 60 seconds.

`context`
: If a context is assigned to a queue, the caller will be able to press a single digit to exit to the corresponding extension within the configured context, if it exists. This action takes the caller out of the queue, which means that she will lose her place in the queue—be aware of this when you use this feature.

`timeout`
: The `timeout` value defines the maximum amount of time (in seconds) to let an agent's phone ring before deeming the agent unavailable and placing the call back into the queue.

`retry`
: When a timeout occurs, the `retry` value specifies how many seconds to wait before presenting the call again to an available agent.

`weight`
: The `weight` parameter assigns a rank to the queue. If calls are waiting in multiple queues, those queues with the highest `weight` values will be presented to agents first. When you are designing your queues, be aware that this strategy can prevent a call in a lower-weighted queue from ever being answered. Always ensure that calls in lower-weighted queues eventually get promoted to higher-weighted queues to ensure that they don't have to hold forever.

`wrapuptime`
: You can configure this parameter to allow agents a few seconds of downtime after completing a call before the queue presents them with another call.

`maxlen`
: `maxlen` is the maximum number of calls that can be added to the queue before the call goes to the next priority of the current extension.

`announce-frequency`
: The `announce-frequency` value (defined in seconds) determines how often to announce to the caller his place in the queue and estimated hold time.

`announce-holdtime`
: There are three possible values for this parameter: `yes`, `no`, and `once`. The `announce-holdtime` parameter determines whether or not to include the estimated hold time within the position announcement. If set to `once`, it will be played to the caller only once.

`monitor-format`
: This parameter accepts three possible values: `wav`, `gsm`, and `wav49`. By enabling this option, you are telling Asterisk that you wish to record all completed calls in the queue in the format specified. If this option is not specified, no calls will be recorded.

`monitor-join`
: The `Monitor( )` application in Asterisk normally records either end of the conversation in a separate file. Setting `monitor-join` to `yes` instructs Asterisk to merge the files at the end of the call.

`joinempty`
: This parameter accepts three values: `yes`, `no`, and `strict`. It allows you to determine whether callers can be added to a queue based on the status of the members of the queue. The `strict` option will not allow callers to join the queue if all members are unavailable.

`leavewhenempty`
: This parameter determines whether you want your holding callers to be removed from the queue when the conditions preventing a caller from joining exist (i.e., when all of your agents log out and go home).

`eventwhencalled`
: Set `eventwhencalled` to `yes` if you wish to have queue events presented on the Manager interface.

`eventmemberstatusoff`
: Setting this parameter to `no` will generate extra information pertaining to each queue member.

`reportholdtime`
: If you set this parameter to `yes`, the amount of time the caller held before being connected will be announced to the answering agent.

`memberdelay`
: This parameter defines whether a delay will be inserted between the time when the queue identifies a free agent and the time when the call is connected to that agent.

`member => member_name`
: Members of a queue can be either channel types or agents. Any agents you list here must be defined in the *agents.conf* file.

res_odbc.conf

The purpose of the *res_odbc.so* module is to store configuration file information in a database and retrieve that information from the database; however a reload is required to make changes take effect. The *res_odbc.conf* file specifies how to access the table within the database. The *extconfig.conf* file is used to determine how to connect to the database.

rpt.conf

The *rpt.conf* file is used to configure Jim Dixon's newest science project. Jim's Radio Repeater Application (*app_rpt*) allows Asterisk to communicate using VoIP via radio repeater technology. This allows people to efficiently provide large-area coverage of wireless networking and routing information to the Amateur Radio public through their local high-speed Internet connections.

rtp.conf

The *rtp.conf* file controls the Real-time Transport Protocol (RTP) ports that Asterisk uses to generate and receive RTP traffic. The RTP protocol is used by SIP, H.323, MGCP, and possibly other protocols to carry media between endpoints.

The default *rtp.conf* file uses the RTP port range of 10,000 through 20,000. However, this is far more ports than you're likely to need, and many network administrators may not be comfortable opening up such a large range in their firewalls. You can limit the RTP port range by changing the upper and lower bound limits within the *rtp.conf* file.

For every bidirectional SIP call between two endpoints, five ports are generally used: port 5060 for SIP signaling, one port for the data stream and one port for the Real-Time Control Protocol (RTCP) in one direction, and an additional two ports for the data stream and RTCP in the opposite direction.

