

Configuration Guide for Asterisk™ 1.4 and 1.6

A step-by-step guide to building an IP PBX using the most advanced open source IP telephony platform.

Flavio E. Gonçalves



Published by V.Office Networks

Configuration Guide for Asterisk[™] 1.4 and 1.6

Copyright © 2006-2010 V.Office Networks Ltda., All rights reserved

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without prior written consent of the publisher. Exceptions are made for brief excerpts used in publish reviews.

Printing History

First Edition: November 2006,

File Date: Sunday, January 01, 2012

ISBN: 978- 14.528893-6-8

Some manufacturers claim trademarks for several designations that distinguish their products. Wherever those designations appear in this book and we are aware of them, the designation is printed in CAPS or the initials are capitalized. Although a great degree of care was used in writing this book, the author assumes no responsibility for errors and omissions, or damages resulting from the use of the information contained in this book.

We have done the maximum effort to provide trademark information about all the companies and products mentioned in this book by the appropriate use of capitals.

Asterisk, Digium, IAX and DUNDI trademarks are property of Digium Inc.

Preface

This book is for anyone who wants to learn how to install and configure a PBX (Private Branch eXchange) based on Asterisk PBX 1.6. Asterisk is an open source telephony platform capable to use VoIP and TDM channels.

This is the forth generation of the book Asterisk Configuration Guide. The material that I present in this book has helped me to prepare for the dCAP certification from Digium in May 2006 and to pass it in the first try.

The Asterisk Open Source PBX concept is revolutionary. For many years, telephony has been dominated by huge companies with proprietary systems. Finally, users can recover their buying power by having access to an open telephony platform. Thus, things that were not possible before, because they were not economically viable are likely to start happening. Examples include resources such as CTI (computer telephony integration, IVR (interactive voice response), ACD (automatic call distribution), and voicemail, that are now available to everybody.

This book was not designed to teach every single detail of Asterisk. In fact, you will probably not become a guru simply by reading this book. However, you will be able to build and configure a PBX with advanced features such as voicemail, IVR an ACD by the end of reading. I hope you enjoy as much learning about Asterisk as I have enjoyed writing about it.

Notes about this edition

In this edition we had changed all the chapters to reflect the changes for the Asterisk version 1.6. A new chapter about Asterisk Now was included and all the formatting of the book has changed. For ecological reasons, we tried to reduce the number of pages as much as possible reducing the unnecessary white spaces. So even increasing the amount of content in the book we still got an approximate reduction of 20% in the number of pages compared to the last formatting.

License

This eBook is licensed by the author using the creative commons license type Attribution-NonCommercial-NoDerivs 3.0 Unported (CC BY-NC-ND 3.0). Read more at, <http://creativecommons.org/licenses/by-nc-nd/3.0/>

Twitter

<http://www.twitter.com/asteriskguide>

Flavio E. Gonçalves
CEO
V.Office Networks
flavio@asteriskguide.com

Audience

This book is intended for those who are new to Asterisk. We assume you are familiar with Linux, Linux shell commands and Linux text editors. You could test Asterisk using a Linux system with a graphical interface which may be easier for Linux newbies. Some users will try to execute Asterisk using VMWare and this is really not a problem, except for poorer voice quality. For production systems we do not encourage VMWare or Linux with a graphical user interface. It is also desirable that the reader has some knowledge of IP networks, voice over IP (VoIP) and telephony concepts.

Mistakes and errors in the e-Book

We always try to find and eliminate errors and mistakes. Please, if you find something wrong, give us feedback and we will act on it immediately. E-mail address for feedback: flavio@asteriskguide.com

Use as a training material

We use this book for Asterisk training. If you are interested to use it in your training center, please send an e-mail to flavio@asteriskguide.com.

Sponsorship

If you want to sponsor this free eBook, please send an email to flavio@asteriskguide.com. I can allocate a footer to promote your brand or product.

Printed version and Kindle version

You can get a hardcopy of this book at amazon.com:

<http://www.amazon.com/Learning-Guide-Asterisk-1-6-Learn/dp/1452889368/>.

The kindle version can be found at

<http://www.amazon.com/Configuration-Guide-Asterisk-PBX-ebook/dp/B00403N2VM>

Credits

Cover Work:

Karla Braga

Reviewers:

Luis F. Goncalves, Guilherme Goes dCAP, Edit Avenue, professional proofreaders

About the Author

Flavio E. Goncalves was born in 1966 in Brazil. Having always had a strong interest in computers, he got his first personal computer in 1983 and since then it has been almost an addiction. He received his degree in Engineering in 1989 with focus in the computer aided design and computer aided manufacturing.

He is also, CEO of V.Office Networks in Brazil, a consulting company dedicated to the areas of Networks, Security and Telecommunications and a training center since its foundation in 1996. Since 1993, he has participated in a series of certifications programs having being certificated as Novell MCNE/MCNI, Microsoft MCSE/MCT, Cisco CCSP/CCNP/CCDP, Asterisk dCAP and some others.

He started writing about open source software, because, he thinks the way certification programs were organized in the past, were very good to help learners. Some books today are written by strictly technical people, who, sometimes, do not have a clear idea on how people learn. He tried to use his 15 year experience as instructor to help people learn open source telephony software. His experience with networks, protocol analyzers and IP telephony, combined with the teaching experience, give him an edge to write this book. This is the second book he writes; the first one was the Configuration Guide for Asterisk PBX.

As the CEO of V.Office, Flavio E. Goncalves, balance his time between family, work and fun. He is a father of two children and lives in Florianopolis, Brazil, one of the most beautiful places in the world. He dedicates his free time in water sports such as surfing and sailing.

Writing this book has been a process that involved many people. I would like to thank the staff at V.Office Networks in all the process of reviewing and editing the book. I would like to thank Guilherme Goes by the countless tips on Asterisk and the book itself. I would also like to thank several students, who took courses of Asterisk for their feedback, more than a thousand users have already taken classes using this material in the last five years. Finally, I would like to thank my family, for all the support they gave me during all these years.

You can contact him at flavio@asteriskguide.com, or visit his website www.asteriskguide.com.

Summary

Introduction to Asterisk™ PBX	1
Objectives	1
What is Asterisk	1
What is AsteriskNOW	2
Role of Digium™	2
The Zapata project and its relationship with Asterisk	3
Why Asterisk?	3
Extreme cost reduction	4
Telephony system control and independence	4
Easy and rapid development environment	4
Feature rich	4
Dynamic content on the phone	4
Flexible and powerful dial plan	4
Open-source running on top of Linux	5
Asterisk architecture limitations	5
Main objections to Asterisk PBX	5
Asterisk's market share is too small	5
If it is free, how does the manufacturer survive?	5
It is hard to find technical support!	6
Does Asterisk support more than 200 extensions?	6
Only "geeks" are able to install Asterisk	6
What if the server fails?	6
Our company does not use open-source software	6
Using the PC's CPU to process signalling and media is not recommended	6
Asterisk Architecture	7
Channels	7
Codec and codec translation	8
Protocols	9
Applications	9
Overview of an Asterisk system	9
Comparing the old and the new world	10
Telephony using Asterisk	11

Building a test system	12
One FXO, one FXS	12
VoIP Service Provider: ATA	12
Inexpensive FXO card or ATA	12
Asterisk scenarios	13
IP PBX	13
IP-enabling legacy PBXs	13
Toll Bypass	14
Application Server (IVR, Conference, Voicemail)	15
Media Gateway	15
Contact Center Platform	16
Finding information and help	17
Additional references: Non-official websites	17
Mailing lists	17
Summary	18
How to download and install Asterisk	19
Objectives	19
Minimum Hardware Required	19
Hardware configuration	20
IRQ sharing	21
Choosing a Linux distribution	21
Required dependencies	21
Installing Linux for Asterisk	22
Preparing Linux for Asterisk	23
Which version to choose	23
Obtaining and compiling Asterisk	23
Starting and stopping Asterisk	27
Installation directories	29
Log files and log rotation	30
Starting Asterisk with a non-root user	32
Uninstalling Asterisk	33
Asterisk installation notes	33
Summary	34
Quiz	34
Building a simple PBX	36
Objectives	36
Understanding the configuration files	36

Grammars	37
Options to build a LAB for Asterisk	38
Installation Sequence	39
Configuration of the extensions	40
SIP extensions	41
IAX Extensions	42
Configuring the SIP devices	44
Configuring the IAX devices	45
Configuring a PSTN interface	46
Analog lines using DAHDI	47
Connecting to the PSTN using a VoIP provider	48
Dial plan introduction	49
The structure of the file extensions.conf	49
The section <code>[general]</code>	49
The section <code>[globals]</code>	50
Contexts	50
Extensions	51
Special extensions	53
Variables	54
Global variables	55
Channel-specific variables	55
Environment-specific variables	56
Application-specific variables	56
Expressions	56
Operators	57
LAB. Evaluate the following expressions:	58
Functions	59
String concatenation	59
Applications	60
Answer()	60
Dial()	60
Hangup()	63
Goto()	63
Building a dial plan	63
Dialing between extensions	63
Dialing to an external destination	64
Dialing 9 to get a PSTN line	64

Receiving a call in the operator extension	64
Receiving a call using direct inward dialing (DID)	65
Playing several extensions simultaneously	65
Routing by Caller ID	65
Using variables in the dial plan	65
Recording an announcement	66
Receiving the calls in an digital receptionist	66
Summary	68
Quiz	68
Analog channels	70
Objectives	70
Telephony basics	70
PSTN interfaces	71
Analog FXS, FXO, and E&M interfaces	72
Asterisk telephony channels setup	74
Configuration Procedure (valid in both cases)	74
Configuration options	78
Echo cancellation	80
Call progress options	81
DAHDI channel format.	82
Quiz	83
Digital channels	86
Objectives	86
E1/T1 digital lines	86
How is the voice converted to bits?	87
Time Division Multiplexing	88
T1/E1 Line code	89
T1/E1 Signaling	89
ISDN BRI	90
Choosing a telephony card for your Asterisk server	90
Data bus	90
Using hardware echo cancellation	91
Type of signaling	91
Zaptel and DAHDI	92
Asterisk telephony channels setup	92
Automatic detection and configuration	93
Manual configuration	93

Loading the kernel drivers	97
Troubleshooting	97
Configuration options in chan_dahdi.conf	103
General options (channel independent)	103
ISDN options	103
CallerID options	104
Audio quality options	105
Billing options	105
MFC/R2 configuration	106
Understanding the problem	106
MFC/R2 sequence	109
How to use the driver libopenr2	110
Debugging OpenR2	113
MFC/R2 Configuration	116
ANI and DNIS	117
DAHDI channel format	117
Questions	118
Designing a VoIP network	120
Objectives	120
VoIP benefits	120
Asterisk VoIP architecture	121
VoIP protocols and the ISO Open Systems Interconnect (OSI) model	122
How to choose a protocol	123
Peers, Users, and Friends	125
Codecs and codec translation	125
How to choose a Codec	126
Overhead caused by protocol headers	127
Traffic Engineering	128
Reducing the bandwidth required for VoIP	132
Summary	134
Quiz	134
The IAX Protocol	137
Objectives	137
IAX design	137
Bandwidth usage	138
Channel naming	139
Outbound channels example:	139

The format of an incoming IAX channel is:	140
Using IAX	140
Connecting a soft-phone using IAX	140
Connecting to a VoIP provider using IAX	143
Connecting to a provider using IAX	144
Connecting two Asterisk servers through an IAX trunk	145
IAX authentication	148
Incoming connections	148
IP address restrictions	150
Outbound connections	151
Connecting two Asterisk servers using RSA keys	151
The iax.conf file configuration	153
[General] Section	153
Jitter buffer	154
Frame tagging	155
IAX2 Encryption	155
IAX2 debug commands	155
Summary	158
Quiz	158
The SIP Protocol	161
<hr/>	
Objectives	161
Theory of Operation	161
SIP Register process	162
Proxy operation	163
Redirect operation	164
How Asterisk handles SIP	164
SIP Messages	165
Session description protocol (SDP)	167
SIP advanced scenarios	167
Connecting Asterisk to a SIP provider	167
Connecting two Asterisk servers together using SIP	170
Asterisk domain support	171
Advanced configurations	173
SIP Presence	173
Codec configuration	176
DTMF options	176
Quality of service (QoS) marking configuration	177

SIP authentication	177
RTP options	179
SIP NAT Traversal	179
Full Cone	179
Restricted Cone	180
Port Restricted Cone	180
Symmetric	180
NAT firewall table	181
SIP signaling and RTP over NAT	181
Asterisk behind NAT	182
SIP limitations	184
SIP dial strings	184
SIP CLI commands	184
Quiz	184
<u>Dial Plan advanced features</u>	<u>186</u>
Objectives	186
Simplifying your Dial Plan	186
Dial Plan Security	187
Receiving calls using an IVR menu.	187
The Background() application	187
The Record() application	188
The Playback() application	189
The Read() application	190
The Gotoif() application	191
Lab: Building an IVR menu step-by-step	191
Matching as you dial	192
Lab: Using the Read() application	193
Context inclusion	193
Troubleshooting the message “number not found”	194
Using the switch statement	194
Dial plan processing order	195
The #INCLUDE statement	195
Macros	196
Defining a macro	196
Calling a macro	196
Using Asterisk DB	197
Functions, applications, and CLI commands	197

