The Asterisk Handbook
Version 2

DRAFT

Mark Spencer
Mack Allison
Christopher Rhodes
The Asterisk Documentation Team

Last Edit Date: 3/30/03

DRAFT

The Asterisk Handbook
Version 2

About this book

Authors:

Mark Spencer
Mack Allison
Christopher Rhodes
The Asterisk Documentation Team

Special thanks to all the users, contributers, and developers who have made Asterisk a reality.

Copyright © 2003 Digium, Inc. All rights reserved. ***This document may not be duplicated, copied, or redistributed in any form, electronic or physical, without the prior written consent of Digium, Inc.***

The latest version of this document may be downloaded for free from http://www.digium.com.

This book was created using OpenOffice, available at http://www.openoffice.org.

DRAFT

Table of Contents

DRAFT

DRAFT

Chapter 1: Introduction

1.1 What is Asterisk?

Officially, Asterisk is an Open Source hybrid TDM and packet voice PBX and IVR platform with ACD functionality. Unofficially, Asterisk is quite possibly the most powerful, flexible, and extensible piece of integrated telecommunications software available. Its name comes from the asterisk symbol, *, which in UNIX (including Linux) and DOS environments represents a wildcard, matching any filename. Similarly, Asterisk the PBX is designed to interface any piece of telephony hardware or software with any telephony application, seamlessly and consistently.

Traditionally, telephony products are designed to meet a specific technical need in a network. However, many applications of using telephony share a great deal of technology. Asterisk takes advantage of this synergy to create a single environment that can be molded to fit any particular application, or collection of applications, as the user sees fit.

DRAFT

Asterisk can, among other things, be used in any of these applications:

Heterogeneous Voice over IP gateway (MGCP, SIP, IAX, H.323)
Private Branch eXchange (PBX)
Custom Interactive Voice Response (IVR) server
Softswitch
Conferencing server
Number translation
Calling card application
Predictive dialer
Call queuing with remote agents
Remote offices for existing PBX

Perhaps more importantly, it can fill all of those roles simultaneously and seamlessly between interfaces.

1.2 Obtaining Asterisk

Released versions of Asterisk can be freely downloaded from ftp://ftp.asterisk.org via anonymous FTP. The preferred method of accessing Asterisk for most installations is via the anonymous repository located at cvs.digium.com, with the CVSROOT of :pserver:anoncvs@cvs.digium.com. For more information, see *Downloading and Installing*.

1.3 Licensing

Asterisk is generally distributed under the terms of the GNU General Public License, or GPL. This license permits you to freely distribute Asterisk in source and binary forms, with or without modifications, provided that when it is distributed to anyone at all, it is distributed with source code (including any changes you make) and without any further restrictions on their ability to use or distribute the code. For more information, refer to the GNU General Public License, included as an appendix.

The GPL does not extend to the hardware or software that Asterisk talks to. For example, if you are using a SIP soft phone as a client for Asterisk, it is not a requirement that that program also be distributed under GPL. Additionally, AGI applications, which are simply launched by Asterisk and communicate

For those applications in which the GNU GPL is not appropriate (because of some sort of proprietary linkage, for example), Digium is the solely capable of licensing Asterisk *outside* of the terms of the GPL at their discression. For more information on licensing Asterisk outside of GPL, contact sales@digium.com.

DRAFT

1.4 Supported Technologies

Asterisk is designed to allow new interfaces and technologies to be added easily. Its lofty goal is to support every kind of telephony technology possible. The latest hardware and protocol compatibility list can be found at http://www.digium.com or http://www.asterisk.org. In general, interfaces are divided into three categories, Zaptel hardware, non-Zaptel hardware, and packet voice:

1.4.1 Zaptel Pseudo TDM interfaces

These interfaces provide integration with traditional and legacy digital and analog telephone interfaces (including connection to the public phone network itself). In addition, Zaptel compatible interfaces support Pseudo-TDM switching between them, to keep latency nearly nonexistent on strictly TDM calls, conferences, etc. Zaptel interfaces are available from Digium (http://www.digium.com) for a variety of network interfaces including PSTN, POTS, T1, E1, PRI, PRA, E&M, Wink, and Feature Group D interfaces among others. Among the hardware available at the time of writing:

T100P - Single span T1 or PRI connection (mixed data/voice permitted)
E100P – Single span E1 or PRA connection (mixed data/voice permitted)
T400P – Quad span T1 or PRI connection (mixed data/voice permitted)
E400P – Quad span E1 or PRA connection (mixed data/voice permitted)
X100P – Single analog PSTN connection
S100U – Single analog POTS connection (USB)
S400P – Single to Quad analog POTS connection (PCI)

Note that for technical reasons, you *must* have at least one Zaptel interface (of any kind) installed in your Asterisk system if you wish to use conferencing.

DRAFT

1.4.2 Non-Zaptel hardware interfaces

These interfaces provide connectivity to the traditional and legacy telephone services, but do not support Pseudo-TDM switching. These include:

ISDN4Linux – Basic Rate ISDN interface for Linux
OSS/Alsa – Sound card interfaces
Linux Telephony Interface (LTI) – Quicknet Internet Phonejack/Linejack
Dialogic hardware[1] – Full-duplex Intel/Dialogic hardware

1.4.3 Packet voice protocols

These are standard protocols for communication over packet (IP and Frame Relay) networks and are the only interfaces that do not require specialized hardware of some kind.

Session Initiation Protocol (SIP)
Inter-Asterisk eXchange (IAX) versions 1 and 2
Media Gateway Control Protocol (MGCP)
ITU H.323[2]
Voice over Frame Relay (VOFR)

DRAFT

1.5 Contributing

Although Asterisk is primarily the work of Digium, its main corporate sponsor, like many Open Source projects, Asterisk grows thanks greatly to contributions, both big and small, from countless individuals. Contributing to Asterisk can be done in many ways:

1 Dialogic hardware is not supported by standard Asterisk but is available as a pay-for add-on for customers with Intel/Dialogic hardware.
2 At the time of writing, H.323 support is freely available as an add-on for Asterisk from third parties

1.5.1 Code Contributions

If you are a developer, you can contribute to the Asterisk codebase through bug fixes, feature enhancements, and new applications and channel drivers. Contributions are typically made as patches against current CVS, and should be submitted in "unified diff" format, which you can generate by executing:

```
# cvs diff -u > mypatch.diff
```

The resulting file (mypatch.diff in the above example) should then be e-mailed to the author (markster@digium.com). Before any patches can be merged with standard Asterisk, the author of the patch is must submit a *copyright disclaimer* which gives Digium (Asterisk's copyright holder) unlimited rights to use the patch. Two versions of the disclaimer are included at the end of this document. Either version may be used (whichever the patch author is more comfortable with). After being filled out and signed, the document should be faxed (and preferably mailed) to Digium. Contact information is available at http://www.digium.com.

DRAFT

1.5.2 Documentation Contributions

Even if you are not a developer, you can contribute to Asterisk in an extremely important way by converting your experience in getting to use Asterisk into a document which can accelerate someone else's entry into the software. Documents can include entries for The Asterisk Handbook (commonly referred to as simply "the book"), application notes for using Asterisk in a specific environment or for a specific use (known as "App Notes"), or in documenting Asterisk's programming API.

1.5.3 Asterisk IRC Channel and Mailing List

One important way to contribute is by assisting in programming discussions, and providing technical support for other Asterisk users on the Asterisk IRC channel, or on the Asterisk mailing list. The Asterisk IRC channel is called (not surprisingly) "#asterisk" and is available on irc.freenode.net. More information on the Asterisk mailing list is available at http://lists.digium.com.

1.5.4 Supporting Asterisk Sponsors

Shameless plug as it may be, when you purchase hardware, support or development from Digium, Asterisk's primary corporate sponsor, you directly benefit the advancement of Asterisk.

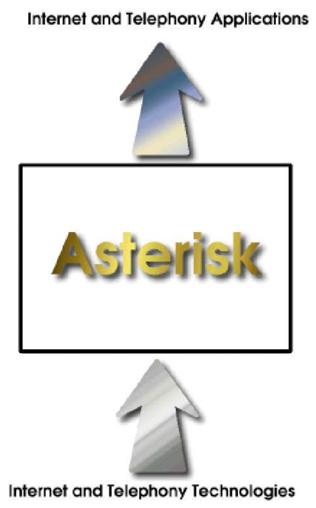
1.5.5 Core Developer Wishlists

Several independent core Asterisk developers have "wishlists" at companies such as Amazon, ThinkGeek, and others. Sponsoring their wishlists is one way to encourage them to continue their contributions and participation in Asterisk development.

DRAFT

Chapter 2: Asterisk's Architecture

2.1 Asterisk Architecture Overview

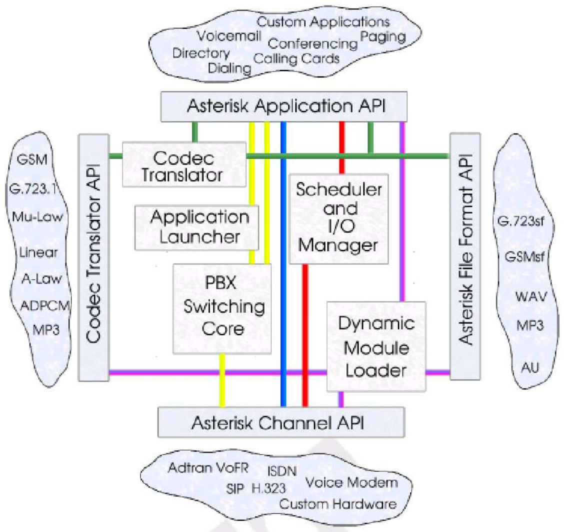

Internet and Telephony Applications
Asterisk
Internet and Telephony Technologies

Asterisk's architecture is fundamentally very simple, but different from most telephony products. Essentially Asterisk acts as middleware, connecting telephony technologies on the bottom to telephony applications on top, creating a consistent environment for deploying a mixed telephony environment. Telephony technologies can include VoIP services like SIP, H.323, IAX, and MGCP (both gateways and phones), as well as more traditional TDM technologies like T1, ISDN PRI, analog POTS and PSTN services, Basic Rate ISDN (BRI), and more. Telephony applications include things such as call bridging, conferencing, voicemail, auto attendant, custom IVR scripting, call parking, intercom, and more.