UDP datagrams contain a 16-bit field for a Cyclic Redundancy Check (CRC), which is used to verify the integrity of the datagram header and its data. It uses polynomial division to create the 16-bit checksum from the 64-bit header. This value is then placed into the 16-bit CRC field of the datagram, which the remote end can then use to verify the integrity of the received datagram.

Setting `rtpchecksums=no` requests that the OS not do UDP checksum creating/checking for the sockets used by RTP. If you add this option to the sample *rtp.conf* file, it will look like this:

```
[general]
rtpstart=10000
rtpend=20000
rtpchecksums=no
```

sip.conf

The *sip.conf* file defines all the SIP protocol options for Asterisk. The authentication for endpoints, such as SIP phones and service providers, is also configured in this file. Asterisk uses the *sip.conf* file to determine which calls you are willing to accept and where those calls should go in relation to your dialplan. Many SIP-related options are configured in *sip.conf*, which was covered in depth in Appendix A.

sip_notify.conf

Asterisk has the ability to reboot a SIP phone remotely by sending it a specially formatted, manufacturer-specific `NOTIFY` message (defined in *sip_notify.conf*) consisting of an event. The phone receives this event, which it interprets as a reboot request. Other phones are supported, but as of this writing only phones by Polycom have been verified to work with this method.

skinny.conf

If you wish to connect to phones using Cisco's proprietary Skinny Client Control Protocol (SCCP), you can use the *skinny.conf* file to define the parameters and channels that will use it. However, since the Asterisk community uses the SIP image on their Cisco phones, you may find it difficult to find community support for this channel type.

voicemail.conf

The *voicemail.conf* file controls the Asterisk voicemail system (called Comedian Mail). It consists of three main sections. The first, called `[general]`, sets the general system-wide settings for the voicemail system. The second, called `[zonemessages]`, allows you to configure different voicemail zones, which are a collection of time and time zone settings. The third and final section is where you create one or more groups of voicemail boxes, each containing the mailbox definitions.

(For more information on adding voicemail capabilities to your dialplan, see Chapter 6.)

General Voicemail Settings

The `[general]` section of *voicemail.conf* contains a plethora of options that affect the entire voicemail system:

`format`
: Lists the codecs that should be used to save voicemail messages. Codecs should be separated with the pipe character (`|`). The first format specified is the format used when attaching a voicemail message to an email. Defaults to `wav49|gsm|wav`.

`serveremail`
: Provides the email address from which voicemail notifications should be sent.

`attach`
: Specifies whether or not Asterisk should attach the voicemail sound file to the voicemail notification email.

`maxmessage`
: Sets the maximum length of a voicemail message, in seconds.

`minmessage`
: Sets the minimum length of a voicemail message, in seconds.

`maxgreet`
: Sets the maximum length of voicemail greetings, in seconds.

`skipms`
: Specifies how many milliseconds to skip forward/back when the user skips forward or backward during message playback.

`maxsilence`
: Indicates how many seconds of silence to allow before ending the recording.

`silencethreshold`
: Sets the silence threshold (what we consider "silence"—the lower the threshold is, the more sensitive it is).

`maxlogins`
: Sets the maximum allowed number of failed login attempts.

`externnotify`
: Supplies the full path and filename of an external program to be executed when a voicemail is left or delivered, or when a mailbox is checked.

`externpass`
: Supplies the full path and filename of an external program to be executed whenever a voicemail password is changed.

`directoryintro`
: If set, overrides the default introduction to the dial-by-name directory.

`charset`
: Defines the character set for voicemail messages.

`adsifdn`
: Specifies the ADSI feature descriptor number to download to.

`adsisec`
: Sets the ADSI security lock code.

`adsiver`
: Indicates the ADSI voicemail application version number.

`pbxskip`
: Causes Asterisk not to add the string `[PBX]:` to the beginning of the subject line of a voicemail notification email.

`fromstring`*:*
: Changes the `From:` string of voicemail notification email messages.

`usedirectory`
: Permits a mailbox owner to select entries from the dial-by-name directory for forwarding and/or composing new voicemail messages.