Implementing Call Forward, DND, and Blacklists	198
Using a blacklist	199
Time-based contexts	200
Time-based messages using gotoiftime()	201
Using DISA to get a new dial tone	201
Limit simultaneous calls	202
Voicemail	202
Using the Voicemailmain() application	204
Voicemail application syntax	205
New feature in version 1.6.x	206
Sending voicemail to e-mail	206
Voicemail Web interface	207
Voicemail notification	208
Lab: Message Notification in the Phone	208
Using the directory application	208
Lab: Using the directory application	209
Lab: Putting it all together	210
Step 1 – Configuring channels	211
Step 2 – Configure the dial plan	212
Step 3 - Receiving calls using an auto-attendant	212
Summary	214
Quiz	214
Using PBX features	216
Objectives	216
Where features are implemented	216
Features implemented by Asterisk	217
Features usually implemented by the dial plan	217
Features usually implemented by the phone	217
The features configuration file	218
Call Transfer	219
Configuration task list	220
Call parking	220
Call pickup	222
Configuration task list	222
Call Conference (Meetme)	222
The meetme() application	223
Meetme configuration file	225

Meetme-related applications	226
Meetme configuration task list	226
Examples	226
Call Recording	227
Using the mixmonitor application	227
Music on hold	228
MOH configuration tasks	230
Application Maps	231
Quiz	231
Call Queues	233
Objectives	233
How queues work?	233
ACD architecture	234
Queues	235
Queue configuration file	235
Members	236
Strategies	236
Agents	236
Agent Groups	237
The configuration file for agents	237
ACD-related applications	238
The application queue()	238
The application agentlogin()	239
The application addQueueMember()	239
Support applications and CLI commands	239
Configuration tasks	240
Configure queue recording	242
Queue operation	242
Advanced resources	243
User menu	243
Penalty	243
Priority	243
The application agentcallbacklogin() is deprecated	243
Queue statistics	244
New in 1.6.2 for advanced users	245
Summary	245
Quiz	245

Asterisk Call Detail Records	247
Objectives	247
Asterisk CDR Format	247
Account codes and automated message accounting	248
Changing the CSV and/or CDR format	249
CDR Storage	249
Storage drivers available	249
CSV Storage	250
Storing in MySQL database	250
Applications and functions	251
CDR(accountcode)	251
CDR(amaflags)	251
NoCDR()	251
ResetCDR()	252
Set(CDR(userfield)=Value)	252
AppendCDRUserField(Value)	252
User authentication	252
Using passwords from voicemail	253
Summary	253
Quiz	253
Extending Asterisk with AMI and AGI	255
Objectives	255
Major ways to extend Asterisk	256
Extending Asterisk with console CLI	256
Extending Asterisk using the System() application	256
Example:	256
What is AMI?	257
What language to use for AMI	257
AMI protocol behavior	257
Packet types	257
Configuring users and permissions	258
Logging in to the AMI	258
Action packets	259
Action commands	259
Event packets	261
Events available	261
Asterisk Gateway Interface	262

Using AGI	264
DeadAGI	267
FASTAGI	267
Changing the source code	267
Summary	267
Quiz	267
Asterisk Real-Time	269
<hr/>	
Objectives	269
How does Asterisk Real Time work?	269
Lab: Installing Asterisk Real/Time	270
Configuring Asterisk Real Time	271
Static configuration section	272
Real Time configuration section	273
Database configuration	273
Building a dial plan using Asterisk Real Time	274
Lab: Installing and creating the database tables	274
Lab: Configuring and testing ARA	276
Summary	278
Quiz	278
Objectives	280
AsteriskNOW	280
Introduction to freePBX	281
AsteriskNOW installation	281
Post-installation procedures	281
IP Address configuration	281
Telephony card configuration	282
Installing extensions on freePBX	283
FreePBX codes	288
Dialing external numbers	289
Creating a trunk using an FXO interface	290
Creating a SIP trunk to a VoIP Provider	291
Creating an outbound route	292
Receiving calls	292
Creating an inbound route	294
Using a digital receptionist	296
Recording an announcement	296

Creating the auto-attendant	297
Creating the IVR itself	297
Creating a conference room	300
Summary	300

1

Introduction to Asterisk PBX

The popularity of ready-to-run distributions such as TrixBOS and AsteriskNOW has recently grown. In this book, we will cover the classic Asterisk, which is the foundation for understanding these distributions. Asterisk PBX is open-source software capable of transforming an ordinary PC into a powerful multiprotocol PBX. In this chapter, we will learn about the possibilities of this new technology and its basic architecture. As it is much simpler to install Asterisk from a ready-to-run distribution, the last chapter will cover AsteriskNOW and its graphical interface called FreePBX.

Objectives

By the end of this chapter you should be able to:

- Explain what Asterisk is and what it does;
- Describe the role of Digium™;
- Recognize the basic architecture of Asterisk and its components;
- Point out several usage scenarios; and
- Identify sources of information and help.

What is Asterisk

Asterisk is an open-source PBX software once installed in a PC's hardware along with the correct interfaces—can be used as a full-featured PBX for home users, enterprises, VoIP service providers, and phone companies. Asterisk is also both an open-source community and a commercial product from Digium™. You are free to use and modify Asterisk to suit your needs.

Asterisk allows real-time connectivity between PSTN and VoIP networks. Since Asterisk is much more than a PBX, you not only have an exceptional upgrade to your existing PBX, but you can also do new things in telephony, such as:

- Connect employees working from home to an Office PBX over broadband Internet;

- Connect several offices in different places over an IP network, private network, or even through the Internet itself;
- Give your employees a voicemail integrated with the web and e-mail;
- Build applications like IVRs that allow connections to your ordering system or other applications;
- Give traveling users access to the company PBX from anywhere with a simple broadband or VPN connection; and
- much more....

Asterisk includes several advanced resources previously only found in high-end systems, such as:

- Music for customers on hold waiting in call queues, supporting media streaming and MP3 files;
- Call queues, whereby a team of agents can answer calls and monitor queues;
- Integration with text-to-speech and voice recognition;
- Detailed records transferred to both text files and SQL databases; and
- PSTN connectivity through both digital and analog lines.

What is AsteriskNOW

Asterisk in its purest form, also known as “classic asterisk” (Debian package denomination) is considered more of a development tool than a finished product by itself. AsteriskNOW is an initiative to transform Asterisk in a soft-appliance. The distribution includes CentOS as the operating system and the FreePBX, which is the most used graphical interface. This distribution is licensed according to the GPL and can be freely downloaded. In 2007, Digium acquired a product called Switchvox targeted to commercial users in the SMB market, which it has been promoting vigorously. You can check out this good piece of software at www.digium.com.

Role of Digium™

Digium, a company located in Huntsville, Alabama, is the creator and primary developer of Asterisk. In addition to being the primary sponsor of Asterisk development, Digium also produces telephony interface cards and other hardware for Asterisk's PBX.

Digium offers Asterisk under three types of license agreements:

- General Public License (GPL) Asterisk. This is the most used version. It includes all features and is free to be used and modified according to the terms of the GPL license.

- Asterisk Business Edition is a more recent version of Asterisk. Some companies use the business edition because they do not want or cannot use the GPL license—usually because they don't want to release their source code together with Asterisk. The GPL license requires that any further code development of a GPL-licensed code be released to the source code.
- Asterisk OEM. This version is mostly used by PBX manufacturers who do not want to reveal to the public that their software is based on Asterisk.

The Zapata project and its relationship with Asterisk

The Zapata project was developed by Jim Dixon, who was also responsible for the development of this revolutionary hardware for use with Asterisk. Note that the hardware is open-source too; as such, it can be used by any company. Today, several companies produce cards compatible with this architecture. More details about the project can be seen at:

<http://www.asteriskdocs.org/modules/tinycontent/index.php?id=10>>

The Zapata project produced an architecture called Zaptel (recently renamed Digium Asterisk Hardware Drivers Interface [DAHDI]). One of the main benefits of this architecture is the ability to use the PC CPU to process media streaming, echo cancellation, and transcoding. In contrast, most existing cards use digital signal processors (DSP) to perform these tasks. The use of the PC CPU instead of dedicated DSPs reduces the board's price dramatically. Thus, these cards are significantly cheaper than previously available interfaces from other manufacturers. On the other hand, these cards require a lot of CPU; a misuse of the PC CPU can significantly impact voice quality. Recently, Digium launched a coprocessor card that uses DSPs to encode and decode G.729 and G.723, allowing better scalability for a large number of channels.

Why Asterisk?

I remember my first contact with Asterisk. Usually, the first reaction to something new—especially something that competes with what you already know—is to reject it! This is exactly what happened in 2003. Asterisk was competing with a solution that I was selling to a customer (4 E1 VoIP Gateway), and it was ten times less expensive than what I was charging for the solution I already knew. This disproportionate price led me to start studying Asterisk in order to identify potential pitfalls and drawbacks. For example, I found that the PC CPU at that time would not support 120 G.729 simultaneous sections. At the end of the day, I won the proposal with my Gateway solution. However, this exercise led me to the discovery that Asterisk could solve a variety of very expensive problems for my customer base. We were in trouble with expensive quotes for IVR, unified messaging, call recording, and dialers; with appropriate dimensioning, the CPU problems could be worked around. Indeed, in just three years Asterisk became the flagship product of my company (I actually decided to open another company just for the

Asterisk business). In my opinion, Asterisk is a revolution in telecommunication that represents to IP telephony what Apache represents to web services.

Extreme cost reduction

If you compare a traditional PBX with Asterisk in regard to digital interfaces and phones, Asterisk is slightly cheaper than those PBXs. However, Asterisk really pays off when you add advanced features such as voicemail, ACD, IVR and CTI. With these advanced features, Asterisk becomes significantly less expensive than traditional PBXs. In fact, comparing Asterisk PBXs with low-end analog PBXs is unfair because Asterisk offers so many features not available in low-end analog systems.

Telephony system control and independence

One of customers' most often-quoted benefits of asterisk is the independence that it provides. Some of today's manufacturers do not even give the customer the system's password or the configuration documentation. With Asterisk's "do-it-yourself" approach, the user achieves total freedom; as a bonus, the user has access to a standard interface.

Easy and rapid development environment

Asterisk can be extended using script languages like PHP and Perl with AMI and AGI interfaces. Asterisk is open-source, and its source code can be modified by the user. The source code is written mostly in ANSI C programming language.

Feature rich

Asterisk has several features that are either not found or optional in traditional PBXs (e.g., voicemail, CTI, ACD, IVR, built-in music on hold, and recording). The costs of these features in some platforms exceed the price of the platform itself.

Dynamic content on the phone

Asterisk is programmed using C language and other languages common in today's development environment. The possibility to provide dynamic content is practically limitless.

Flexible and powerful dial plan

Another Asterisk breakthrough is its powerful dial plan. In traditional PBXs, even simple features like least cost routing (LCR) are either not feasible or optional. With Asterisk, choosing the best route is easy and clean.

Open-source running on top of Linux

One of the greatest features of Asterisk is its community. Several resources are available, including the Asterisk wiki (www.voip-info.org <<http://www.voip-info.org>>), e-mail distribution lists, and forums. As Asterisk becomes increasingly adopted, any bugs found and fixed quickly. Asterisk is probably the most tested PBX software in the world. From versions 1.0 to 1.2, more than 3,000 changes and bugs in the source code were corrected, thereby ensuring a code that is both stable and almost error free.

Asterisk architecture limitations

Some limitations in Asterisk stem from the use of the Zapata telephony design. In this design, Asterisk uses the PC CPU to process voice channels instead of dedicated digital signal processors (DSPs), which are common in other platforms. Although this allows for a huge cost reduction in hardware interface, the system becomes dependent on the PC CPU. My recommendation is to run Asterisk in a dedicated machine and be conservative about hardware dimensioning. You can also use Asterisk in a separate VLAN to avoid excessive broadcasts that consume the CPU (broadcast storms caused by loops or viruses). Some newer interface cards from several vendors are now including DSPs to process echo cancellation, codecs, and other features, which will make Asterisk even better.

Main objections to Asterisk PBX

It is common to hear objections to adopting Asterisk, which we will address here.

Asterisk's market share is too small

The market share is usually measured by the number of PBXs sold. These statistics are generally acquired from the biggest distributors. Asterisk is free software that does not appear in sales statistics. However, independent numbers prove that Asterisk “rocks the world”. According to VoIP-Supply, more than 300,000 systems run Asterisk, and Digium has sold more than 4 million voice interfaces. Last year, the Eastern Management Group concluded that open-source PBXs account for 18% of the market share, with the vast majority of them being Asterisk. In fact, 85% of the open-source PBX market is based on Asterisk, which now ranks second in terms of lines connected to an IP PBX.