DRAFT

2.2 Detailed Asterisk Architecture

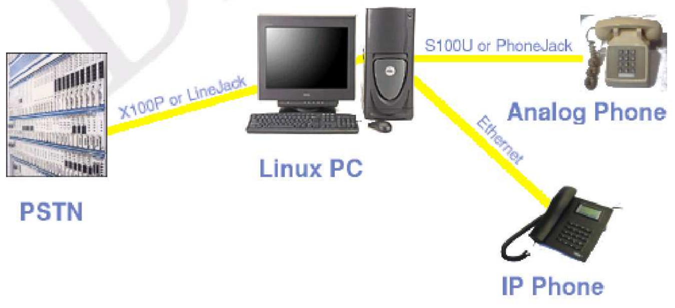
Asterisk's core contains several engines that each play a critical role in the software's operation. When Asterisk is first started, the *Dynamic Module Loader* loads and initializes each of the drivers which provide channel drivers, file formats, call detail record back-ends, codecs, applications and more, linking them with the appropriate internal APIs. Then, Asterisk's *PBX Switching Core* begins accepting calls from interfaces and handling them according to the dialplan, using the *Application* Launcher for ringing phones, connecting to voicemail, dialing out outbound trunks, etc. The core also provides a standard *Scheduler and I/O Manager* that applications and drivers can take advantage of. Asterisk's *Codec Translator* permits channels which are compressed with different codecs to seamlessly talk to one another. Most of Asterisk's usefulness and flexibility come from the applications, codecs, channel drivers, file formats, and more, which plug into Asterisk's various programming interfaces.



2.3 Network Examples

Asterisk is extremely flexible in the networks that can be built, but it's probably useful to present a few sample network diagrams.

2.3.1 The Mythical 1x1 PBX

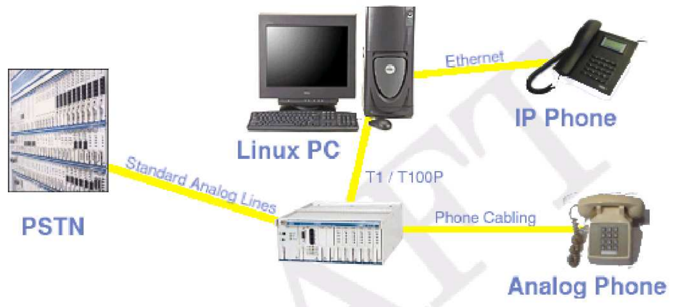


One question that is often heard is "How small of a PBX can you build with Asterisk?". Well, you can make a PBX as small as one port of PSTN and one port of analog or IP phone. Yes, it is true that you can make a PBX with just one port, but it isn't very useful unless

DRAFT

you just enjoy leaving yourself voicemail or talking to an autoattendant. In the above diagram an analog phone could be connected directly to PC running Asterisk, and in turn either to an IP phone over ethernet, or to an analog phone over an S100U USB to FXS converter, for example. Such a PBX can provide voicemail, act as a gateway, or provide some sort of basic IVR script, (such as controlling your lights with X10, for all you home automation geeks out there).

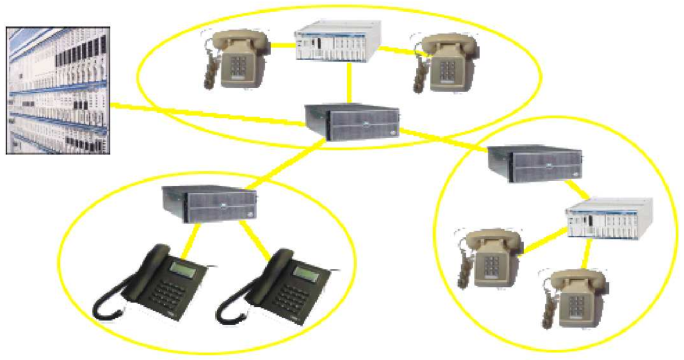
2.3.2 An 8x16 Small Office PBX



DRAFT

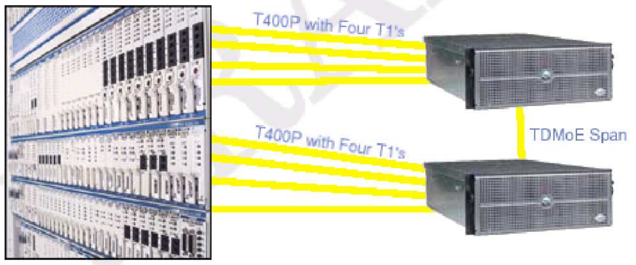
A more typical office scenario is to provide greater density of phones on the inside of phone network than is presented on the outside of a network. In the above example, eight phone lines and 16 analog telephones are brought into a *channel bank* which multiplexes the channels into a single T1 interface, which would then run on a cable just a few feet to a Linux PC (with a T1 card such as the T100P), where Asterisk would provide dialtone. In addition, VoIP phones could be connected via Ethernet, augmenting the number of phones that can be deployed. Even with such a relatively small setup, you can take advantage of call conferencing, voicemail (with individual mailboxes for everyone), the ability to check voicemail over the web, custom IVR scripting, autoattendant, and other features to make all your friends jealous.

2.3.3 SME with Remote Offices



One of Asterisk's most powerful features is its ability to link remote offices of a SME (that's “Small to Medium Enterprise”) together. The above diagram shows how you can build individual small PBX's for multiple offices using Asterisk, and then link them together transparently into a single network.

2.3.4 High Density IVR and Conferencing



Asterisk can be used as a high density IVR and conferencing platform, using traditional PRI/T1 interfaces, and providing redundancy, scalability, and intercommunication using TDM over Ethernet, which permits Asterisk to extend the TDM bus across the ethernet network, while retaining minimal latency.

DRAFT

2.4 Filesystem Organization

Asterisk's organization is designed to follow Linux tradition, and is grouped into several directories.

/etc/asterisk

The */etc/asterisk* directory contains all of Asterisk's configuration files. For more information on configuration files, see the configuration section of this document.

/usr/sbin

The system binary directory */usr/sbin* contains actual Asterisk executables and scripts, including *asterisk*, *astman*, *astgenkey* and *safe_asterisk.*

/usr/lib/asterisk

Contains binary objects related to Asterisk which are architecture specific.

/usr/lib/asterisk/modules

Contains runtime modules for applications, channel drivers, codecs, file format drivers, etc.

/usr/include/asterisk

Contains header files required for building asterisk applications, channel drivers, and other loadable modules.

/var/lib/asterisk

Contains variable data used by Asterisk in its normal operation.

/var/lib/asterisk/agi-bin

Location of AGI scripts to be used by the AGI application in the dialplan.

DRAFT

/var/lib/asterisk/astdb

Asterisk database, roughly the Asterisk equivalent of the “Windows Registry.” This file is never used directly, but its contents can be displayed and modified at the Asterisk command line with the “database” set of functions.

/var/lib/asterisk/images

Storage area for images referenced in dialplan and applications.

/var/lib/asterisk/keys

Storage area for public and private keys used for RSA authentication within Asterisk (especially IAX).

/var/lib/asterisk/mohmp3

Storage area for MP3 music on hold. Should contain any mp3's you want to be available for musiconhold. Note that musiconhold must still be configured in */etc/asterisk/musiconhold.conf*.

/var/lib/asterisk/sounds

Storage area for audio files, prompts, etc. used by Asterisk applications. Some prompts are further organized as subdirectories under the */var/lib/asterisk/sounds* directory.

/var/run

Asterisk stores runtime named pipes and PID files in the system standard /var/run directory

/var/run/asterisk.pid

Contains the primary process identifier (PID) of the running Asterisk process.

DRAFT

/var/run/asterisk.ctl

A named pipe used by Asterisk for enabling the “remote mode” of operation.

/var/spool/asterisk

Used for runtime spooled files of voicemail, outgoing calls, etc.

/var/spool/asterisk/outgoing

Monitored by Asterisk for outbound calls. When a file is created in */var/spool/asterisk/outgoing*, Asterisk parses the file and attempts an outbound call which is then dumped into the PBX if it is answered. For more information see the section “Outbound Calls”

/var/spool/asterisk/qcall

Used for the now deprecated qcall application. Do not use.

/var/spool/asterisk/vm

Storage of voicemail boxes, announcements, and folders

DRAFT

2.5 Naming Channels

Understanding Asterisk channel naming convention is critical to using it effectively. Outgoing channel names (used, for example, with the Dial application) are named in the format:

```
<technology>/<dialstring>
```

The <*technology*> parameter represents which sort of interface one is trying to create or reference (e.g. SIP, Zap, MGCP, IAX, etc). The <*dialstring*> is a driver-specific string representing the destination desired. This section describes the naming convention for each channel type.