`pagerfromstring`
: Changes the `From:` string of voicemail notification pager messages.

`emailsubject`
: Specifies the email subject of voicemail notification email messages.

`emailbody`
: Supplies the email body of voicemail notification email messages.

> Please note that both the `emailsubject` and `emailbody` settings can use the following variables to provide more in-depth information about the voicemail:
>
> - `VM_NAME`
> - `VM_DUR`
> - `VM_MSGNUM`
> - `VM_MAILBOX`
> - `VM_CALLERID`
> - `VM_CIDNUM`
> - `VM_CIDNAME`
> - `VM_DATE`

`mailcmd`
: Supplies the full path and filename of the program Asterisk should use to send notification emails. This option is useful if you want to override the default email program.

Voicemail Zones

As voicemail users may be located in different geographical locations, Asterisk provides a way to configure the time zone and the way the time is announced for different callers. Each unique combination is known as a *voicemail zone*. You configure your voicemail zones in the `[zonemessages]` section of *voicemail.conf*. Later, you can assign your voicemail boxes to use the settings for one of these zones.

Each voicemail zone definition consists of a line with the following syntax:

```
zonename=timezone | time_format
```

The *`zonename`* is an arbitrary name used to identify the zone. The *`timezone`* argument is the name of a system time zone, as found in */usr/share/zoneinfo*. The *`time_format`* argument specifies how times should be announced by the voicemail system. The *`time_format`* argument is made up of the following elements:

`'`*`filename`*`'`
: The filename of a sound file to play (single quotes around the filename are required)

`${`*`VAR`*`}`
: Variable substitution

`A` *or* `a`
: The day of the week (Saturday, Sunday, etc.)

`B` *or* `b` *or* `h`
: The name of the month (January, February, etc.)

`d` *or* `e`
: The numeric day of the month (first, second... thirty-first)

`Y`
: The year

`I` *or* `l`
: The hour, in 12-hour format

`H`
: The hour, in 24-hour format—single-digit hours are preceded by "oh"

`k`
: The hour, in 24-hour format—single-digit hours are *not* preceded by "oh"

`M`
: The minute

`P` *or* `p`
: A.M. or .P.M.

`Q`
: "today", "yesterday," or ABdY (note: not standard strftime value)

`q`
: "" (for today), "yesterday", weekday, or ABdY (note: not standard strftime value)

`R`
: 24-hour time, including minutes

For example, the following example sets up two different voicemail zones, one for the Central time zone in 12-hour format, and a second in the Mountain time zone, in 24-hour format:

```
[zonemessages]
central=America/Chicago|'vm-received' Q 'digits/at' IMp
mountain24=America/Denver|'vm-received' q 'digits/at' H 'digits/hundred' M 'hours'
```

Defining Voicemail Contexts and Mailboxes

Now that the system-wide settings and voicemail zones have been set, you can define your voicemail contexts and individual mailboxes.

Voicemail contexts are used to separate out different groups of voicemail users. For example, if you are using Asterisk to host voicemail for more than one company, you should place each company's mailboxes in different voicemail contexts, to keep them separate. You might also use voicemail contexts to create per-department dial-by-name directories.

To define a new voicemail context, simply put the context name inside of square brackets, like this:

```
[default]
```

Inside a voicemail context, each mailbox definition takes the following syntax:

```
mailbox=password,name[,email[,pager_email[,options]]]
```

The *`mailbox`* argument is the mailbox number.

The *`password`* argument is the code the mailbox owner must enter to access his voicemail. If the password is preceded by a minus sign (-), the password may not be changed by the mailbox owner.

The *`email`* and *`pager_email`* arguments are email addresses where voicemail notifications will be sent. These may be left blank if you don't want to send voicemail notifications via email.

The *`options`* argument is a pipe-separated list of voicemail options that may be specified for the mailbox. (These options may also be set globally by placing them in the `[general]` section.) Valid voicemail options include:

`tz`
: Sets the voicemail zone from the `[zonemessages]` section above. This option is irrelevant if `envelope` is set to `no`.

`attach`
: Attaches the voicemail to the notification email (but *not* to the pager email). May be set to either `yes` or `no`.

`saycid`
: Says the Caller ID information before the message.