If it is free, how does the manufacturer survive?

Actually, there is no such thing as open-source software manufacturer. Digium is a software development company, as well as a community, and has been developing Asterisk since 1999. With more than a hundred employees, it has revenues attached to the sales of telephony interface cards, PBX systems such as Switchvox, and related software. The company has made a profit in the last 24 quarters.

It is hard to find technical support!

Digium provides technical support for those who buy the Asterisk Business Edition. Recently, technical support for open-source Asterisk has become available as well. Hundreds of professionals have already been certified as Digium Certified Asterisk Professional (dCAP) and serve as the first line of support and professional services, much like any IT company.

Does Asterisk support more than 200 extensions?

Yes, absolutely. Asterisk has been used in installations with more than 10,000 users. It is largely scalable using load balancing and failover systems. It is not uncommon to see more than a thousand users on a single server.

Only “geeks” are able to install Asterisk

With AsteriskNOW and freePBX, even professionals with limited knowledge about Linux are able to install and configure a PBX of medium complexity. With the help of a GUI, it is possible to configure an entire PBX in just a few hours.

What if the server fails?

One of the main advantages of Asterisk is its capability to run in fault-tolerant systems. It is relatively simple and inexpensive to have two servers running in parallel. I dare you to try this with a conventional PBX!

Our company does not use open-source software

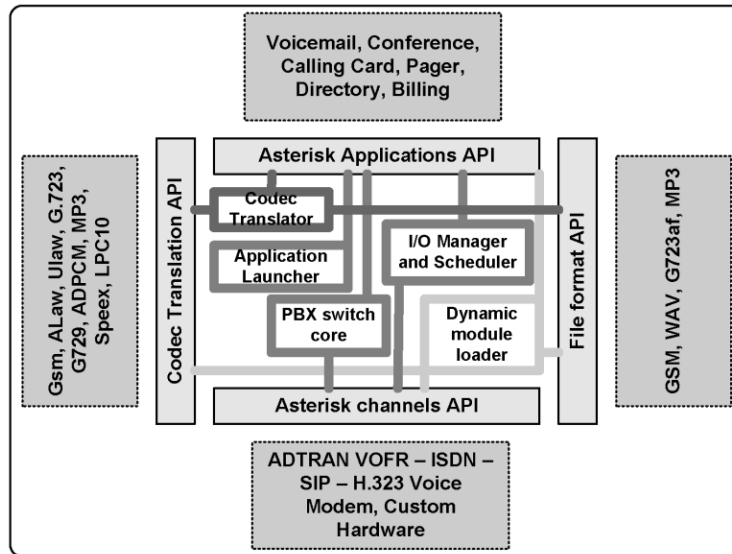
Your company probably uses open-source software without even realizing it. Several appliances use Linux as their operating system. Moreover, you can still license Asterisk commercially using the Asterisk Business Edition.

Using the PC's CPU to process signalling and media is not recommended

Asterisk uses the server's CPU to process signaling and media for voice channels instead of having dedicated DSPs. Although this allows a cost reduction of up to five times, it makes the system dependent on the performance of the main CPU. With the correct dimensioning, Asterisk is capable of handling large volumes. If you still want to release the main CPU from these tasks, you can also use hardware echo cancellation and even transcoder cards, such as the Digium's TC400B based on DSPs.

Asterisk Architecture

This section will explain how Asterisk's architecture works. The figure below shows the basic Asterisk architecture. Next, we will explain architecture-related concepts, including channels, codecs, and applications.



Channels

A channel is the equivalent of a telephone line, but in a digital format. It usually consists of an analog or digital (TDM) signaling system or a combination of codec and signaling protocol (e.g., SIP-GSM, IAX-uLaw). Initially, all telephony connections were analog and susceptible to echo and noise. Later, most systems were converted to digital systems, with the analogical sound converted into a digital format using pulse code modulation (PCM) in most cases. This format allows voice transmission in 64 kilobits/second without compression.

Channels interfacing with the Public Switch Telephony Service (PSTN)

- **chan_dahdi:** Supports cards from Sangoma, Digium, Xorcom, and others
- **chan_mISDN:** Supports ISDN cards based in the Linux ISDN drivers

Channels interfacing with Voice-over IP

- **chan_sip:** Supports voice-over IP using SIP protocol. Dial string: sip/channel
- **chan_iax:** Supports voice-over IP using IAX2 protocol. Dial string: iax2/channel
- **chan_h323:** H.323 is one of the oldest and most implemented voice-over IP protocols. It's useful for connecting to existing H.323 networks. There are

different flavors of H.323 in Asterisk, including `chan_h323`, `chan_oh323`, and `chan_ooH323`. The channel `chan_h323` can be used in Asterisk as a gateway. Asterisk can point to a gatekeeper, but cannot work as one. Dial string `h323/hostname` if using a gatekeeper or `h323/extension@hostname` if going directly to the gateway.

- **chan_mgcp:** Supports the voice-over IP protocol using MGCP. Currently Asterisk supports MGCP phones, but it cannot connect to a VoIP provider using MGCP. Dial string: `MGCP/aaIn/1@hostname`
- **chan_skinny:** Supports Cisco™ voice-over IP skinny protocol. Dial String: `skinny/channel`.

Miscellaneous channels

- **chan_agent:** Used for automatic call distribution (ACD). It is not related to specific hardware or protocol. It can also be used for mobility, allowing any person to use any phone just by logging in to the agent.
- **chan_local:** Is a pseudo channel that simply loops back into the dial plan in a different context. This is useful for recursive routing. Dial string: `Local/extension@context`

Codec and codec translation

We usually try to put as many voice connections as possible in a data network. Codecs enable new features in digital voice, including compression, which is one of the most important features as it allows compression rates larger than 8 to 1. Other features include voice activity detection, packet loss concealment, and comfort noise generation. Several codecs are available for Asterisk and can be transparently translated from one to another. Internally, Asterisk uses `slinear` as the stream format when it needs to convert from one codec to another. Some codecs in Asterisk are supported only in pass-through mode; these codecs cannot be translated. To verify which codecs are installed in your system, you can use the console command:

```
CLI>core show translation
```

The following codecs are supported:

- G.711 ulaw (USA) - (64 Kbps).
- G.711 alaw (Europe) - (64 Kbps).
- G.722 (High Definition) – (64 Kbps)
- G.723.1 - Only pass-through mode
- G.726 - (16/24/32/40kbps)
- G.729 - Needs licensing (8Kbps)
- GSM - (12-13 Kbps)
- iLBC - (15 Kbps)

- LPC10 - (2.5 Kbps)
- Speex - (2.15-44.2 Kbps)

Protocols

Sending data from one phone to another should be easy provided that the data find a path to the other phone on their own. Unfortunately, it doesn't happen this way, and a signaling protocol is necessary in order to establish connections between phones, discover end devices, and implement telephony signaling. It has recently become extremely common to use SIP as a signaling protocol. IAX is another option becoming popular because it works well with NAT traversal and some bandwidth can be saved in trunk mode. Asterisk supports the following protocols.

- SIP
- H323
- IAX2
- MGCP
- SCCP (Cisco Skinny)
- Nortel unistim

Applications

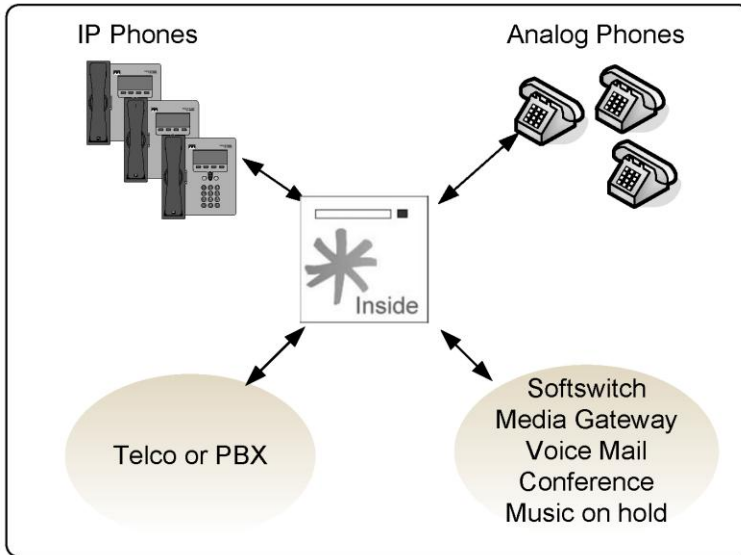
To bridge calls from one phone to another, the application `dialed()` is used. Most Asterisk features (e.g., voicemail and conferencing) are implemented as applications. You can see available Asterisk applications by using the `core show applications` console command.

```
CLI>core show applications
```

You can add applications from Asterisk add-ons, third-party providers, or even those you develop yourself.

Overview of an Asterisk system

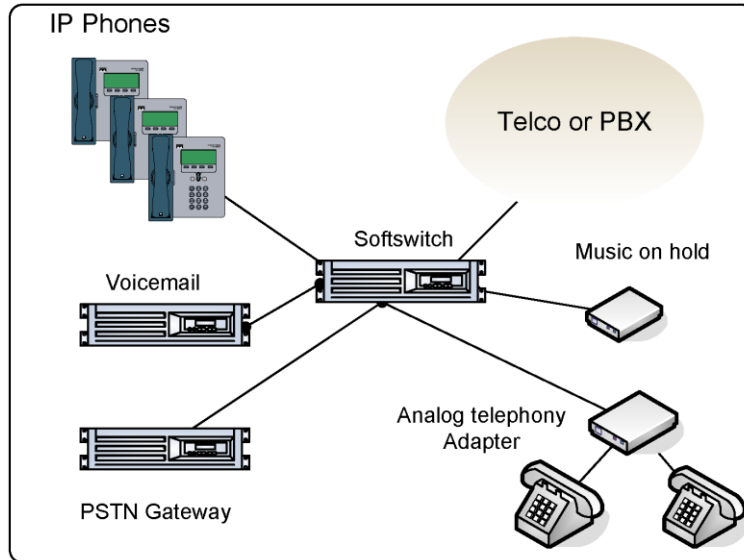
Asterisk is an open-source PBX that acts like a hybrid PBX, integrating technologies such as TDM and IP telephony. Asterisk is ready to implement functionality such as interactive voice response (IVR) and automatic call distribution (ACD); moreover, as previously mentioned, it is open to the development of new applications.



This figure shows how Asterisk connects to the PSTN and existing PBXs using analog and digital interfaces as well as supports analog and IP phones. It can act as a soft-switch, media gateway, voicemail, and audio conference and also has built-in music on hold.

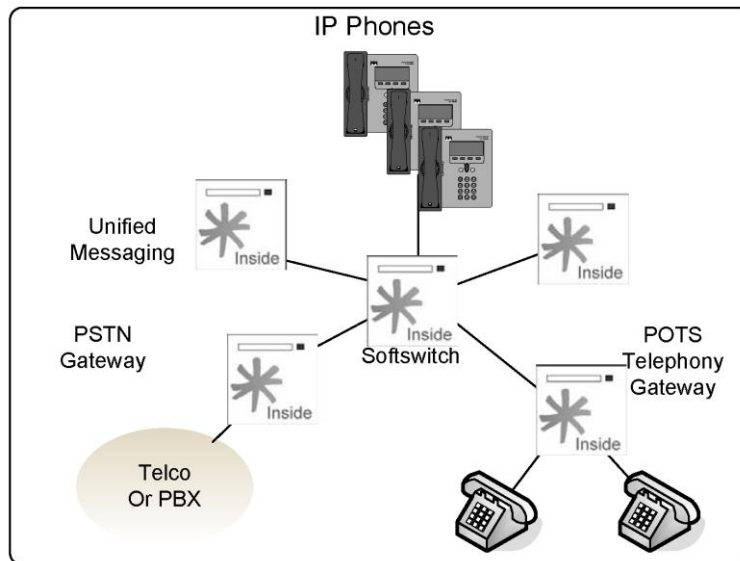
Comparing the old and the new world

In the old soft-switch model, all components were sold separately, meaning you had to purchase each component separately and then integrate to the PBX or soft-switch environment. The costs and risks were high and most of the equipment proprietary.