2.5.1 Zap: Zaptel TDM Channels

Outgoing:

The basic formats of a Zap channel name are:

```
Zap/[g]<identifier>[c][r<cadence>]
```

Where <*identifier*> is a numerical identifier for the physical channel number of the desired channel. If the identifier is prefixed by the letter *g*, then the number is interpreted as a *group* number instead of as a channel (See Zapata.conf). The identifier may be followed by one or more options. If the letter *r* and a number follow, that number is used as a “distinctive ring” for this dial command (valid numbers are 1-4). If the letter *c* follows, then “Answer Confirmation” is requested, in which the call is not considered answered until the *called* user presses '#'.

Zap/1 – TDM Channel 1
Zap/g1 – First available channel in group 1
Zap/3r2 – TDM Channel 3 with 2nd distinctive ring
Zap/g2c – First available channel in group 2 with confirmation

Incoming:

Incoming Zap channels are labeled simply:

```
Zap/<channel>-<instance>
```

Where <*channel*> is the channel number and <*instance*> is a number from 1 to 3 representing which of up to 3 logical channels associated with a single physical channel this is.

Zap/1-1 – First call appearance on TDM channel 1
Zap/3-2 – Second call appears on TDM channel 3

DRAFT

2.5.2 SIP: Session Initiation Protocol Channels

Outgoing:

Outgoing channels typically take the form:

```
SIP/[<exten>@]<peer>[:<portno>]
```

Where <*peer*> is the name of the peer (or hostname/IP of the remote server), <*portno*> is an optional port number (the default is the SIP standard port 5060), and <*exten*> is an optional extension.

SIP/ipphone – SIP peer "ipphone"
SIP/8500@sip.com:5060 – Extension 8500 at sip.com, port 5060

DRAFT

Incoming:

Incoming SIP channels are of the form:

```
SIP/<peer>-<id>
```

Where <*peer*> is the identified peer and <*id*> is a random identifier to be able to uniquely identify multiple calls from a single peer.

SIP/192.168.0.1-01fb34d6 – A SIP call from 192.168.0.1
SIP/sipphone-45ed721c – A SIP call from peer "sipphone"

2.5.3 IAX: Inter-Asterisk eXchange Channels

Outgoing

Outgoing IAX channels take the form:

```
IAX/[<user>[:<secret>]@]<peer>[:<portno>][/<exten>[@<context>][
/<options>]]
```

Where <*user*> and <*secret*> are optional username and secret to use to connect to the host identified by <*peer*> and an optional port number <*portno*>, optionally requesting a specific extension <*exten*> at an optional context <*context*>, and optionally with <*options*> connection options, of which only "a" is currently defined for "request autoanswer."

IAX/mark:asdf@myserver/6275@default – Call to "myserver" using "mark" as username and "asdf" as password, and requesting extension 6275 in default context

IAX/iaxphone/s/a – Call to "iaxphone" requesting immediate answer.

IAX/guest@misery.digium.com – Call Digium

DRAFT

Incoming:

Incoming IAX channels are of the form:

IAX[[<*username*>@]<*host*>]/<*callno*>

Where <*username*> is the username, if known, <*host*> is the apparent host connecting, and <*callno*> is the local call number.

IAX[mark@192.168.0.1]/14 – Call number 14 from user "mark" at 192.168.0.1

IAX[192.168.10.1]/13 – Call 13 from 192.168.10.1

Chapter 3: Running Asterisk

Running Asterisk is actually rather straight forward. Asterisk, if run with no arguments, is launched as a daemon process. Often, it is useful to execute Asterisk in a verbose, console mode, providing you with useful debugging and state information, as well as access to the powerful Asterisk command line interface.

3.1 Asterisk Command Line Arguments

Like most Linux applications, Asterisk has several command line options. These are tpyically preceeded by a "-", and several options may be specified in a row after a single "-". For example:

```
# asterisk -vvvgc
```

The above example is probably the most commonly used asterisk command line.

-c

Enables console mode. If console mode is enabled, Asterisk will provide a command line that can be used to issue commands and view the state of the system. Implies **-f** as well

-C *<configfile>*

Executes Asterisk with a different configuration file.

-d

Enables extra debugging across all modules.

-f

Prevents Asterisk from daemonizing into the background.

DRAFT

-g

Forces Asterisk to dump core in the unlikely event of a segmentation violation.

-h

Displays basic command line help.

-i

Forces Asterisk to prompt for cryptographic initialization passcodes at startup.

-n

Disables ANSI color support.

-p

Run with a real-time priority.

-q

Run in quiet mode.

-r

Connects to an already running instance of Asterisk.

-v

Causes asterisk to produce more verbose output. More *-v*'s mean more verbose.

-x *<command>*

Executes a command in Asterisk (when combined with -r)

DRAFT

3.2 Asterisk Command Line Interface

The Asterisk command line is one of the most powerful interfaces for obtaining status on a running Asterisk. Although a complete description of all options is beyond the scope of this document, a brief introduction is certainly in order. When running Asterisk with the **-r** or **-c** flag, the user is provided with the Asterisk CLI prompt, which looks, unimpressively, like this:

```
*CLI>
```

or

```
localhost*CLI>
```

DRAFT

In any case, once you are at the command line, you enter instructions by typing them in and pressing enter. The Asterisk CLI includes command completion, available by pressing the *tab* key. The most obvious, and useful, command is `help`, which will show you a list of all the Asterisk CLI commands you can enter:

```
*CLI> help
    add extension Add new extension into context
.
.
.
 zap show channel Show information on a channel
*CLI>
```

For more information about a *specific* command, you can type `help` *<command>*. For example:

```
*CLI> help soft hangup
Usage: soft hangup <channel>
       Request that a channel be hung up.  The
       hangup takes effect the next time the
       driver reads or writes from the channel
*CLI>
```

This shows you that the soft hangup command takes an argument (a channel name) and that it requests the channel be hungup. This command can be used, for example, to hangup any active call in the system.

A few more extremely useful commands:

iax debug:	Enable IAX debugging
mgcp debug:	Enable MGCP debugging
reload:	Reload configuration files
restart when convenient:	Restarts Asterisk when all calls are gone
show agi:	Displays AGI commands
show applications:	Shows all Asterisk apps
show application <*app*>:	Shows usage of a specific Asterik app
show channels:	Shows all active channels
show channel <*channel*>:	Shows information on a specific channel
sip debug:	Enable SIP debugging
stop now:	Stops Asterisk immediately

Chapter 4: The Asterisk Dialplan

The most important part of understanding Asterisk is understanding its dialplan. It is the dialplan which routes every call in the system from its source through various applications, to its final destination. Everything from voicemail, to conferencing, to autoattendant voice menus is done through a consistent concept and logic.

4.1 Introduction to Extension Contexts

4.1.1 Extension Contexts Uses

The dialplan is composed of one or more *extension contexts*. Each extension context is itself simply a collection of extensions. Each extension context in a dialplan has a unique name associated with it. The use of contexts can be used to implement a number of important features including:

Security – Permit long distance calls from certain phones only
Routing – Route calls based on extension
Autoattendant – Greet callers and ask them to enter extensions
Multilevel menus – Menus for sales, support, etc.
Authentication – Ask for passwords for certain extensions
Callback – Reduce long distance changes
Privacy – Blacklist annoying callers from contacting you
PBX Multihosting – Yes, you can have “virtual hosts” on your PBX
Daytime/Nighttime – You can vary behavior after hours
Macros – Create scripts for commonly used functions

The goal of this chapter is to familiarize you with the concepts behind the dialplan, show some examples, and empower you with the knowledge you need to perform neat tricks and impress friends, coworkers, and competitors with your Asterisk-foo like a pro.

4.1.2 Basic Extension Context

An example extension context might look something like this:

default	
Extension	***Description***
101	Mark Spencer
102	Wil Meadows
103	Greg Vance
104	Check voicemail
105	Conference Room
0	Operator

In this example context (called “default”), the first three extensions (101 to 103) all would be associated with ringing phones belonging to various employees. The fourth extension (104) would be associated with allowing someone to check their voicemail. The fifth extension (105) would be associated with a conference room. Finally, the “0” extension would be associated with the operator.

DRAFT

4.1.3 Sample Voice Menu

Another example extension context might look like this:

mainmenu	
Extension	***Description***
s	Welcome message and instructions
1	Sales
2	Support
3	Accounting
9	Directory
#	Hangup

This example, called "mainmenu" has only single digit extensions. The "s" extension is the *start* extension, where the caller begins. This extension would play a message along the lines of "Thank you for calling OurCompany. Press 1 for sales, 2 for support, 3 for accounting, 9 for a company directory, or # to hangup." Each menu option is, in fact, an extension and could either dial someone's real extension, or could send someone to another menu for example.

4.1.4 Pattern Matching

Extensions can also match patterns, instead of being single digits. Patterns to be pattern matched must start with the underscore character ("_") and may use any of the following special characters:

X	– Any digit from 0-9
N	– Any digit from 2-9
[14-6]	– Any a 1,4, 5, or 6
.	– Matches anything

Consider the following context for example:

routing	
Extension	***Description***
_61XX	Dallas Office
_62XX	Huntsville Office
_63XX	Dallas Office
_7[1-3]XX	San Jose Office
_7[04-9]XX	Los Angeles Office

This context (called "routing") splits calls according to their extension to be sent to various servers. In this example, it is assumed all extensions are four digits long (Asterisk has no such requirement, of course, nor is there a requirement that all extensions be the same length. Anyway, anything starting with 61, would be sent to the

Dallas office, 62 would go to the Huntsville office and so on. Anything starting with 71, 72, or 73 would go to San Jose.