`cidinternalcontexts`
: Sets the internal context for name playback instead of extension digits when saying the Caller ID information.

`sayduration`
: Turns on/off the duration information before the message. Defaults to `on`.

`saydurationm`
: Specifies the minimum duration to say when `sayduration` is `on`. Default is 2 minutes.

`dialout`
: Specifies the context to dial out from (by choosing option 4 from the advanced menu). If not specified, dialing out from the voicemail system will not be permitted.

`sendvoicemail`
: Specifies the context to send voicemail from (by choosing option 5 from the advanced menu). If not specified, sending messages from within the voicemail system will not be permitted.

`callback`
: Specifies the context to call back from. If not specified, calling the sender back from within the voicemail system will not be permitted.

`review`
: Allows senders to review/rerecord their messages before saving them. Defaults to `off`.

`operator`
: Allows senders to hit `0` before, after, or while leaving a voicemail message to reach an operator. Defaults to `off`.

`envelope`
: Turns on/off envelope playback before message playback. Defaults to `on`. This does not affect option 3,3 from the advanced options menu.

`delete`
: Deletes voicemails from the server after notification is sent. This option may be set only on a per-mailbox basis; it is intended for use with users who wish to receive their voicemail messages *only* by email.

`nextaftercmd`
: Skips to the next message after the user hits 7 or 9 to delete or save the current message. This can be set only globally at this time, not on a per-mailbox basis.

`forcename`
: Forces new users to record their names. A new user is determined by the password being the same as the mailbox number. Defaults to `no`.

`forcegreetings`
: Forces new users to record greetings. A new user is determined by the password being the same as the mailbox number. Defaults to `no`.

`hidefromdir`
: Hides the mailbox from the dial-by-name directory. Defaults to `no`.

You can specify multiple options by separating them with the pipe character, as shown in the definitions for mailboxes 9855 and 6522 below.

Here are some sample mailbox definitions:

```
[default]
; regular mailbox with email notification
101 => 4242,Example Mailbox,somebody@asteriskdocs.org

; more advanced mailbox with email and pager notification and a couple of
; special options
102 => 9855,Another User,another@asteriskdocs.org,pager@asteriskdocs.org,
attach=no|tz=central
```

```
; a mailbox with no email notification and lots of extra options
103 => 6522,John Q. Public,,,tz=central|attach=yes|saycid=yes|
dialout=fromvm|callback=fromvm|review=yes
```

vpb.conf

This file is used to configure Voicetronix cards with Asterisk.

zapata.conf

The *zapata.conf* file is used to define the relationship between Asterisk and the Zaptel driver. Because *zapata.conf* is specific to Asterisk, it is located with the other Asterisk configuration files in */etc/asterisk/*. As with *zaptel.conf*, the *zapata.conf* file contains a multitude of choices reflecting the multitude of hardware it supports, and we won't try to list all of the options here. In this book we've covered only the analog interfaces to the Zaptel driver, as described in Chapter 3.

zaptel.conf

The *zaptel.conf* file is not located with the other Asterisk *.conf* files—the Zaptel driver is available to any application that can make use of it, so it makes more sense to store it in a non-Asterisk-specific directory (*/etc/*). *zaptel.conf* is parsed by the *ztcfg* program to configure the TDM hardware elements in your system. You configure three main elements in the *zaptel.conf* file:

- A way of identifying the interfaces on the card within the dialplan
- The type of signaling the interface requires
- The tone language associated with a particular interface, as found in *zonedata.c*

Be very careful not to plug your FXS module into a telephone line. The voltage associated with the phone line, especially during an incoming call, will be much too high for the module to handle and may permanently damage it, rendering it useless!

Within the *zaptel.conf* file, we define the type of signaling that the channel is going to use. We also define which channels to load. The options in the configuration file are the information that will be used to configure the channels with the `ztcfg` command.

The actual parameters available in the *zaptel.conf* file are quite extensive, as a wide variety of PSTN interfaces make use of the Zaptel telephony engine. Also, as this technology is rapidly evolving, anything we write now may not be accurate by the time you read it. Consequently, we won't try to list all of the options here.

In this book, we have focused on the Zaptel analog interfaces as provided by the Digium TDM400P card (see Chapter 3).