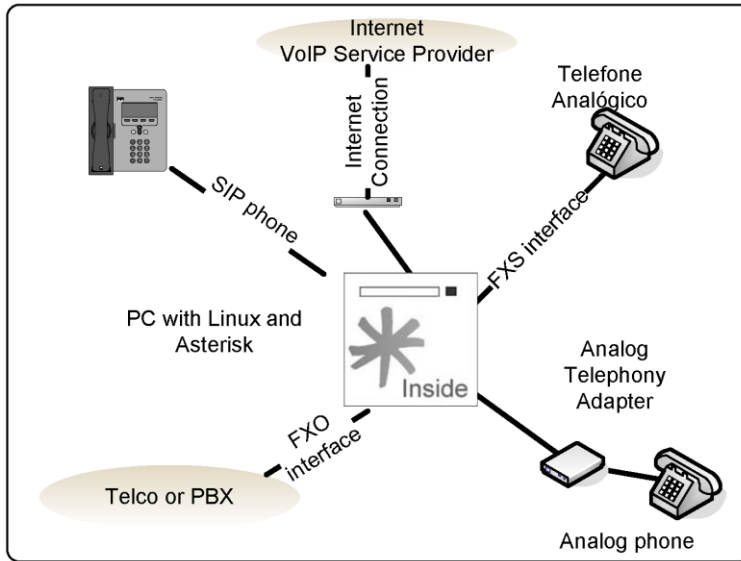
Telephony using Asterisk

All functions are integrated in the Asterisk platform in the same or in different boxes according to the dimensioning, and all are GPL licensed. Sometimes it is easier to install Asterisk than license some of the mainstream IP-PBXs



Building a test system

When implementing an Asterisk solution, our first step is generally to build a test machine. The easiest test machine is the 1x1 PBX, including at least one phone and one line. There are several ways to do this.



One FXO, one FXS

The first and simplest way to build a test machine is to purchase a card with one FXO and one FXS interface. Connect the FXO port to an existing line and connect one FXS to an analog phone. Thus, you have a 1x1 PBX.

VoIP Service Provider: ATA

This is the VoIP option. In this case, you would sign up with a voice service provider to have the SIP trunks and will have to purchase a SIP analog telephony adapter. You will probably spend less than a hundred dollars if you already have the PC.

Inexpensive FXO card or ATA

I started with an inexpensive FXO card. Some inexpensive V.90 fax/modems work with Asterisk as an FXO card. Some of the first Digium cards were created using these cards (e.g., X100P and X101P), which are old modems based on Motorola and Intel chipsets (Motorola 68202-51, Intel 537PU, Intel 537PG, and Intel Ambient MD3200 are known to work). These modems are often incompatible with new motherboards. Recently some

manufacturers started to sell these cards as X100P clones. Some of the incompatibilities can be solved using a patch, more information can be found at:

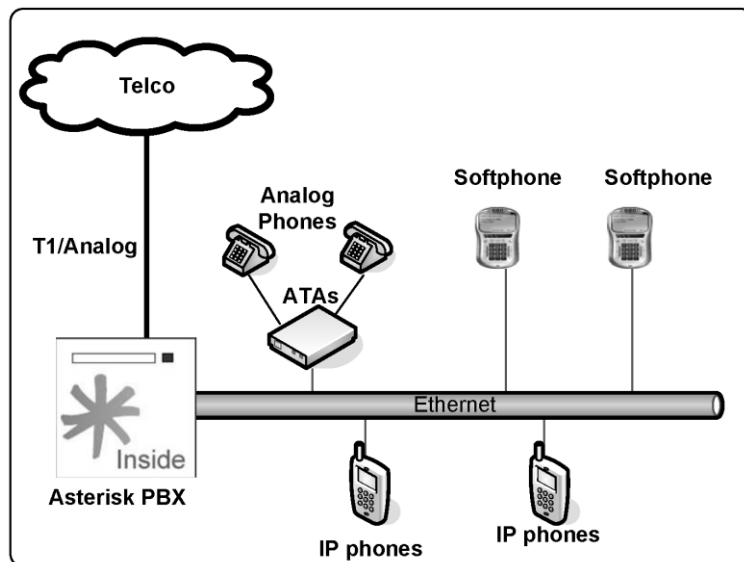
- http://www.asteriskguide.com/mediawiki/index.php/Asterisk_patch_for_the_X100P_card

Asterisk scenarios

Asterisk can be used in several different scenarios. We will list some of them and explain the advantages and possible limitations of each.

IP PBX

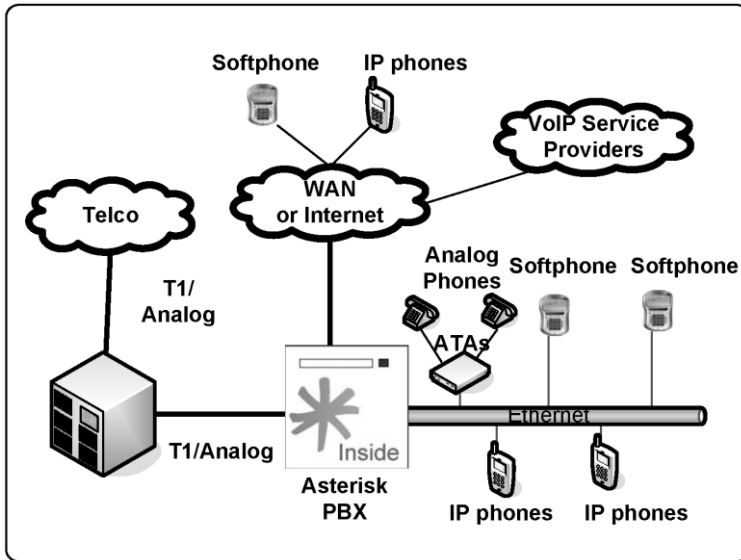
The most common scenario is the installation of a new or the replacement of an existing PBX. If you compare Asterisk with some other alternatives, you will find it to be cheaper and richer in features than most PBXs currently available on the market. Several companies are now changing their specifications to Asterisk instead of other brand-name PBXs.



IP-enabling legacy PBXs

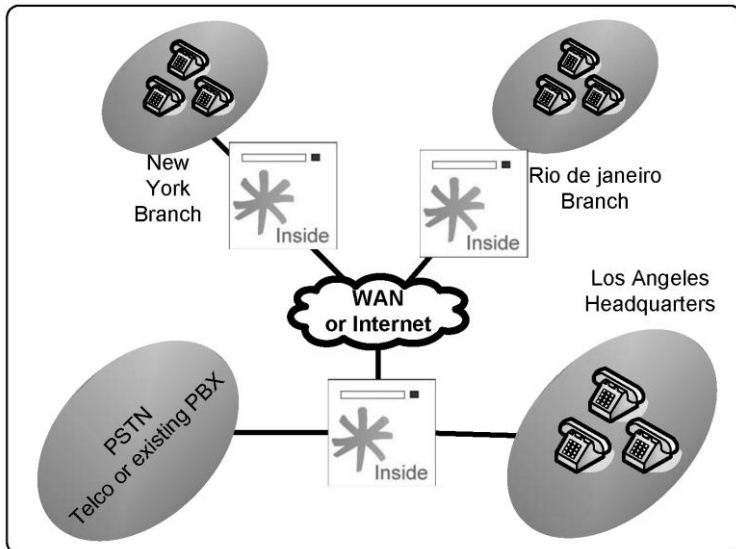
The following image illustrates one of the most commonly used setups. Large companies generally do not want to take significant risk when investing in new technologies and simultaneously wish to preserve their investments in legacy equipment. IP-enabling legacy PBX can be very expensive; thus, connecting an Asterisk PBX using T1/E1 lines

can be a good alternative for cost-conscious customers. Another benefit is the possibility of connecting to a VoIP service provider with better telephony rates.



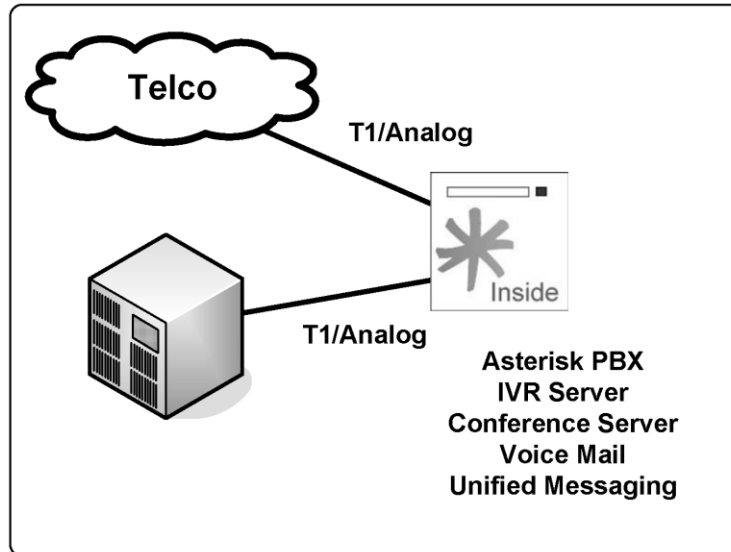
Toll Bypass

A very useful application for VoIP is connecting branch offices over the Internet or a WAN. Using an existing data connection allows you to bypass toll charges incurred in telecommunication connections between headquarters and branch offices.



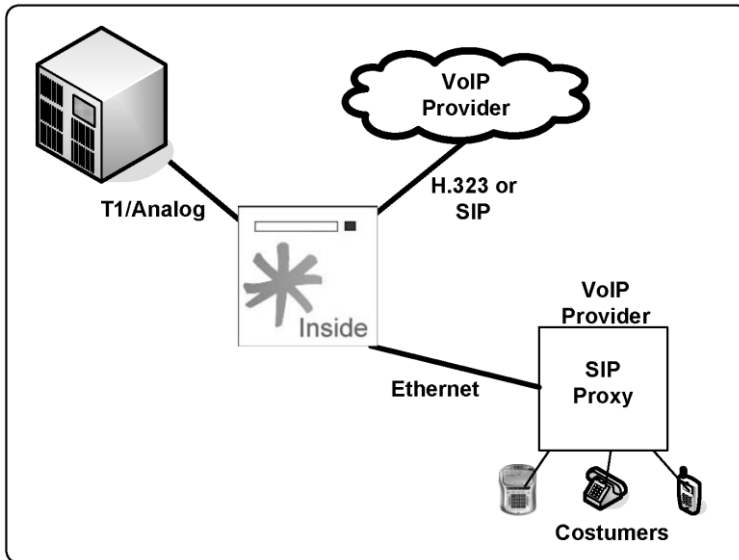
Application Server (IVR, Conference, Voicemail)

Asterisk can be used as an application server for the existing PBX or be directly connected to PSTN. Asterisk offers services such as voicemail, fax reception, call recording, IVR connected to a database, and an audio conferencing server. If you integrate voicemail and fax into an existing e-mail server, you will have a unified messaging system, which is usually an expensive solution. Using Asterisk as an application server provides extreme cost reduction compared to other solutions.



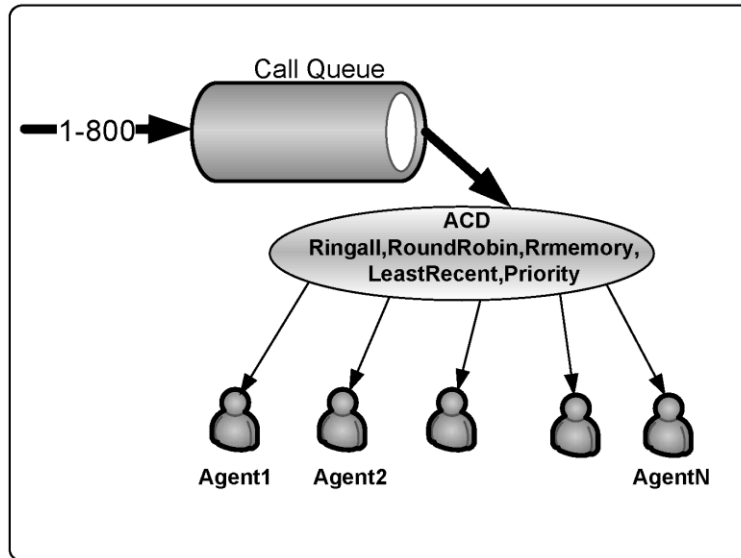
Media Gateway

Most voice-over IP service providers use an SIP proxy to host all registration, location, and authentication of SIP users. They still have to send calls to the PSTN directly or route it through a wholesale call termination provider using an SIP or H.323 voice-over IP connection. Asterisk can act as a back-to-back user agent (B2BUA) or media gateway, replacing very expensive soft switches or media gateways. Compare the price of a four E1/T1 gateway from the main market manufacturers with Asterisk. The Asterisk solution can cost several times less than other solutions and is capable of translating signaling protocols (H.323, SIP, IAX...) and codecs (G.711, G.729...).



Contact Center Platform

A contact center is a very complex solution that combines several technologies, such as automatic call distribution (ACD), interactive voice response (IVR), and call supervision. Basically, three types of contact centers are available: inbound, outbound, and blended. Inbound contact centers are very sophisticated and usually require ACD, IVR, CTI, recording, supervision, and reports. Asterisk has a built-in ACD to queue the calls. IVR can be done using Asterisk Gateway Interface (AGI) or internal mechanisms such as the application background(). Computer telephony integration (CTI) is achieved using Asterisk Manager Interface (AMI); recording and reporting are built in to Asterisk. For an outbound contact center, a predictive or power dialer is one of the main components. Although several dialers are available for the open-source Asterisk, it is not hard to build your own for the platform if you so desire. A blended contact center allows simultaneous inbound and outbound operation, saving money by ensuring better use of the agent's time. It is possible to use Asterisk and its ACD mechanism to implement a blended solution.