4.1.5 Context Inclusion

One extension context can include the contents of another. For example, consider the following contexts:

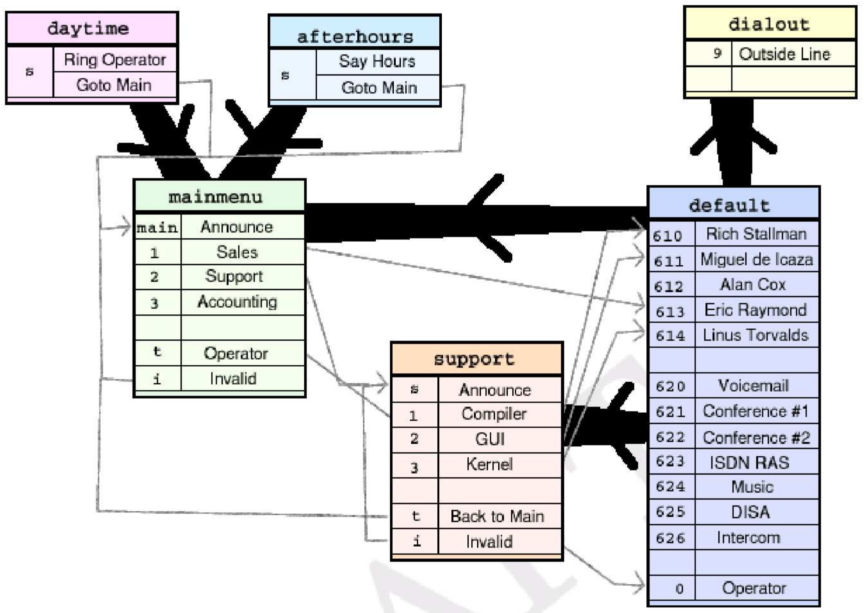
longdistance	
Extension	***Description***
_91NXXNXXXXXX	Long distance calls
include => “local”	

local	
Extension	***Description***
_9NXXXXXX	Local calls
include => “default”	

Here, a context called “local” provides a single extension for dialing local calls, and includes the “default” extension as well. Then, there is a “longdistance” context which includes an extension for long distance calling, and includes the “local” context. Phones which are in the “longdistance” context would be able to make long distance calling. Those in “local” could only make 7 digit local calls, and those in the “default” context would not have outside line access at all. Thus, using extension contexts, you can carefully control who has access to toll services.

DRAFT

4.2 Complete Set of Contexts



A more complete example of a dialplan may be helpful. Here is a dialplan for a fictional Open Source company. Several contexts are listed with familiar names. Thick black lines indicate an extension context being included in another. A slim gray line indicates one extension sending you into another extension or extension context. In general, extensions that are local to your company should be listed under *default* context since they should be accessible by anyone at more-or-less anytime. Typically, no actual phones would be associated with the default context alone, but it could be associated with a VoIP “guest” account, for example. A *local* (in this case *dialout*) context can include the default context as well as either local only or long distance dialing. It is important that access to the local context not be permitted from “guest” accounts, unless it is your intention to let people use your local phone resources. Next we have a couple of menu contexts, *mainmenu* and *support*. Both of these present menus while including the default context so that direct extensions may be dialed at any time. The mainmenu context

DRAFT

includes both *daytime* and *afterhours* so that an incoming call rings to an operator first during the day, and directly to an announcement about company hours in the evening.

4.3 Defining Extensions

Unlike a traditional PBX, where extensions are associated with phones, interfaces, menus, and so on, in Asterisk an extension is defined as a list of *applications* (and arguments) to run. Each step of an extension is referred to as a *priority*. Each priority is generally executed in-order, although applications (especially "Dial" and "Goto" may redirect a call to a different priority. When an extension is dialed, each priority is executed until either the call is hungup, an application returns -1, or the call is routed to a new extension. Each step in an extension is typically notated as follows:

```
exten => <exten>,<priority>,<application>, [(<args>)]
```

4.3.1 Basic Extension Example

Consider the following example:

```
exten => 100,1,Wait(1)
exten => 100,2,Answer
exten => 100,3,Playback(demo-congrats)
exten => 100,4,Hangup
```

DRAFT

This creates an extension with four steps. When a call enters this extension, the first thing that happens is that Asterisk waits for one second. Then, Asterisk answers the call (if it hasn't already been answered). Third, it plays back an audio file called "demo-congrats" and finally it hangs upon the caller. If the caller hungup at any point while Asterisk was processing the extension, processing would be terminated at that point.

4.3.2 Dialing a Phone

The most common sort of extension is that for dialing out another interface. Calling out another interface is done with the “Dial” application. While the “Dial” application has a very extensive list of options (see Dial reference), this example uses it only in its most basic form:

```
exten => 100,1,Dial(Zap/1,20)
exten => 100,2,Voicemail(u100)
exten => 100,102,Voicemail(b100)
```

This example illustrates one of the few exceptions to execution of an extension being out of order. When this extension is entered, the first thing Asterisk does is attempt to dial out the “Zap/1” interface for a maximum of 20 seconds. If the interface is busy, it will jump to priority *n+101* if such a priority exists in this extension. In this case, we have such a priority (102), which sends the caller to voice mailbox 100, preceeded with a “busy” announcement (something like “The person at extension 100 is on the phone”). If there was simply no answer (or if there was a busy and we didn't have a step 102), then execution would continue at step 2, where the caller is put into voice mailbox 100, but with an unavailable announcement (something like “The person at extension 100 is currently unavailable”).

DRAFT

4.3.3 Routing by Caller ID

This example, often known as the “Anti-Ex Girlfriend” extension, shows how Asterisk can route not only by called number, but by *calling* number.

```
exten => 100/2565551212,1,Congestion
exten => 100,1,Dial(Zap/1,20)
exten => 100,2,Voicemail(u100)
exten => 100,102,Voicemail(b100)
```

This example builds upon the previous by adding a special rule that if the caller is 2565551212 (routing by Caller ID is indicated by placing a "/" and the Caller ID number to match immediately following), they are immediately presented with Congestion tone. Other callers proceed normally. A more common example of routing by CallerID is:

```
exten => 100/,1,Zapateller
exten => 100,1,Wait(0)
exten => 100,2,Dial(Zap/1)
```

In this example, if a call is received with *no* Caller ID, then the Zapateller application is run (which plays the familiar "special information tone" which you hear when you call a number that is not in service, often times causing autodialers to disconnect). If Caller ID is provided, then "Wait" is executed for 0 seconds (in otherwords, "do nothing"). In either case, the Zap/1 channel is then rung indefinitely (i.e. No timeout).

DRAFT

4.3.4 Ringing Phones in Sequence

Often it is desired that a given extension first ring one phone, and then if there is no answer, ring another phone (or set of phones). Consider this "Operator" example:

```
exten => 0,1,Dial(Zap/1,15)
exten => 0,2,Dial(Zap/1&Zap/2&Zap/3,15)
exten => 0,3,Playback(companymailbox)
exten => 0,4,Voicemail(100)
exten => 0,5,Hangup
```

In this example, when a caller would dial "0" for the operator, we first trying ringing the interface Zap/1 (which is the phone that the receptionist uses for example). If that interface is busy, or there is no answer after 15 seconds, we try ringing a group of phones (including the receptionist's phone again) for another 15 seconds. If there is still no answer (or if everyone is busy) then it will playback a message announcing that no one is available, and to please leave a message in the company mailbox. Finally the caller is dumped into voice mailbox 100, without having any additional announcement played.

4.3.5 Basic Voice Menu

A voice menu is typically implemented as its own extension context.

```
[sales]
exten => s,1,Background(welcome-sales)
exten => 1,1,Goto(default,100,1)
exten => 2,1,Goto(default,101,1)

[mainmenu]
exten => s,1,Background(welcome-mainmenu)
exten => 1,1,Goto(sales,s,1)
exten => 2,1,Dial,Zap/2
exten => 9,1,Directory(default)
exten => 0,1,Dial,Zap/3
```

DRAFT

An announcement is usually played on the “s” extension, upon entering the menu. Then, the “Background” application plays a prompt, while waiting for the user to enter an extension. The above example presents two menus, one called “mainmenu” and one called “sales.” When a caller entered the “mainmenu” context, they would hear some sort of announcement (like “Press 1 for sales, 2 for support, 9 for a directory, or 0 for an operator”). Upon entering a “1”, the caller would be transferred to the “sales” menu, which would in turn present other options. Dialing 2 would ring Zap/2, 0 would ring Zap/3, and 9 would present the user with a company directory.

4.3.6 Using Variables

Asterisk can make use of global and channel specific variables for arguments to applications. Variables are expressed in the dialplan using ${foo} where “foo” is the name of the variable. A variable may be any alphanumeric string beginning with a letter, but there are some variables whose names have special meanings. Specifically:

${CONTEXT}	– The current context
${EXTEN}	– The current extension
${EXTEN:*x*}	– The current extension with *x* leading digits dropped
${PRIORITY}	– The current priority
${CALLERID}	– The current Caller ID (name and number)
${CALLERIDNUM}	– The current Caller ID number
${CALLERIDNAME}	– The current Caller ID name
${RDNIS}	– The current redirecting DNIS

Global variables may be specified in the [globals] section of the dialplan. Consider the following example:

```
[globals]
MARK => Zap/1
GREG => Zap/2&SIP/pingtel
WIL => Zap/3
JUDY => Zap/4

[mainmenu]
exten => 1,1,Dial(${GREG}&${MARK})
exten => 2,1,Dial(${WIL}&${JUDY})
exten => 3,1,Dial(${JUDY}&${MARK})
```

By organizing the dialplan in this fashion, it is easy to change the physical interfaces for any particular user and have all references to them in the dialplan update instantly as well.