Finding information and help

This section will provide some of the main sources of information related to Asterisk.

- Asterisk's official website: <http://www.asterisk.org> Here you can find information about:
 - Support-> <http://www.asterisk.org/support>
 - Knowledge base-> <http://kb.digium.com/>
 - Forum-><http://forums.digium.com/>
 - Bug tracking-><http://bugs.digium.com/>

Additional references: Non-official websites

These sites are not official, but they provide useful content.

- <http://www.voip-info.org>
- <http://www.asteriskguru.com>
- <http://svn.digium.com/svn> (check the doc directory on each branch)

Mailing lists

Mailing lists are quite handy when you have questions. Usually, you will receive answers for your questions. Try to gather as much information as possible before posting to the

list. Nobody will help you if you haven't done your homework. In other words, try at least once to solve the problem by yourself.

- <http://www.asterisk.org/support/mailling-lists>>

Summary

The Asterisk is software licensed according to the GPL that enables an ordinary PC to act as a powerful IP PBX platform. Digium's Mark Spencer created Asterisk in the late 1990s. Digium survives by selling hardware related to Asterisk. Hardware is open-sourced as well and originated in the Zapata project developed by Jim Dixon. The Asterisk architecture has the following main components:

- **CHANNELS:** Analog, digital, or voice-over IP.
- **PROTOCOLS:** Communication protocols, which are responsible for signaling the calls, can be SIP, H323, MGCP, and IAX.
- **CODECS:** Translate digital formats of voice allowing compressions, packet loss concealment, silence suppression, and comfort noise generation. Asterisk does not support silence suppression.
- **APPLICATIONS:** Responsible for the Asterisk PBX functionality. Conference, voicemail, and fax are examples of Asterisk applications.

Asterisk can be used in various scenarios, from a small IP PBX to a sophisticated contact center. You can easily find help at www.asterisk.org

2

How to download and install Asterisk

In the first chapter, we learned a bit about how Asterisk is useful in the telephony environment. In this chapter, we will cover how to download and install Asterisk. Before starting, it is essential to learn how to compile and install it. The compilation process may seem weird for traditional Microsoft™ Windows™ users, but it is fairly common in the Linux™ environment. One can get an optimized code for your hardware when compiling Asterisk, which is what we will do here. Asterisk runs in several operating systems, but we chose to keep things easy and start with only one of them: Linux. We chose Debian as the Linux™ distribution because the dependencies are easy to install and the distribution is stable, with a low footprint. If you want to use another distribution, please change the name of the dependencies accordingly.

Objectives

By the end of this chapter you should be able to:

- Determine the hardware requirements for Asterisk;
- Install Linux with the required dependencies;
- Download a stable version using FTP;
- Compile Asterisk; and
- Learn how to start Asterisk at boot time.

Minimum Hardware Required

Asterisk does not need a lot of hardware to run, however there are some tips to choose the best hardware for your requirements. You should take into consideration the following main factors when choosing your hardware:

- Total number of registered users. Define how many registrations per second you need to support
- Total number of simultaneous calls. Define how many network conversations you need to process in the network adapter and bridge on the Asterisk server
- Which codecs you need to support. High complexity codecs will require a lot of CPU/FPU power in your server, a single iLBC session can require as much as 18MIPS
- Echo cancellation. Echo cancellation may take a lot of CPU/FPU, in some cases you should choose hardware echo cancellation using DSPs in the telephony interface card
- Availability. Use RAID1 or 5 to increase availability. Remember, Asterisk is 24x7 application.
- Redundancy on the telephony interfaces. Xorcom (<http://www.xorcom.com>) and Red-fone(<http://www.red-fone-com>) have very good solutions for this.

The main component for an Asterisk Server is the network adapter. A good server network adapter is recommended. CPU is important when you need to support high complexity codecs such as g.729 and iLBC and echo cancellation. You may choose to use dedicated DSPs, Digium provides a DSP card named TC400B capable to support 120 g729 simultaneous calls.

The best practice is to choose a new, server class, computer from a known manufacturer. To know exactly how many simultaneous calls or how many registered users an specific machine can support, you should test this hardware with a stress test tool such as SIPP (<http://sipp.sourceforge.net>). Some hardware manufacturers such as Xorcom (<http://www.xorcom.com>) publish its results in the website.

Note: Some Asterisk applications, such as meetme and music on hold, requires a clock source. Usually, the clock source is an telephony interface card. If your system does not use a telephony interface card, you will have to load dahdi_dummy to provide a clock source.

Hardware configuration

The Asterisk hardware does not need to be sophisticated. You don't need an expensive video card or numerous peripherals. Some tips about hardware configuration;

- Disable unused USB, serial and parallel ports to avoid the consumption of unnecessary interrupts.
- A robust network interface card is essential.
- Take particular care if you are using telephony interface cards. Some cards use a 3.3 volts PCI bus, and it is not easy to find motherboards for them. In these days, PCI express is more easily found.
- Pay a close attention to the hard disk, PBX used to work in a 24x7 regime while desktops work 8x5. Do not use desktop hardware for a PBX, usually the hard disk fails before the

first year if used intensively. My recommendation is to use a server machine or an appliance designed to run 24x7 applications.

IRQ sharing

Telephony interface cards (e.g., X100P) generate large quantities of interruptions. Serving these interruptions requires processor time. The drivers can't do this processing if you have another device using the same interruption. In a single CPU system, you should avoid IRQ sharing between devices. We recommend the use of dedicated hardware to run Asterisk. Don't forget to disable any foreign or unnecessary hardware. Some hardware can be disabled in the motherboard bios setup. Once you have started your computer, see your assigned interrupts in `/proc/interrupts`.

#cat /proc/interrupts

```
CPU0
0: 41353058 XT-PIC timer
1: 1988 XT-PIC keyboard
2: 0 XT-PIC cascade
3: 413437739 XT-PIC wctdm <-- TDM400
4: 5721494 XT-PIC eth0
7: 413453581 XT-PIC wcfxo <-- X100P
8: 1 XT-PIC rtc
9: 413445182 XT-PIC wcfxo <-- X100P
12: 0 XT-PIC PS/2 Mouse
14: 179578 XT-PIC ide0
15: 3 XT-PIC ide1
NMI: 0
ERR: 0
```

Here you can see three Digium cards, each in their own IRQ. If this is the case in your system, go ahead and install the hardware drivers. If this is not the case, go back and try something else to avoid IRQ sharing.

Choosing a Linux distribution

Asterisk was initially developed to run on Linux. However, it can also run on BSD Unix or Mac OS X. If you are new to Asterisk, try using Linux first since it is much easier. Several Linux distributions were successfully tested with Asterisk (e.g., Fedora, Redhat, SuSe, Debian, and Gentoo); choose one for your system. You can download the Debian distribution from the address below:

<http://www.us.debian.org/CD/netinst/#netinst-stable>.

Required dependencies

The following dependencies are required to compile Asterisk.

- bison
- libssl-dev

- openssl
- libasound2-dev
- libc6-dev
- libnewt-dev
- zlib1g-dev
- gcc
- g++
- make
- libncurses5-dev
- doxygen
- libxml2-dev

Required by DAHDI

- kernel sources

Caution: DAHDI packages are necessary to compile some Asterisk applications like `meetme()`. If you have compiled Asterisk before DAHDI, you will have to recompile it again to include the application `meetme()` as well as certain others.

Required by Xorcom Astribank

- libusb-dev
- fxload

Installing Linux for Asterisk

Install your Linux as usual, without a graphical user interface. Install and configure the email server as well. We will need the email server (exim4) to send voicemail notifications later in this book.

Caution: This installation will format your PC. All your disk data will be erased. Please make sure to back up all data before starting.

Step 1: Put the CD in the CD-ROM drive and boot your PC. Most questions are very simple to answer.

Preparing Linux for Asterisk

Immediately after installing Asterisk, we will install the packages required for the subsequent compilation of Asterisk and DAHDI drivers. First, we will indicate to Debian where the packages will be downloaded from. This is done by using the apt-setup utility.

Step 1: Login as root.

Step 2: Install the kernel headers.

```
apt-get install linux-headers-`uname -r`  
ln -s /usr/src/kernel-headers-`uname -r` /usr/src/linux
```

Step 3: Install the required packages.

```
apt-get install bison openssl libssl-dev libusb-dev fxload libasound2-dev libc6-  
dev libnewt-dev libncurses5-dev zlib1g-dev gcc g++ make doxygen libxml2-dev
```

Which version to choose

As a rule of thumb, you should use the version with the required features. Versions 1.2 and 1.4 are more stable than the newest 1.6 while the newer versions include the new features, meaning 1.2 and 1.4 are feature frozen. The Asterisk team has changed the version system for 1.6. Now, instead of having major versions each year, they are releasing major and minor versions. The newest version is 1.6.2; it is undergoing just bug fixes, too. All new development is integrated in the trunk. With 1.6 you will have at least three versions maintained simultaneously, which allows you an extended period to upgrade from one version to another. Recently they announced the change back to the old versioning system.

All examples in this book were created or converted to Asterisk 1.6.2, but most should work in 1.4.

Obtaining and compiling Asterisk

The next step is the installation of Asterisk. To obtain the sources, you should download them from www.asterisk.org. We will use the `wget` utility to download them. Create a directory `/usr/src` to receive the files. You should consult www.asterisk.org to verify which version is the newest.

For Asterisk 1.4

Download the source files from the Asterisk repository. Please, check for a newer version.

```
cd /usr/src  
wget http://downloads.asterisk.org/pub/telephony/asterisk/releases/asterisk-1.4.29.1.tar.gz  
wget http://downloads.asterisk.org/pub/telephony/libpri/releases/libpri-1.4.10.1.tar.gz  
wget http://downloads.asterisk.org/pub/telephony/asterisk/releases/asterisk-addons-1.4.10.tar.gz
```

For Asterisk 1.6

Download the source files from the Asterisk repository. Please, check for a newer version.

```
cd /usr/src
wget http://downloads.asterisk.org/pub/telephony/asterisk/releases/asterisk-1.6.2.5.tar.gz
wget http://downloads.asterisk.org/pub/telephony/libpri/releases/libpri-1.4.10.2.tar.gz
wget http://downloads.asterisk.org/pub/telephony/asterisk/releases/asterisk-addons-1.6.2.0.tar.gz
```

DAHDI

The same version of DAHDI is used for both versions.

```
wget http://downloads.asterisk.org/pub/telephony/dahdi-linux/releases/dahdi-linux-2.2.1.tar.gz
wget http://downloads.asterisk.org/pub/telephony/dahdi-tools/releases/dahdi-tools-2.2.1.tar.gz
```

Uncompress the files using:

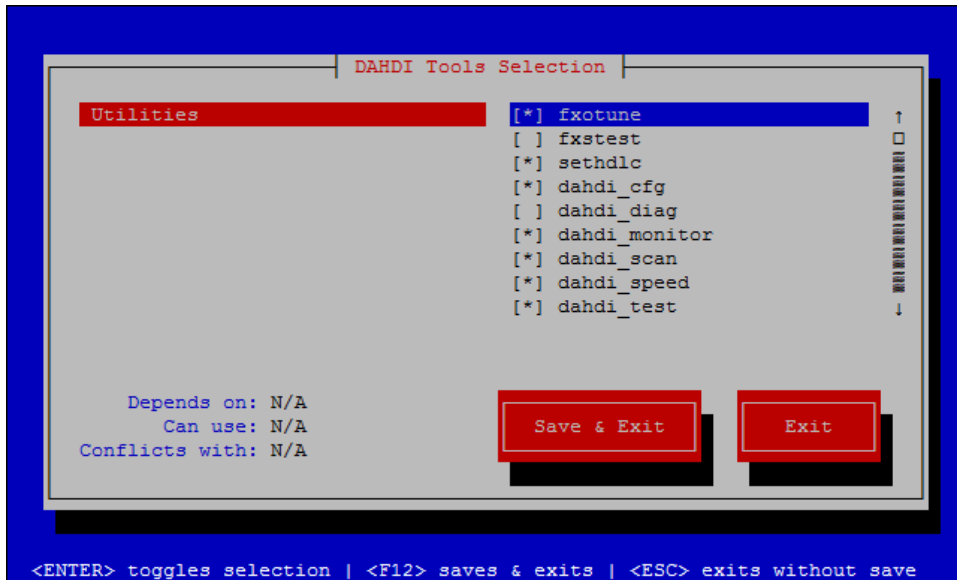
```
tar xzvf file.tar.gz
```

Compiling DAHDI drivers

You will need to compile the DAHDI modules. The commands `./configure` and `make menuselect` were added in version 1.4. The latter enables you to select which utilities and modules to build. The following commands will do this:

```
cd /usr/src/dahdi-linux-2.2.1
make
make install
cd /usr/src/dahdi-tools-2.2.1
./configure
make menuselect #(optional, you may select some options)
make
make install
make config #(optional, it installs the init scripts)
```

Use `make menuselect` to install only the necessary modules. This is the `make menuselect` screenshot.