4.3.7 Including Contexts

One context can include zero or more other contexts, optionally with a date/time limitation. Contexts are included in the order they are listed. The format for include is:

DRAFT

```
include => <context>[|<hours>|<weekdays>|<monthdays>|<months>]
```

Where <*context*> is the context to be include, <*hours*> are the hours in which this include is considered valid (in the form of a range, in military time, e.g. 9:00-17:00), <*weekdays*> are the days of the week considered valid (e.g. mon-fri), <*monthdays*> are the days of the month considered valid (e..g 22-25), and <*months*> are the months considered valid. Consider the following example:

```
[salespeople]
exten => 1000,1,Dial(Zap/1)
exten => 1000,2,Voicemail(u1000)
exten => 1001,1,Dial(Zap/2)
exten => 1001,2,Voicemail(u1001)

[techpeople]
exten => 2000,1,Dial(SIP/2000)
exten => 2000,2,Voicemail(u2000)
exten => 2001,1,Dial(SIP/2001
exten => 2001,2,Voicemail(u2001)

[default]
include => salespeople
include => techpeople
```

DRAFT

In this example, the default context simply includes two other contexts, thus making the contexts smaller and easier to track someone down in.

4.3.8 Daytime/Nighttime Modes

Including contexts can be used to implement daytime and nighttime modes (and even holiday modes) by taking advantage of the ability to make includes based upon times and dates. Consider the following example:

```
[newyears]
exten => s,1,Playback(happy-new-years)

[daytime]
exten => s,1,Dial(Zap/1,20)

[nighttime]
exten => s,1,Playback(after-hours-msg)

[default]
include => newyears||||1|jan
include => daytime|9:00-17:00|mon-fri
include => nighttime
```

In this example, the normal mode of operations is the nighttime mode.

4.3.9 Outbound Dialing

Outbound dialing can be done either by directly connecting a short extension (e.g. “9”) with an outbound line, or by establishing full length extensions for numbers to be dialed. Consider the following example:

DRAFT

```
[directdial]
ignorepat => 9
exten => 9,1,Dial(Zap/g2/)
exten => 9,2,Congestion

[international]
ignorepat => 9
exten => _9011.,1,Dial(Zap/g2/${EXTEN:1})
exten => _9011.,2,Congestion
include => longdistance

[longdistance]
ignorepat => 9
exten => _91NXXNXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _91NXXNXXXXXX,2,Congestion
include => local
```

```
[local]
ignorepat => 9
exten => _9NXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _9NXXXXXX,2,Congestion
include => default
```

This example creates 4 separate contexts with various levels of access to the phone network. First, it is assumed that one wants "9" to be the number for connecting to an outside line. The *ignorepat* lines instruct Asterisk's channel drivers not to take away dialtone when that pattern is dialed, so that even after the caller dials 9, they still have a dialtone. The *local* context is able to dial only 7 digit numbers, in addition to anything in the default context. The calls are sent out using any channel in "group 2" of the Zaptel driver, after stripping the "9" off. The *longdistance* context is permitted to dial any 1+ number as well as anything in the local context. The *international* context gives the caller the ability to connect to any number starting with 011+, in addition to anything in the *longdistance* context. The *directdial* context connects a user directly to a trunk when the caller dials 9.

DRAFT

4.3.10 Failover Trunking and LCR

One of Asterisk's most useful cost-saving features is the ability to build simple Least Cost Routing (LCR) tables, including with failover. Consider the following optimized dialplan:

```
[tolllongdistance]
exten => _91NXXNXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
exten => _91NXXNXXXXXX,2,Congestion

[hsvlongdistance]
exten => _91256NXXXXXX,1,Dial(IAX/hsv/${EXTEN})
exten => _91256NXXXXXX,2,Dial(Zap/g2/${EXTEN:1})
exten => _91256NXXXXXX,3,Congestion

[longdistance]
include => hsvlongdistance
include => tolllongdistance
include => local
```

In this example, the long distance context is setup to attempt to use a remote VoIP host called *hsv* (presumably in Huntsville) to dial calls with a 256 area code. Failing that, it will use the TDM group 2 interface (presumably a toll call) to dial (in case the host is unavailable or unreachable for example).

4.3.11 Using Macros

While the Asterisk extension logic is very flexible, it can also be very verbose when creating many extensions which are very similar. In order to ease this task, you can take advantage of *macros* which simplify dialplans and make it easier to modify flows on a large scale. Macros are implemented by creating an extension context whose name begings with "macro-", followed by the name of the macro. Execution begins at the "s" extension and ends as soon as the extension drops to a location that is no longer within the macro. Macros define some useful local variables, specifically:

${MACRO_EXTEN}	– The extension calling the macro
${MACRO_CONTEXT}	– The extension context calling the macro
${MACRO_PRIORITY}	– The active priority when the macro was called
${MACRO_OFFSET}	– If set, causes the macro to attempt to return to *n + ${MACRO_OFFSET}*
${ARG*n*}	– The *n*th argument passed to the macro.

Consider the following example:

```
[macro-oneline]
;
; Standard one-line phone.
;
; ${ARG1} - Device to use
;
exten => s,1,Dial(${ARG1},20)
exten => s,2,Voicemail(u${MACRO_EXTEN})
exten => s,3,Hangup
exten => s,102,Voicemail(b${MACRO_EXTEN})
exten => s,103,Hangup

[macro-twoline]
;
; Standard two-line phone.
;
; ${ARG1} - First phone
; ${ARG2} - Second phone
;
exten => s,1,Dial(${ARG1},20)
exten => s,2,Voicemail(u${MACRO_EXTEN})
exten => s,102,Dial(${ARG2},20)
exten => s,103,Voicemail(b${MACRO_EXTEN})

[default]
exten => 1000,1,Macro(oneline,Zap/1)
exten => 1001,1,Macro(oneline,SIP/1001)
exten => 1002,1,Macro(twoline,Zap/3,Zap/4)
```

DRAFT

After doing the complex work of defining the *oneline* macro for a single line phone and the *twoline* macro for a two-line phone, implementing the default context becomes extremely easy, and each extension requires only a single line instead of several similar lines.

Chapter 5: Configuration Files

5.1 Introduction to Config Files

Most of Asterisk's flexibility is controlled through configuration files located in the /etc/asterisk directory. Its configuration syntax was designed to be easily parseable both by software (like configuration GUI interfaces) and by humans (like, presumably, you). The format of Asterisk config files is , ironically,most similar to the *win.ini* format back in the days of Microsoft® Windows 3.1. The file is a flat ASCII formatted file divided into sections, which are titled with a section name in square brackets, followed by keyword value pairs separated by the equals sign, or equals greater-than. Semicolon is the comment character (since '#' can be useful, especially in extensions). Blank lines are ignored. Here is an example configuration file:

```
;
; The first non-comment line in a config file
; must be a section title
;
[section1]
keyword = value     ; Variable assignment

[section2]
keyword = value
object => value     ; Object declaration
```

DRAFT

Asterisk's configuration parser interprets "=" and "=>" identically, and the syntax is used solely for the benefit of making more obvious to a person reading the file which pairs represent options, and which pairs represent the creation of some sort of object.

5.2 Configuration File Grammars

Although all of Asterisk's configuration files share the same syntax, there are at least three distinct grammars that are typically used.

5.2.1 Simple Groups (e.g. voicemail.conf

The “Simple Groups” format is (not surprisingly) the simplest format and is used by configuration files in which objects are declared with *all options on the same line*. Examples include extensions.conf, meetme.conf, voicemail.conf and others. Consider this example:

```
[mysection]
object1 => option1a,option2a,option3a
object2 => option1b,option2b,option3b
```

In this example, “object1” is created with options “option1a,” “option2a” and “option3a” while “object2” is created with “option1b,” “option2b” and “option3b.”
Individual Entities
The “Individual Entities” configuration syntax is used by configuration files in which objects are declared with many options, and where those options are rarely shared with other objects. In this format, a section is associated with each object (there is sometimes a *general* or similar section for any global configuration options). For example:

```
[general]
globaloption1=globalvalue1
globaloption2=globalvalue2

[object1]
option1=value1a
option2=value2a

[object2]
option1=value1b
option2=value2b
```

In this example, a general section defines two global variables “globaloption1” and “globaloption2” with values “globalvalue1” and

DRAFT

"globalvalue2" respectively. Then, two objects are created ("object1" and "object2") with two options each.

5.2.2 Inherited Option Object (e.g. zapata.conf)

The "Inherited Option Object" format is used by zapata.conf, phone.conf, mgcp.conf and other interfaces in which there are many options, but where most interfaces or objects share the same value for options as others. In this class of configuration file, typically there are one more sections which contain declarations of one or more channels (or objects). The options for the object are specified above the declaration of the object and may be changed afterwords for another object declaration. This is probably one of the more unusual concepts to understand, but once you do, you will almost certainly find it extremely easy to use. Consider this very basic example:

```
[mysection]
option1 = foo
option2 = bar
object => 1
option1 = baz
object => 2
```

DRAFT

The first two lines set the value of options "option1" and "option2" to "foo" and "bar" respectively. When object "1" is instantiated, it is created with its option1 being "foo" and its option2 being "bar." After declaring object "1", we change the value of option1 to "baz" and create a new object "2." Now, object "2" is created with its option1 being "baz" and its option2 remaining "bar" just as with object "1." Again, changing the value of "option1" *after* the declaration of object "1" does not affect its value in object 1, only in object 2.

5.2.3 Complex Entity Object (e.g. iax.conf)

The "Complex Entity Object" format is used by iax.conf, sip.conf, and other interfaces in which there are numerous entities, with many options, which typically do not share a great deal of common settings. Each entity receives its own context (sometimes there is a reserved context such as "general" for global settings). Options are then specified in the context declaration. Consider:

```
[myentity1]
option1=value1
option2=value2

[myentity2]
option1=value3
option2=value4
```

The entity *myentity1* has values *value1* and *value2* for options *option1* and *option2* respectively. Entity *myentity2* has values *value3*and *value4* for options *option1* and *option2* respectively.

5.3 Channel Interfaces

This section defines, in detail, the configuration files for various Asterisk channel drivers.

5.3.1 zapata.conf

Synopsis

The zapata.conf file contains parameters relating to TDM channels provided by the Zaptel interface layer. Channels must be defined in this file before they can be used by Asterisk. In addition, a number of features relating to Asterisk's operation of the channels may be configured here.

DRAFT

Arrangement

The zapata.conf file consists of keyword and value pairs. Keywords set parameters for the operation of channels. They may be boolean (yes/no) or contain values specific to the keyword. Most keywords set parameters for the operation of channels. Values remain in effect for all following channel definitions until they are overidden.

Keywords

These keywords are available in zapata.conf.

context: Defines the initial context for the channel. This will be the context available to the channel upon the initiation of a call. Note that contexts are an important part of maintaining site security. The initial context will govern the availability of extensions to a given channel. If an extension is placed in a different context from the initial context, that extension is unavailable to the caller.

```
context = default
```

Important Note: Careless use of contexts can allow open access to billable services and internal features.

channel: Define a channel or range of channels. Each channel definition will inherit all options stated ahead of it in the file. Channels maybe specified individually, separated by commas, or as a range separated by a hyphen.

```
channel => 16
channel => 2,3
channel => 1-8
```

DRAFT

group: Allows a number of channels to be treated as one for the purpose of dialing. For dialing out, the channels will be called on a first available basis. For the purpose of ringing stations, all channels in the group will ring at once. Multiple group memberships may be specified with commas, and to signify no group membership, the portion after the equals sign may be omitted

```
group = 1
group = 2,3
group =
```

switchtype**:** Sets the type of signalling used for a PRI line. Acceptable values are:

national:	National ISDN
dms100:	Nortel DMS100
4ess:	AT&T 4ESS
5ess:	Lucent 5ESS
euroisdn:	EuroISDN

```
switchtype = national
```

pri_dialplan: Sets an option required for some (rare) switches that require a dialplan parameter to be passed. This option is ignored by most PRI switches. It may be necessary on a few pieces of hardware. Valid options are: *unknown, local*, *private, national,* and *international*.

```
pri_dialplan = national
```

This option can almost always be left unset.

DRAFT

signalling**:** Sets the signaling type for following channel definitions. These parameters should match the channels as defined in /etc/zaptel.conf. Correct choices are based on the hardware available. Asterisk will fail to start if a channel signaling definition is incorrect or unworkable, if the statements do not match zaptel.conf, or if the device is not present or properly configured.
Legal values for signalling are:

fxo_ks: FXO Kewlstart signalling. Used to signal an FXS device within the system, which would normally drive a handset or other station device. Kewlstart is Loopstart with Disconnect Supervision.
fxs_ks: The opposite side of fxo_ks. To signal an internal (or T1 connected) FXO device.
fxo_gs: Use FXO groundstart signalling.
fxs_gs: Use FXS groundstart signalling.
fxo_ls: Use FXO loopstart signalling
fxs_ls: Use FXS loopstart signalling
pri_cpe: Use PRI signalling, customer equipment side. Used when terminating a PRI line into Asterisk channels.
pri_net: Use PRI signalling, network side.
em: Use E&M signalling
em_w: Use E&M wink signalling
featd: Feature Group D, Adtran compatible. For use with the Atlas and similar equipment made by Adtran (DTMF version).
featdmf: Standard Feature Group D (MF version).
featb: Feature Group B

! **Important Note** – Analog phone signalling can be a source of some confusion. FXS channels are signalled with FXO signalling, and vice versa. Asterisk 'talks' to internal devices as the opposite side. An FXO interface card is signalled

```
with FXS signalling by
Asterisk, and should be
configured as such.
```

```
signalling => fxs_ks
signalling => featd
```

Analog Call Progress

These items are used to attempt to emulate having a smarter line (like a PRI) that gives us call progress information, when using analog channels that don't pass us any digital information.

busydetect: Attempt to detect a standard busy signal on analog (FXS and FXO) or certain T1 signalling types (E&M, Wink, Feature Group D). This option can be used to determine when to hang up a call or to have Asterisk handle the busy condition internally. Takes 'yes' or 'no'.

callprogress: Used in combination with a variety of phone lines, enabling call progress will cause Asterisk to attempt to monitor the state of the call, and detect ringing, busy, and answered line. Note that this is not explicitly supported by the line technology, and is subject to errors, especially false answer detection. This only works with US phone tones at the time of writing. Takes yes or no.

```
busydetect = yes
callprogress = yes
```

Multi-link PPP Options (for PRI, requires network support):

These options are used to set adjust multi-link PPP options on PRI lines that support it. Multi-link PPP is a technology that allows channels on a PRI to be dynamically allocated between

DRAFT

voice and data. Asterisk can take voice channels allocated to it, dial a Remote Access Server, and dump the channels into a special extension that delivers the channel to the zaptel data layer. See ZapRAS.

minunused: The minimum number of unused channels available. If there are fewer channels available, Asterisk will not attempt to bundle any channels and give them to the data connection. Takes an integer.

minidle: The minimum number of idle channels to bundle for the data link. Asterisk will keep this number of channels open for data, rather than taking them back for voice channels when needed. Takes an integer.

idledial: The number to dial as the idle number. This is typically the number to dial a Remote Access Server (RAS). Channels being idled for data will be sent to this extension. Takes an integer that does not conflict with any other extension in the dialplan, and has been defined as an idleext.

idleext: The extension to use as the idle extension. Takes a value in the form of 'exten@context'. Typically, the extension would be an extension to run the application ZapRAS.

```
minunused => 2
minidle => 1
idleext => 6999@idle
idledial => 6999
```

Timing Parameters:

These keywords are used only with (non-PRI) T1 lines. All values are in milliseconds. These do not need to be set in most configurations, as the defaults work with most hardware. It has been noted that the common Adtran Atlas uses long winks of about 300

DRAFT

milliseconds, and channels from them should be configured accordingly.

prewink: Sets the pre-wink timing.
preflash: Sets the pre-flash timing.
wink: Sets the wink timing.
rxwink: Sets the receive wink timing.
rxflash: Sets the receive flash timing.
flash: Sets the flash timing.
start: Sets the start timing.
debounce: Sets the debounce timing.

```
rxwink => 300
prewink => 20
```

Caller ID Options:

These keywords set various Caller ID options, including turning certain features off and setting the Caller ID string for channels. Most Caller ID features default to on.

The following three options are boolean (yes/no).

usecallerid: Disables or enables Caller ID transmission for the following channels.

hidecallerid: Sets whether to hide outgoing Caller ID. Defaults to no.

calleridcallwaiting: Sets whether to receive Caller ID during call waiting indication.

```
usecallerid => yes
hidecallerid => no
```

DRAFT

callerid: Sets the caller ID string for a given channel. This keyword takes a properly formatted string containing the name and phone number to be supplied as caller ID. Caller can be set to *asreceived* on trunk interfaces to pass the received Caller ID forward.

Important Note: Caller ID can only be transmitted to the public phone network with supported hardware, such as a PRI. It is not possible to set external caller ID on analog lines. On supported systems, the phone company only receives the number, and supplies the name from their records.

DRAFT

```
callerid = "Mark Spencer" <256 428-6000>
callerid =
callerid = asreceived
```

Call Feature Options

These options enable or disable the availability of advanced call features offered by Asterisk such as three-way calling and call forwarding on FXS (FXO signalled) interfaces. All of these options are boolean (yes/no).

threewaycalling: Sets whether to allow three-way calling from the channel.

cancallforward: Disables or enables call forwarding. Call forwarding is activated with *72 and deactivated with *73.

transfer: Disables or enables flash-hook call transferring. In order for this option to work, *threewaycalling* must also be set to yes.

immediate: When Asterisk is in immediate mode, instead of providing dialtone and reading digits, it immediately jumps into the "s" extension. This is often referred to as *batphone* mode.

adsi: Explicitly enables or disables support for ADSI. The ADSI specification is system similar to Caller ID to pass encoded information to an analog handset. It allows the creation of interactive visual menus on a multiline display, offering access to services such as voicemail through a text interface.

```
threewaycalling = yes
transfer = yes
immediate = no
adsi = yes
cancallforward = yes
```

DRAFT

Audio Quality Tuning Options:

These options adjust certain parameters of Asterisk that affect the audio quality of Zapata channels.

echocancel: Disable or enable echo cancellation. In almost every configuration it is recommended that this be left on (or left unstated, as the default is always on.) Takes 'yes', 'no', or a number of taps. Valid values of taps are 16, 32, 64, 128, or 256.

echocancelwhenbridged:? Enables or disables echo cancellation during a bridged TDM call. In principle, TDM bridged calls should not require echo cancellation, but often times audio performance is improved with this option enabled. Should be set on or left unset. Takes 'yes' or 'no'.

rxgain: Adjusts receive gain. This can be used to raise or lower the incoming volume to compensate for hardware differences. Takes a percentage of capacity, from -100% to +100%

txgain: Adjusts transmit. This can be used to raise or lower the outgoing volume to compensate for hardware differences. Takes the same argument as rxgain.

```
echocancel = yes
echocancelwhenbridged = no
rxgain = 20%
```

Call Logging Options:

These options change the way calls are recorded in the call detail records generated by Asterisk.

amaflags: Sets the AMA flags, affecting the categorization of entries in the call detail records. Accepts these values:

billing: Mark the entry for billing
documentation: Mark the entry for documentation.
omit: Do not record calls.
default: Sets the system default.

accountcode: Sets the account code for calls placed on the channel. The account code may be any alphanumeric string.

```
accountcode = spencer145
amaflags = billing
```

Miscellaneous Options

There are a few other keywords that don't fit neatly into the previous categories.