Just after executing `make config`, the init scripts will be installed, and the following screen will be shown.

```
install -D dahdi.init /etc/init.d/dahdi
```

```
/usr/bin/install -c -D -m 644 init.conf.sample /etc/dahdi/init.conf
```

```
/usr/bin/install -c -D -m 644 modules.sample /etc/dahdi/modules
```

```
/usr/bin/install -c -D -m 644 blacklist.sample /etc/modprobe.d/dahdi.blacklist
```

```
/usr/sbin/update-rc.d dahdi defaults 15 30
```

Adding system startup for /etc/init.d/dahdi ...

```
/etc/rc0.d/K30dahdi -> ../init.d/dahdi
```

```
/etc/rc1.d/K30dahdi -> ../init.d/dahdi
```

```
/etc/rc6.d/K30dahdi -> ../init.d/dahdi
```

```
/etc/rc2.d/S15dahdi -> ../init.d/dahdi
```

```
/etc/rc3.d/S15dahdi -> ../init.d/dahdi
```

```
/etc/rc4.d/S15dahdi -> ../init.d/dahdi
```

```
/etc/rc5.d/S15dahdi -> ../init.d/dahdi
```

DAHDI has been configured.

If you have any DAHDI hardware it is now recommended you edit `/etc/dahdi/modules` in order to load support for only the DAHDI hardware installed in this system. By default

support for all DAHDI hardware is loaded at DAHDI start.

I think that the DAHDI hardware you have on your system is:

```
usb:004/002      xpp_usb-   e4e4:1150 Astribank-multi no-firmware
```

This screen (above) asks you to change the file `/etc/dahdi/modules` to load only the required drivers for your specific configuration and show the detected hardware. Edit the file `/etc/dahdi/modules` and load only the required hardware. In my case, I was using a test machine with a Xorcom Astribank 6FXS and 2FXO. The file is shown below.

```
# Contains the list of modules to be loaded / unloaded by /etc/init.d/dahdi.
#
# NOTE: Please add/edit /etc/modprobe.d/dahdi or /etc/modprobe.conf if you
#       would like to add any module parameters.
#
# Format of this file: list of modules, each in its own line.
# Anything after a '#' is ignore, likewise trailing and leading
# whitespaces and empty lines.

# Digium TE205P/TE207P/TE210P/TE212P: PCI dual-port T1/E1/J1
# Digium TE405P/TE407P/TE410P/TE412P: PCI quad-port T1/E1/J1
# Digium TE220: PCI-Express dual-port T1/E1/J1
# Digium TE420: PCI-Express quad-port T1/E1/J1
#wct4xxp

# Digium TE120P: PCI single-port T1/E1/J1
# Digium TE121: PCI-Express single-port T1/E1/J1
# Digium TE122: PCI single-port T1/E1/J1
#wcte12xp

# Digium T100P: PCI single-port T1
# Digium E100P: PCI single-port E1
#wct1xxp

# Digium TE110P: PCI single-port T1/E1/J1
#wcte11xp

# Digium TDM2400P/AEX2400: up to 24 analog ports
# Digium TDM800P/AEX800: up to 8 analog ports
# Digium TDM410P/AEX410: up to 4 analog ports
#wctdm24xxp

# X100P - Single port FXO interface
# X101P - Single port FXO interface
#wcfxo

# Digium TDM400P: up to 4 analog ports
#wctdm
```

```
# Xorcom Astribank Devices
xpp_usb
```

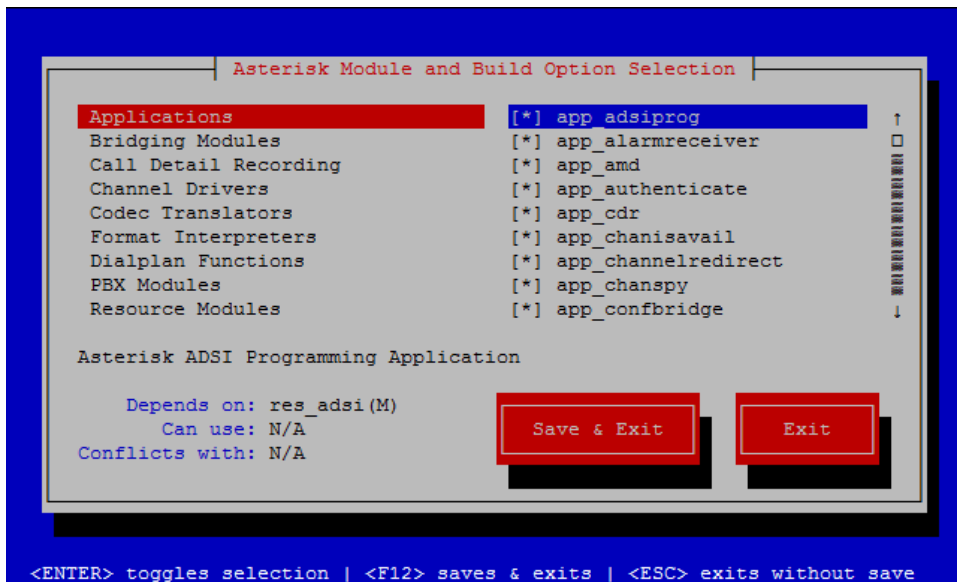
Re-initialize your computer and verify the correct loading of the drivers.

Compiling Asterisk

If you have previously compiled software, compiling Asterisk will be an easy task. Run the following commands to compile and install Asterisk. Remember, you can choose which applications and modules to build using **make menuselect**.

```
cd /usr/src/libpri-1.4.10.2
make
make install
cd /usr/src/asterisk-1.6.2.5
./configure
make menuselect
make
make install
make samples ;use to create sample configuration files
make config ;to start asterisk at boot time
```

Use **make menuselect** to install only the necessary modules.



Starting and stopping Asterisk

With this minimal configuration, it's possible to start Asterisk successfully.

```
/usr/sbin/asterisk -vvvgc
```

Use the CLI command **stop now** to shutdown Asterisk.

```
CLI>stop now
```

Asterisk runtime options

The Asterisk starting process is very simple. If Asterisk is run without any parameters, it is launched as a daemon.

```
/sbin/asterisk
```

You can access the Asterisk console by executing the following command. Please note that more than one console process can be run at the same time.

```
/sbin/asterisk -r
```

Available runtime options for Asterisk

You can show the available runtime options using `asterisk -h`

```
sipast:/usr/src/asterisk-1.6# asterisk -h
```

Asterisk 1.6.1.1, Copyright (C) 1999 – 2008, Digium, Inc. and others.

Usage: asterisk [OPTIONS]

Valid Options:

- V Display version number and exit
- C <configfile> Use an alternate configuration file
- G <group> Run as a group other than the caller
- U <user> Run as a user other than the caller
- c Provide console CLI
- d Enable extra debugging
- f Do not fork
- F Always fork
- g Dump core in case of a crash
- h This help screen
- i Initialize crypto keys at startup
- l Enable internal timing if DAHDI timer is available
- L <load> Limit the maximum load average before rejecting new calls
- M <value> Limit the maximum number of calls to the specified value
- m Mute debugging and console output on the console
- n Disable console colorization
- p Run as pseudo-realtime thread
- q Quiet mode (suppress output)
- r Connect to Asterisk on this machine
- R Same as -r, except attempt to reconnect if disconnected
- t Record soundfiles in /var/tmp and move them where they belong after they are done
- T Display the time in [Mmm dd hh:mm:ss] format for each line of output to the CLI
- v Increase verbosity (multiple v's = more verbose)

- x <cmd> Execute command <cmd> (only valid with -r)
- s <socket> Connect to Asterisk via socket <socket> (only valid with -r)

Installation directories

Asterisk is installed on several directories, which can be modified in the [asterisk.conf](#) file.

asterisk.conf

```
[directories](!) ; remove the (!) to enable this
astetcdir => /etc/asterisk
astmoddir => /usr/lib/asterisk/modules
astvarlibdir => /var/lib/asterisk
astdbdir => /var/lib/asterisk
astkeydir => /var/lib/asterisk
astdatadir => /var/lib/asterisk
astagidir => /var/lib/asterisk/agi-bin
astspooldir => /var/spool/asterisk
astrundir => /var/run/asterisk
astlogdir => /var/log/asterisk

[options]
;verbose = 3
;debug = 3
;alwaysfork = yes ; same as -F at startup
;nofork = yes ; same as -f at startup
;quiet = yes ; same as -q at startup
;timestamp = yes ; same as -T at startup
;execincludes = yes ; support #exec in config files
;console = yes ; Run as console (same as -c at startup)
;highpriority = yes ; Run realtime priority (same as -p at startup)
;initcrypto = yes ; Initialize crypto keys (same as -i at startup)
;nocolor = yes ; Disable console colors
;dontwarn = yes ; Disable some warnings
;dumpcore = yes ; Dump core on crash (same as -g at startup)
;languageprefix = yes ; Use the new sound prefix path syntax
;internal_timing = yes
;systemname = my_system_name ; prefix uniqueid with a system name for global
uniqueness issues
;autosystemname = yes ; automatically set systemname to hostname - uses
'localhost' on failure, or systemname if set
;maxcalls = 10 ; Maximum amount of calls allowed
;maxload = 0.9 ; Asterisk stops accepting new calls if the load average exceed
this limit
;maxfiles = 1000 ; Maximum amount of openfiles
;minmemfree = 1 ; in MBs, Asterisk stops accepting new calls if the amount of
free memory falls below this watermark
;cache_record_files = yes ; Cache recorded sound files to another directory
during recording
```

```

;record_cache_dir = /tmp ; Specify cache directory (used in conjunction with
cache_record_files)
;transmit_silence_during_record = yes ; Transmit SLINEAR silence while a
channel is being recorded
;transmit_silence = yes ; Transmit SLINEAR silence while a channel is being
recorded or DTMF is being generated
;transcode_via_sln = yes ; Build transcode paths via SLINEAR, instead of
directly
;runuser = asterisk ; The user to run as
;rungroup = asterisk ; The group to run as
;lightbackground = yes ; If your terminal is set for a light-colored background
documentation_language = en_US ; Set the Language you want Documentation
displayed in. Value is in the same format as locale names
;hideconnect = yes ; Hide messages displayed when a remote console connects and
disconnects

; Changing the following lines may compromise your security.
;[files]
;astctlpermissions = 0660
;astctlowner = root
;astctlgroup = apache
;astctl = asterisk.ctl

[compat]
pbx_realtime=1.6
res_agi=1.6
app_set=1.6

```

Log files and log rotation

Asterisk PBX logs its messages on the `/var/log/asterisk` directory. The file that controls the logs is the `logger.conf`.

```

; Logging Configuration
;
; In this file, you configure logging to files or to
; the syslog system.
;
; "logger reload" at the CLI will reload configuration
; of the logging system.

[general]
; Customize the display of debug message time stamps
; this example is the ISO 8601 date format (yyyy-mm-dd HH:MM:SS)
; see strftime(3) Linux manual for format specifiers
;dateformat=%F %T
;
; This appends the hostname to the name of the log files.
;appendhostname = yes
;

```



```
; This determines whether or not we log queue events to a file
; (defaults to yes).
;queue_log = no
;
; This determines whether or not we log generic events to a file
; (defaults to yes).
;event_log = no
;
;
; For each file, specify what to log.
;
; For console logging, you set options at start of
; Asterisk with -v for verbose and -d for debug
; See 'asterisk -h' for more information.
;
; Directory for log files is configures in asterisk.conf
; option astlogdir
;
[logfiles]
;
; Format is "filename" and then "levels" of debugging to be included:
;   debug
;   notice
;   warning
;   error
;   verbose
;   dtmf
;
; Special filename "console" represents the system console
;
; We highly recommend that you DO NOT turn on debug mode if you are simply
; running a production system. Debug mode turns on a LOT of extra messages,
; most of which you are unlikely to understand without an understanding of
; the underlying code. Do NOT report debug messages as code issues, unless
; you have a specific issue that you are attempting to debug. They are
; messages for just that -- debugging -- and do not rise to the level of
; something that merit your attention as an Asterisk administrator. Debug
; messages are also very verbose and can and do fill up logfiles quickly;
; this is another reason not to have debug mode on a production system unless
; you are in the process of debugging a specific issue.
;
;debug => debug
console => notice,warning,error
;console => notice,warning,error,debug
messages => notice,warning,error
;full => notice,warning,error,debug,verbose

;syslog keyword : This special keyword logs to syslog facility
```

```
;
;syslog.local0 => notice,warning,error
;
```

Some console commands are associated with the logger process.

```
CLI> logger list channels
Channel                                     Type      Status    Configuration
-----
/var/log/asterisk/messages                 File      Enabled   - Warning Notice Error
                                           Console  Enabled   - Warning Notice Error

CLI> logger rotate
== Parsing '/etc/asterisk/logger.conf': Found
Asterisk Event Logger restarted
Asterisk Queue Logger restarted
```

You can control the log rotation using the **logrotate** daemon. Edit the file **/etc/logrotate.d** and include the content below to start rotating the log files.

```
/var/log/asterisk/messages /var/log/asterisk/*log {
    missingok
    rotate 5
    weekly
    create 0640 asterisk asterisk
    postrotate
        /usr/sbin/asterisk -rx 'logger reload'
    endscript
}
```

More information about **logrotate** can be obtained using:

```
#man logrotate
```

Starting Asterisk with a non-root user

It is safer to execute Asterisk with a non-root user. In case of a security failure or a buffer overflow attack, running Asterisk within an environment with fewer privileges to the user limits an intruder's possible actions.