DRAFT

mailbox: This keyword can be set to allow Asterisk to offer an audible (and visual, if supported by the handset) message waiting indication when the station handset is picked up. When the *mailbox* keyword is defined and an unheard message exists in the associated Inbox, Asterisk will produce a stutter dialtone for one seconds after the phone is picked up. On supported hardware, the message waiting light will be activated. Takes as an argument a mailbox number (which must be defined in voicemail.conf).

language: Turn on internationalization and set the language. This feature will set all system messages to a given language. Though the feature is prepared, English is the only language that has been completely recorded for the default Asterisk installation.

stripmsd: Strip the 'Most Significant Digit,' the first digit or digits from all calls outbound on the given trunk channels. Takes as an argument the number of digits to strip. This option is deprecated, see the application 'StripMSD' or use ${EXTEN:*x*} for this functionality.

Complete File Example:

This is a complete example of a functional zapata.conf file. It is based on an 8 FXO by 16 FXS T1 channel bank.

```
[channels]

;set the FXO's in a group so we can dial out of
;them
;on a first-available basis

group = 1

;set the correct context for our dialout lines

context = pstn

;set the signalling (remember that we signal fxs
;channels
;with fxo, and vice versa)
```

DRAFT

```
signalling = fxs_ls

;set the AMA flags for clarity in the logs

amaflags = documentation

;define the channels that will be covered by the
;previous declarations (in this case all of our
;FXO's)

channel => 1-8

;reset the group, so we don't send outgoing
calls to
;the internal lines

group = 2

;change the context, so we can allow greater
;access to
;services to internal users

context = internal

;set the signalling on the station lines (fxs)
signalling = fxo_ks

;set a mailbox number on the following channels
mailbox = 1234

;set the callerid string (though since we don't
;have a PRI
;it's only seen inside, not on the PSTN.)

callerid = "Dave Schools" <256 555 1234>

;and state the channel this will apply to

channel => 9

;continue and state more channels with mailbox
;indication
;and caller id strings

mailbox = 1235
callerid = "Michael Houser" <256 555 1235>
channel => 10
```

DRAFT

```
mailbox = 1236
callerid = "John Bell" <256 555 1236>
channel => 11

mailbox = 1237
callerid = "Grace Slick" <256 555 1237>
channel => 12

;remember the downward inheritance of options.
;if the next channel doesn't have a voicemail
;box, we need
;to set an empty string, or he'll know whenever
;Grace has a message.  Also the callerid should
;be nulled as well

mailbox =
callerid =

;define a bunch of channels with no other
options

channel => 13-22

;Put this phone in a different context, so we
;can give it
; a different initial dialplan...perhaps a lobby
;phone
;with public access

context = lobby
callerid = "Lobby" <5000>
channel => 23
;and turn the callerid off

callerid =

;we can create a 'hotline' phone by placing a
;phone in a special context
;and setting it to answer immediately. In
;extensions.conf we can route
;the phone to an IVR, direct to security, or
;make it call Steak-Out

context => hotline
immediate => yes
channel => 24
```

DRAFT

5.3.2 sip.conf

Synopsis

The sip.conf file contains parameters relating to the configuration of Session Initiation Protocol (SIP) access to the Asterisk server. Clients must be configured in this file before they can place or receive calls using the Asterisk server.

Arrangement

The sip.conf file is read from the top down. The first section is for general server options, such as the IP address and port number to bind to. The following sections define client parameters such as the username, password, and default IP address for unregistered clients. Sections are delineated by a name in brackets. The first section is called general (which cannot be used as a client name.) The following sections begin with the client name in brackets, followed by the client options.

Keywords

The following keywords are defined in sip.conf.

DRAFT

General Section Keywords:
These settings are for the [general] section of sip.conf and adjust global settings for the SIP stack.

port: The port Asterisk should listen for incoming SIP connections. The default is 5060, in keeping with standards. Takes as an argument a port number (which must not be in use by any other service.)

bindaddr: The IP address Asterisk should listen on for incoming SIP connections. If the machine has multiple real or aliased IP addresses, this option can be used to select which IP addresses Asterisk listens on. The default behavior is to listen on all available interfaces and aliases. Takes as it's argument an IP address (which must be an interface available on the system.)

context: Sets a default context all further clients are placed in, unless overridden within their entity definition.

allow: Explicitly allows a SIP codec. Note that codecs are preferred in the order they are allowed.

disallow: Explicitly disallows a SIP codec from being used.

tos: Configures type of service (TOS) used for SIP and SIP+RTP transmissions. Acceptable values are: *lowdelay, throughput, reliability,* and *mincost*. Also, an integer (0-255) may be specified.

maxexpirey: Maximum permitted length of a registration request in seconds.

defaultexpirey: Default length of a registration request in seconds.

register: Registers this Asterisk instance with another host. Takes a SIP address (without the sip:) optionally followed by a forward slash ('/') and an extension to use for contact.

```
[general]
port = 5060
bindaddr = 192.168.0.1
context = default
disallow = g729
allow = ulaw
allow = gsm
maxexpirey = 180
defaultexpirey = 160
register => 1234@mysipprovider.com/1234
register => 2345@myothersipprovider.com
```

DRAFT

Entity options:

After the general section are listed each entity in the SIP configuration. Entities are divided into three categories:

> *peer*: A SIP entity to which Asterisk sends calls (a SIP provider for example)
> *user*: A SIP entity which places calls through this Asterisk (A phone which can place calls only)
> *friend*: An entity which is both a user and a peer. This make sense for most desk handsets and other devices.

type: The type option sets the connection class for the client. Options are *peer*, *user*, and *friend.*

host: Sets the IP address or resolvable host name of the device. This can alternately be set to *'dynamic'* in which case the host is expected to come from any IP address. This is the most common option, and normally necessary within a DHCP network.

defaultip: This option can be used when the *host* keyword is set to *dynamic.* When set, the Asterisk server will attempt to send calls to this IP address when a call is received for a SIP client that has not yet registered with the server.

username: This option sets the username the Asterisk server attempts to connect when a call is received. Used when for some reason the value is not the same as the username the client registered.

canreinvite: This option is used to tell the server to *never* issue a reinvite to the client. This is used to interoperate with some (buggy) hardware that crashes if we reinvite, such as the common Cisco ATA 186.

context: When defined *within* a client definition, this keyword sets the default context for *this client only.*

DRAFT

dtmfmode: Selects whether DTMF digits should be sent in-band or out of band. Valid values are:

inband: DTMF is send as audio in-band, and is detected in-band.
rfc2833: DTMF is sent out-of-band using RFC2833 (default)
info: DTMF is sent and received out of band using INFO messages (very rarely used)

mailbox: One or more mailboxes may be listed (separated by commas) for sending Message Waiting Indicator (MWI) messages to a given SIP peer.

qualify: A maximum time in milliseconds for a peer to respond. This causes Asterisk to poll the device periodically and consider it down if it takes longer than this number of milliseconds to respond.

secret: A shared secret used for authenticating registrations for peers and for users making calls.

nat: Causes Asterisk to interpret a peer or user as a potentially network address translated host. This is useful when peers are behind firewalls.

DRAFT

! Note that enabling the nat functionality causes Asterisk to violate the RFC specified ways of dealing with Contact: and SDP portions of calls, in order to try to work with NATed hosts. At the time of this writing, nat=yes is incompatible with Pingtel phones.

Complete SIP File Example:

The following is a complete example of a workable sip.conf file.

```
[general]
port=5060
bindaddr=192.168.0.10
context=default
register => 1234@mysipprovider.com

[snom]
type=friend
secret=snom100
host=dynamic
defaultip=192.168.0.15
mailbox=2345,1234

[cisco]
type=friend
secret=mysecret
host=192.168.0.20
canreinvite=no
mailbox=1234
context=trusted
```

DRAFT

5.3.3 iax.conf

Synopsis

This file is used to configure clients connecting via the Inter-Asterisk eXchange protocol. IAX is primarily used for passing calls between Asterisk servers. Frequently Multiple Asterisk servers are configured to intercommunicate with each other using this file. The iax.conf file is shared by both IAX version 1 and version 2 implementations.

Arrangement

The iax.conf file begins with a general section, which sets global server options. Within the general section, we can also configure the

Asterisk server to register as a client with a remote server, for access to the dialplan of another Asterisk system.