To change Asterisk's running user:

Step 1: Edit the file: vi /etc/init.d/asterisk

Step 2: Uncomment the following lines:

```
AST_USER="asterisk"
AST_GROUP="asterisk"
```

Step 3: To change user rights in Asterisk folders, type:

```
cd /
chown --recursive asterisk:asterisk /etc/asterisk
chmod --recursive u=rwX,g=rX,o= /etc/asterisk
chown --recursive asterisk:asterisk /var/lib/asterisk
chown --recursive asterisk:asterisk /var/log/asterisk
chown --recursive asterisk:asterisk /var/run/asterisk
chown --recursive asterisk:asterisk /var/spool/asterisk
```

```
chown --recursive asterisk:asterisk /dev/dahdi
chmod --recursive u=rwX,g=rX,o= /var/lib/asterisk
chmod --recursive u=rwX,g=rX,o= /var/log/asterisk
chmod --recursive u=rwX,g=rX,o= /var/run/asterisk
chmod --recursive u=rwX,g=rX,o= /var/spool/asterisk
chmod --recursive u=rwX,g=rX,o= /dev/dahdi
```

Step 4: Test changes using `/etc/init.d/asterisk`

Uninstalling Asterisk

To uninstall Asterisk, use:

```
make uninstall
```

To uninstall Asterisk and all configuration files, use:

```
make uninstall-all
```

Asterisk installation notes

This section will provide some advice about issues to address before installing Asterisk.

Production Systems

If Asterisk is installed in a production environment, you should pay attention to the system design. A server has to be optimized in such a way that telephony systems have priority over other system processes. Asterisk should not run together with processor-intensive software such as X-Windows. If you need to run CPU-intensive processes (e.g., a huge database), use a separate server. Generally speaking, Asterisk is susceptible to hardware performance variations. Thus, try using Asterisk in a hardware environment that does not require more than 40% of CPU utilization.

Network Tips

If you plan to use IP phones, it is important that you pay attention to your network. Voice protocols are very good and resistant to latency and even jitters; however, if you use a poorly configured local area network, voice quality will suffer. It is only possible to guarantee good voice quality using quality of service (QoS) in switches and routers. Voice in a local area network tends to be good, but even in a LAN environment, if you have 10 Mbps hubs with too many collisions, you will end up having a distorted or crappy voice. Follow these recommendations to ensure the best possible voice quality:

- Use end-to-end QoS if possible or economically feasible. With end-to-end QoS, the voice quality is perfect. No excuses!
- Avoid using 10/100 Mbps hubs for voice in a production environment. Collisions can impose jitters on the network. Full duplex 10/100 Mbps are preferred because no collisions occur.
- Use VLANs to separate unnecessary broadcasts of the voice network. You don't want a virus destroying your voice network with ARP broadcasts.

- Educate users about expectations in a voice network. Without QoS, don't state that the voice will be perfect as in most cases it won't be. A quality of voice similar to a mobile phone will most often be achieved. Use quality phones as problems with firmware and hardware design are common.

Summary

In this chapter, you have learned about the minimum hardware requirements as well as how to download, install, and compile Asterisk. Asterisk should be executed with a non-root user for security reasons. You should check your network environment before starting the production environment.

Quiz

1. What's the minimal Asterisk hardware configuration?
2. Telephony interface cards for Asterisk usually have some Digital Signal Processors (DSPs) built in and do not need a lot of CPU resources from the PC.
 - A. True
 - B. False
3. If you want perfect voice quality, you need to implement end-to-end quality of service (QoS).
 - A. True
 - B. False
4. You should always choose the latest Asterisk version as it is the most stable version.
 - A. True
 - B. False
5. List the necessary packages for Asterisk and the DAHDI compilation.
6. If you don't have a TDM interface card, you will end up needing a clock source for synchronization. The `dahdi_dummy` driver fills this role by using the USB as a clock source (Kernel 2.4). This is necessary because some applications like _____ and _____ require a time reference.
7. When you install Asterisk, it's better to leave desktop interfaces such as GNOME or KDE out. Graphical user interfaces take up numerous CPU cycles.
 - A. True

B. False

8. Asterisk configuration files are located in the _____ directory.
9. To install Asterisk sample files, you need to type the following command:
10. Why is it important to start Asterisk with a non-root user?

3

Building a simple PBX

In this chapter, you will learn how to perform a basic Asterisk PBX configuration. The main objective here is to see the PBX running for the first time, be able to dial between extensions, dial a message being played, and dial to a single analog or SIP trunk. The idea behind this chapter is to ensure that your Asterisk is up and running as soon as possible. After completing the work in this chapter, you will have sufficient background to prepare for subsequent chapters, where we will delve more deeply into configuration details.

Objectives

By the end of this chapter, you should be able to:

- Understand and edit configuration files;
- Install soft-phones based on SIP;
- Install and configure a SIP trunk;
- Install and configure an analog connection;
- Dial between extensions;
- Dial between phones and external destinations; and
- Configure an auto attendant.

Understanding the configuration files

Asterisk is controlled by text configuration files located in `/etc/asterisk`. The file format is similar to the Windows “.ini” files. A semicolon is used as a remark character, the signs “=” and “=>” are equivalent, and spaces are ignored.

```
;
; The first line without a comment should be the session title.
;
[Session]
key = value; Variable designation
[Session 2]
key => value; Object declaration
```

Asterisk interprets “=” and “=>” in the same way. Differences in syntax are used to distinguish between objects and variables. Use “=” when you want to declare a variable and “=>” to designate an object. The syntax is the same between all files, but three types of grammar are used, as discussed below.

Grammars

Grammar	Object is created:	Conf. File	Example
Simple Group	All in the same line	extensions.conf	exten=> 4000,1,Dial(SIP/4000)
Option Inheritance	Options are defined first, object inherit the options	chan_dahdi.conf	[channels] context=default signalling=fxs_ks group=1 channel => 1
Complex Entity	Each entity receives a context	sip.conf, iax.conf	[cisco] type=friend secret=mysecret host=10.1.30.50 context=trusted [xlite] type=friend secret=xlite host=dynamic

Simple Group

The simple group format used in **extensions.conf**, **meetme.conf**, and **voicemail.conf** is the most basic grammar. Each object is declared with options in the same line.

Example:

```
[Session]
object 1 => op1,op2,op3
object 2=> op1b,op2b,op3b
```

In this example, object 1 is created with options op1, op2, and op3 while object 2 is created with options op1, op2, and op3.

Object options inheritance grammar

This format is used by the files **chan_dahdi.conf** and **agents.conf**, where numerous options are available, and most interfaces and objects share the same options. Typically, one or more sections have objects and channels declarations. Options to the object are **declared above** the object and can be changed to another object. Although this concept is hard to understand, it is very easy to use.

Example:

```
[Session]
op1 = bas
op2 = adv
object=>1
```

```
op1 = int
object => 2
```

The first two lines configure the value of the options `op1` and `op2` to “bas” and “adv”, respectively. When object 1 is instanced, it is created using option 1 as “bas” and option 2 as “adv”. After defining object 1, we change option 1 to “int”. Next, we create object 2 with option 1 as “int” and option 2 as “adv”.

Complex entity object

This format is used by `iax.conf`, `sip.conf`, and other configuration files in which numerous entities with many options exist. Typically, this format does not share a large volume of common configurations. Each entity receives a context. Sometimes reserved contexts exist, like `[general]` for global configurations. Options are declared in the context declarations.

Example:

```
[entity1]
op1=value1
op2=value2
[entity2]
op1=value3
op2=value4
```

The entity `[entity1]` has values “value1” and “value2” for options `op1` and `op2`, respectively. The entity `[entity2]` has values “value3” and “value4” for options `op1` and `op2`.

Options to build a LAB for Asterisk

To configure a PBX, you will need some basic hardware. It is not hard or expensive, but there are some options to be considered. All you will need are two phones and a connection to the public network. Some options and combinations are possible when creating your lab, which we will discuss below.

Option 1: Complete LAB

With the complete LAB, it is possible to test all the scenarios available and compare solutions such as ATA, IP-phones, and soft-phones. You can also learn about analog and SIP trunks.

Qty.	Description
1	SIP Analog Telephone Adapter
2	IP Phone
3	Dedicated Server for the Asterisk Server
4	Workstation with the soft-phone
5	Analog Interface Card with at least two interfaces: 1 FXO and 1 FXS
6	VoIP provider Account

Option 2: Economy LAB

With the economy LAB, we simplify it a bit. We use the ATA, which is usually less expensive than the IP-phone, and a single FXO card, which is really inexpensive. We won't be able to use analog phones connected directly to the server, but this does not commonly occur in practice.

Qty.	Description
1	SIP Analog Telephone Adapter
2	Dedicated Server for Asterisk
3	Workstation for the soft-phone
4	Analog Interface Card with 1 FXO
5	Account in a VoIP provider

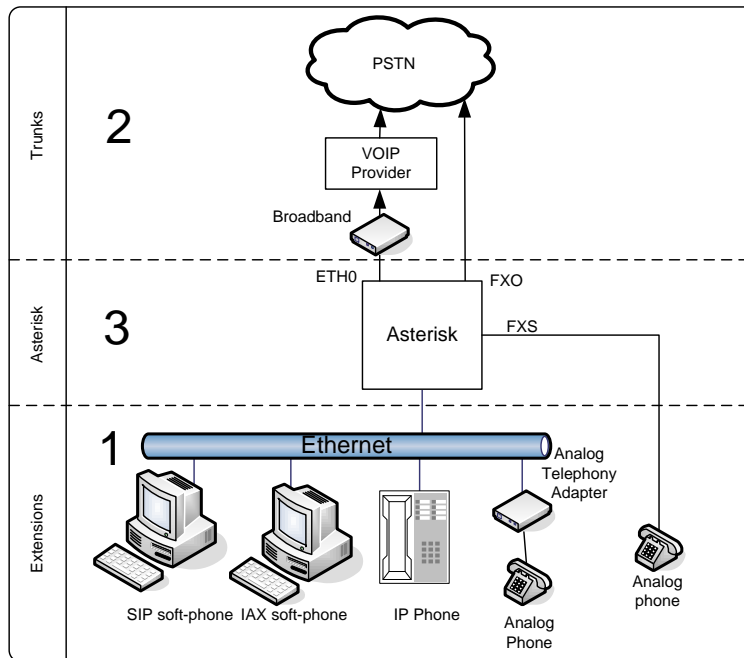
Option 3: Super economy lab

The third LAB uses a virtualized server in the student's own notebook. The problem with this model is the conflicts generated by the UDP port. Sometimes both the Asterisk server and the soft-phone try to access the same port, preventing Asterisk from binding the address port. Another issue is the quality of the calls; virtual environments are not indicated for real-time applications such as Asterisk. Use a free soft-phone for the server and workstation and a trunk connection to a SIP provider.

Qty.	Description
1	Laptop with 1 GB memory and a soft-phone
3	Virtual Machine (VMWare, Xen, or other) to install Asterisk and a soft-phone
4	Account in a VoIP provider

Installation Sequence

To help you understand the installation sequence, we outlined the sequence of steps necessary to install and configure Asterisk.



1. Extensions configuration
 - a. SIP extensions (ATA, Soft-phone, IP Phone)
 - b. IAX extensions
 - c. FXS extensions
2. Trunk configuration
 - a. Configuration of a SIP trunk
 - b. Configuration of a FXO trunk
3. Building a basic dial plan
 - a. Dialing between extensions
 - b. Dialing external destinations
 - c. Receiving a call from in the operator extension
 - d. Receiving a call in an auto-attendant

Configuration of the extensions

The extensions are SIP, IAX, or analog phones connected to an FXS port. To configure an extension, you should edit the configuration file related to the channel ([sip.conf](#), [iax.conf](#), [chan_dahdi.conf](#))

SIP extensions

Let's configure the SIP phones. The idea is to configure a simple PBX. (Subsequent chapters will provide an entire SIP session with all the details.) SIP is configured in the `/etc/asterisk/sip.conf` directory and has all the parameters related to SIP phones and VoIP providers. SIP clients have to be configured before you can make and receive calls.