Following the general section, clients are defined, one per section. Sections are delineated by their name in brackets.

Keywords

The following keywords are used in iax.conf.

In the *general* section:

port: The port to listen on for incoming connections. The default is port 5036. Takes as it's argument a port number (which must not be in use by another service.)

bindaddr: If multiple IP addresses are available in the same system, this option may be set to bind Asterisk to a single interface.

```
port = 5036
bindaddr = 0.0.0.0
```

amaflags: Sets the AMA flags, affecting the categorization of entries in the call detail records. This keyword may also be set on a per client basis, within their client definition. Accepts these values:

billing: Mark the entry for billing
documentation: Mark the entry for documentation.
omit: Do not record calls.
default: Use the system default.

accountcode: Sets the default account code to log IAX calls to. This keyword can also be used within a client definition to set the account code for that client.

DRAFT

```
accountcode = wmeadows
amaflags = documentation
```

bandwidth: This option is used to control which codecs are used generally. Rather than allowing or disallowing specific codecs, this option may be set to *'low'* to automatically avoid some codecs that don't work well in low bandwidth sitiuations. Takes an option of *low* or *high.*

allow: Specifically allow a certain codec to be used. Takes a codec, or *all.* Using *all* is the same as specifying *bandwidth=high.*

disallow: Specifically disallow a certain codec. See *allow.*

```
bandwidth=low
disallow=all
allow=gsm
```

jitterbuffer: Turn on or off the jitter buffer. The jitter buffer is used to maximize audio quality by balancing latency against the number of dropped packets. A number of keywords exist to fine tune the jitterbuffer.

dropcount: Sets the maximum number of packets to be dropped in order to reduce latency, per memory size.

maxjitterbuffer: Sets the maximum size of the jitterbuffer.

maxexcessjitterbuffer: Sets the the maximum excess jitter buffer, which if exceeded, causes the jitter buffer to slowly shrink in order to improve latency.

register: Register is used to tell the Asterisk server to register with another Asterisk server. This is normally only needed if our local

DRAFT

server has a dynamic IP address and needs to tell the other server where to find it. The format of a register statement is:

```
register => username:secret@server
```

The 'secret' field is optional, if no secret has been specified on the server being connected to. If RSA encryption is in use, specify the key to send to the server with this format:

```
register => username:[key]@server
```

tos: Specify the type of service bits to set on IAX packets, which may improve routing of the packets. Available values are:

lowdelay:	minimize delay
throughput:	maximize throughput
reliability:	maximize reliability
mincost:	use the lowest cost path
none:	use no routing flags

```
tos = lowdelay
```

Draft

Options for Entities

Entity definitions begin with the entity name in brackets. The name is followed by a number of keyword/value pairs applying to the entity in which they are set.

The following keywords are available for users:

type: This sets the type of entity for the client. Valid types are:

user: A user can place calls to or through the Asterisk server.
peer: A peer receives calls from the Asterisk server, but does not place them
friend: A friend both sends and receives calls through the Asterisk server. This makes the most sense for handsets or other station devices. When in doubt use this type.

context: When used within a client definition, this keyword overrides the default incoming context set in the general section for the user only.

callerid: Sets the Caller ID string to be used for this entity. This callerid string will be used internally, and sent to the PSTN if a PRI line is used to route the call to the outside world. If left blank, the Caller ID sent by the entity will be used instead

```
callerid => "Judy" <256 555-1234>
```

DRAFT

auth: Sets the authentication type. IAX supports three methods of authentication. The first (and least secure) is *plaintext*. The passwords (or secrets) are sent in clear text over the network. The second is *md5*, which uses an md5 challenge response algorithm. Both machines will have cleartext access to the passwords, but they will be confirmed using an md5 hash while passing over the network. The most secure option is to use RSA public/private key encryption to store and transmit the secret. Public/private key pairs can be generated using the included program astgenkey. The public key will need to be manually tranfered to the server and stored in /var/lib/asterisk/keys/*name*.pub. Server private keys are stored in the same location as *name*.key.

! **Important Note:** In order use RSA keys with Asterisk, you will have to 'init keys' at the console during startup. Asterisk will prompt you to do so every time it is launched.

inkeys: The public keys to use to decrypt authentication for an incoming client request or registration.

outkey:The private key to encrypt outgoing authentication communication for this client.

```
auth=md5
secret=password
```

```
auth=rsa
inkeys=theirkey
outkey=mykey
```

permit: Hosts to permit to connect as this user. This can be a single host or a host/netmask pair.

deny: Hosts to deny for any incoming connection attempt as this user. *deny* takes the same argument format as *permit.*

```
deny = 0.0.0.0/0.0.0.0
permit=192.168.0.1/255.255.255.0
permit=216.207.245.45
```

host: Sets the expected outgoing host for this client. Can be set to an ip address or *dynamic,* which will allow incoming connections from any host (that is not explicitly denied.)

DRAFT

defaultip: The default IP address for an IAX client. This field is consulted if Asterisk receives a call for an IAX client that is dynamic and has not registered to let Asterisk know the current IP address. Takes as it's argument an IP address.

```
host=dynamic
defaultip=192.168.0.1
```

accountcode: When used within a client definition, sets the account code for that client only. This is used by the call logging service.

qualify: Tells Asterisk whether to test whether the peer is alive before attempting to connect the call. If set to yes Asterisk will periodically contact the peer before forwarding any call information. The argument specified is the maximum number of milliseconds that a peer can take to respond before it is considered "unavailable."

```
qualify=1000
```

mailbox: Provides a mailbox to associate with a given peer, so that when it registers it can be notified of any pending messages waiting.

```
mailbox=1234
mailbox=1002,1003
```

trunk: Enables or disables trunking for a given user or peer. Trunk mode is a more efficient method of operating IAX *if* there are typically many calls running on the link. Trunk mode *requires* having a Zaptel interface in the Asterisk server.

```
trunk=yes
```

DRAFT

Complete File Example

```
[general]
;set up some general items
port=5036

accountcode=iaxcalls
amaflags=default

bandwidth=low
allow=gsm
disallow=lpc10

jitterbuffer=yes
dropcount=3
maxjitterbuffer=500
maxexcessjitterbuffer=100

register =>
asterisk1:opensecret@telco.digium.com

context=iax

;from here on it’s client definitions

[trustedhost]
host=192.168.0.50
trunk=yes
context=trusted

[authhost]
secret=foobar
host=dynamic
defaultip=68.62.178.239

[rsahost]
auth=rsa
inkeys=rsapublickey
host=dynamic
defaultip=216.207.245.55
accountcode=log1234
callerid=”Mark Spencer” <256 428 6000>
```

DRAFT

5.4 Application Configurations

This section details the configuration file syntax for various Asterisk applications.

5.4.1 voicemail.conf

Synopsis

The voicemail.conf file configures system wide parameters for the voicemail system, and stores mailbox information including mailbox number to passcode mapping, box owner names, and e-mail addresses for message received notification.

Arrangement

The voicemail.conf file is arranged in two sections. The first section, *general*, contains system wide parameters such as the formats messages are to be stored in and the address e-mail from the voicemail system should appear to originate from. The second section, *default*, contains the configurations for individual voicemail boxes.

Keywords

The general section takes these keywords and options:

***format*:** Format sets the file formats for saving voicemails. If multiple formats are specified, all formats will be written, and the best available format will be used for playback. The format listed last is used for e-mailing voicemails, if that options is enabled. Available formats are:

gsm : use raw gsm encoding. Best for VoIP.
wav: MS wav format, 16 bit linear
WAV: MS wav format, gsm encoded

DRAFT

g723sf: G.723.1 simple frame (note that Asterisk cannot directly encode , due to licensing issues. It can, however, store and transmit file received from an external source, i.e. from a SIP phone with a built in codec).

```
format=gsm|wav|WAV
```

In this example each received voicemail will be written in gsm, MS-GSM, and linear wav formats.

```
format=gsm
```

This example will store voicemails in raw gsm format only.

serveremail: Serveremail sets the e-mail address that voicemail-waiting e-mails should appear to originate from. This value will be used in the 'From:' field of the e-mail. Available options are any alphanumeric string, or any alphanumeric

Examples:

```
serveremail=asterisk
```

In this example the 'From:' field will be set to 'asterisk'. In most cases the outgoing mail server will append the local hostname.

```
 serveremail=asterisk@myhost.com
```

This example will set the e-mail to . This will normally NOT be rewritten by the outgoing mail server. This is useful if you want the

DRAFT

e-mail to appear to come from a hostname other than the hostname of the local machine.

***append*:** Append set whether to append the voicemail sound file as an attachment to the notification e-mail. Takes an argument of yes or no.

```
append=yes
```

```
append=no
```

The default section takes as a keyword the mailbox number. The keyword takes as parameters the passcode, owner name, and owner e-mail address to send message waiting notification to.

```
1234 => 4321,John Doe,jdoe@misc.com
```

Important Note: The owner name is used by the 'Directory' application to find extensions based on names provided by the caller.

maxmesssage: Sets the maximum length for a voicemail message in seconds. This option can be useful for keeping people from leaving too lengthy of messages.

maxgreet: Sets the maximum length in seconds of the greeting that a user can record for their busy, unavailable, and name messages.

DRAFT

Complete File Example

This is a complete example of a working voicemail.conf file.

```
[general]
format=gsm|wav
serveremail=asterisk@mymachine.com
append=yes
maxgreet=30
maxmessage=90

[default]
1234 => 4321,John Doe,jdoe@mycompany.com
```

DRAFT