The section `[general]` includes some parameters to be configured; it is the first section we will configure. The main options are:

- `allow/disallow`: Defines which codecs are going to be used.
- `bindaddr`: Address to be bound to the Asterisk SIP listener. If you set it up as `0.0.0.0` (default), it will bind to all interfaces.
- `context`: Sets the default context for all clients unless it is changed in the client section. We used `dummy` for security reasons. Unauthenticated users get into this context when the option `allowguest` is set to `yes`.
- `bindport`: SIP UDP port to listen.
- `maxexpiry`: Maximum time to register (seconds).
- `defaultexpiry`: Default time to register (seconds).
- `register`: Registers Asterisk to another host.
- `allowguest`: Usually set to `no` to avoid non-authenticated users in the context of the `[general]` section.
- `alwaysauthreject`: When an incoming `INVITE` or `REGISTER` is received, always reject with an identical response (valid username, invalid password). This avoids username guessing.

Example:

```
[general]
bindport = 5060
bindaddr = 10.1.30.45
context = dummy
disallow = all
allow = ulaw
maxexpiry = 120
defaultexpiry = 80
allowguest=no
alwaysauthreject=yes
```

SIP clients

After completing the general sections, it is time to set up the SIP clients. I would once again like to remind the reader that we will have an entire SIP chapter later in the book. For now, let's concentrate on the basics and leave the details for later.

- **[name]:** When a SIP device connects to Asterisk, it uses the username part of the SIP URI to find the peer/user.
- **type:** Configures the connection class. Options are peer, user, and friend.
 - peer: Asterisk sends calls to a peer.
 - user: Asterisk receives calls from a user.
 - friend: Both occur at the same time.
- **host:** IP address or host name. The most common option is “dynamic”, which is used when the host registers to Asterisk.
- **secret:** Password to authenticate peers and users.

Warning: Use **strong passwords**, with at least 8 characters, alphanumeric and numeric characters, and at least one symbol. Reports of hacked servers have appeared in the mailing lists, and brute force password crackers for SIP are easily available for script kiddies. Toll fraud costs thousands of dollars for consumers and providers.

Example:

```
[6000]
type=friend
secret=#MySecret1#7
host=10.1.30.50
context=from-internal
```

```
[6001]
type=friend
secret=Mys3cr3t#
host=dynamic
context=from-internal
defaultip=10.1.30.17
```

IAX Extensions

You may also create IAX extensions. This protocol is native to the Asterisk, and we will have an entire section devoted to it later in this book. For now, let’s create a few extensions using the protocol.

The file is very similar to `sip.conf`. As the first section to be configured, the section `[general]` has certain parameters to be configured. The main options are:

- `allow/disallow`: Defines which codecs are going to be used.
- `bindaddr`: Address to be bound to Asterisk SIP listener. If you set it up as 0.0.0.0 (default), it will bind to all interfaces.

- **context:** Sets the default context for all clients unless changed in the client section. We used dummy for security reasons. Unauthenticated users get into this context when the option allowguest is set to yes.
- **bindport:** SIP UDP port to listen.
- **delayrejects:** When set to yes, delays sending the authentication rejects, which improves the security against brute force password attacks.
- **bandwidth:** When set to high, it allows the selection of high bandwidth codecs, such as the g711 in their variants **u1aw** and **a1aw**.

The following is a sample of the **[general]** section of the file **iax.conf**.

```
[general]
bindport = 4569
bindaddr = 10.1.30.45 ;(use your IP)
context = dummy
delayreject=yes
bandwidth=high
disallow = all
allow = u1aw
```

IAX Clients

After finishing the general sections, it is time to set up the IAX clients.

- **[name]:** When a SIP device connects to Asterisk, it uses the username part of the SIP URI to find the peer/user.
- **type:** Configures the connection class. Options are peer, user, and friend.
 - peer: Asterisk sends calls to a peer.
 - user: Asterisk receives calls from a user.
 - friend: Both occur at the same time.
- **host:** IP address or host name. The most common option is **dynamic**, which is used when the host registers to Asterisk.
- **secret:** Password to authenticate peers and users.

Warning: Use strong passwords with at least 8 characters, alphanumeric and numeric characters, and at least one symbol. Reports of hacked servers have appeared in the mailing lists, and brute force password crackers for SIP md5 hashes are available for script kiddies. Toll fraud costs thousands of dollars for consumers and providers.

Example:

```
[guest]
type=user
context=dummy
callerid="Guest IAX User"
```

```
[6003]
context=from-internal
type=friend
secret=#sup3rs3cr3t#
host=dynamic
context=from-internal
```

```
[6004]
context=from-internal
type=friend
secret=#s3cr3ts3cr3t#
host=dynamic
context=from-internal
```

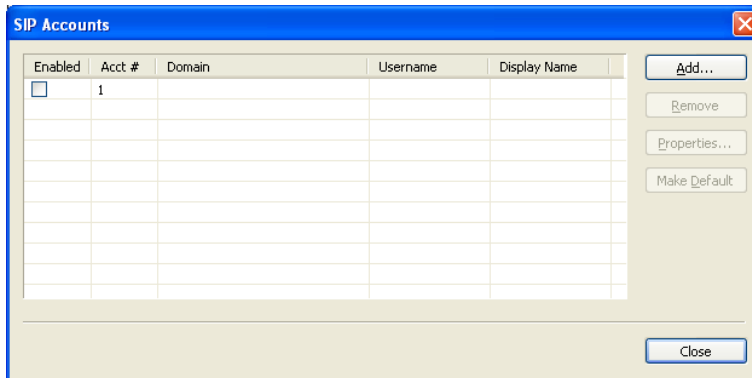
Configuring the SIP devices

After defining the phones in the Asterisk configuration file, it is time to configure the phone itself. In this example, we will show how to configure a free soft-phone—in this case, xlite from Counterpath (<http://www.counterpath.com>). Check your device's manual to understand the parameters of your phone.

Step 1: Configure the phone to use the extension 6000

Execute the installation program.

After the execution, click the mouse's right button and choose SIP Account Settings.



Select the button Add...

Fill in the required information.

Properties of Account 1

Account Voicemail Topology Presence Advanced

User Details

Display Name: 6000

User name: 6000

Password: ••••

Authorization user name: 6000

Domain: 192.168.110.134

Domain Proxy

Register with domain and receive incoming calls

Send outbound via:

domain

proxy Address

Dialing plan: #1\a\a.T;match=1;prestrip=2;

OK Cancel Apply

Display Name: 6000
User Name: 6000
Password: =#MySecret1#7
Authorization User Name: 6000
Domain: ip_of_your_server

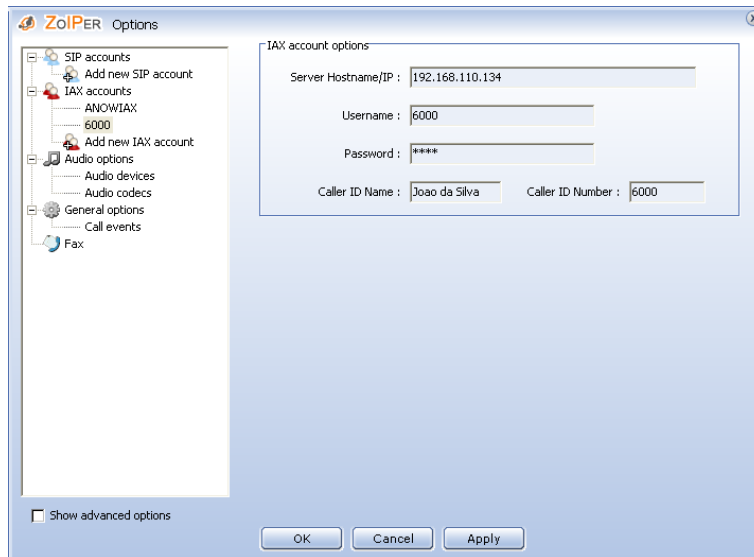
Confirm that your phone is registered using the console command `sip show peers`. Repeat the configuration for the phone 6001.

Configuring the IAX devices

In this example, we are going to use the free soft-phone Zoiper, which you can download from www.zoiper.com.

1. Download and install the Zoiper Free.
2. Click with the right button to access options.

3. Select new IAX account.
4. Insert the related options for the 6003 phone and optionally for the 6004.

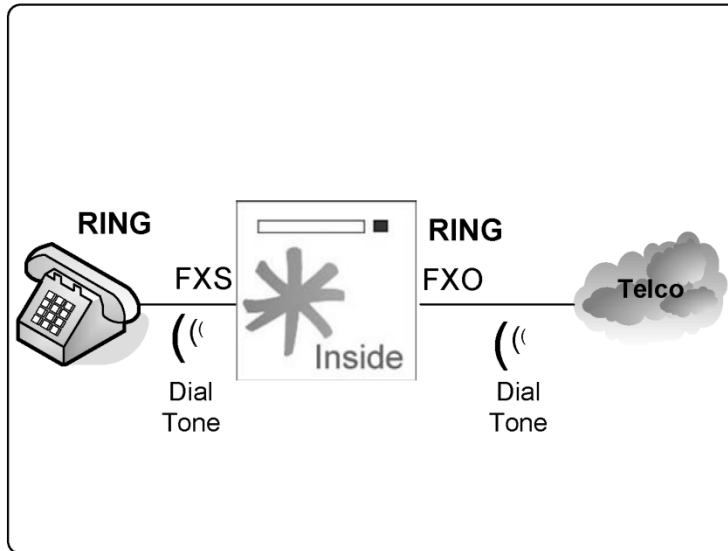


5. Save the configuration and check if the phone is registered using `iax2 show peers`.

Important: Use one account for SIP and another one for IAX. If you want to configure the system to ring both IAX and SIP at the same time, we will show you how to do so in the dial plan section.

Configuring a PSTN interface

To connect to the PSTN, you will need an interface foreign exchange office (FXO) and a telephone line. You can use an existing PBX extension too. You can obtain a telephony interface card with an FXO interface from several manufacturers. In this example, we will show you how to install a DAHDI interface card.



Analog lines using DAHDI

You can buy an analog card compatible with the DAHDI from several manufacturers. X100P was one of the first Digium cards and had already been discontinued. Some manufacturers still produce similar clones. In addition to the price of the X100P, we have found several issues between these cards and new motherboards, so use it with care. X100P, in my opinion, is not a good choice for a production environment. Any card compatible with DAHDI should work.

Thanks to the team of DAHDI developers, we now have a tool for detecting and configuring the interface cards almost automatically. If you have just installed the DAHDI drivers, please don't forget to run `make config` and reboot the machine to load it automatically. You can use the commands below to detect and configure your card.

Step 1: To detect your hardware, use:

`dahdi_hardware`.

Step 2: To configure use:

`dahdi_genconf`.

The command above will generate two files `/etc/system/dahdi.conf` and `/etc/asterisk/dahdi-channels.conf`. The default parameters for `dahdi_genconf` are usually fine, but you can change them in the file `/etc/dahdi/genconf_parameters`. By default, it will insert the lines (FXO) in the context `from-pstn` and the phones (FXS) in the context `from-internal`.

Step 3: After running `dahdi_genconf`, in the last line of the file `/etc/asterisk/chan_dahdi.conf` insert the following line:

```
#include dahdi-channels.conf
```

Step 4: Edit the file `/etc/dahdi/modules` and comment for all the unused drivers. Reboot before proceeding and check if the channels are being recognized using:

```
CLI>dahdi show channels
```

Connecting to the PSTN using a VoIP provider

If your budget is really limited, you can configure a SIP trunk to connect to the PSTN. It is certainly the most affordable way to connect to the PSTN. Thousands of VoIP providers exist worldwide. To connect to one of them, you will need some parameters.

Parameters provided by the SIP provider.

- username: **login**
- password: **secret**
- Provider's domain: **domain**
- UDP port: **5060**
- Allowed codecs: **g729, ilbc, alaw**

Two parameters should be determined by you.

- Extension to receive calls—in this case: **9999**
- context: **from-sip**

Configure the file `sip.conf` using the following parameters:

```
[general]
srvlookup=yes
register => login:secret@domain:port/9999

[siptrunk]
username=login
type=peer
secret=secret
port=5060
insecure=invite
host=dominio
fromuser=login
fromdomain=domain
dtmfmode=rfc2833
context=from-sip
disallow=all
allow=ilbc
allow=alaw
allow=g729
```

To access this trunk, we will use the channel name `SIP/siptrunk`

Dial plan introduction

Dial plan is like Asterisk's heart. It defines how Asterisk handles every single call to the PBX. It consists of extensions that make an instruction list for Asterisk to follow. Instructions are fired by digits received from the channel or application. In order to configure Asterisk successfully, it is crucial to understand the dial plan. Most of the dial plan is contained in the `extensions.conf` file in the `/etc/asterisk` directory. This file uses the simple group grammar and has four major concepts:

- Extensions
- Priorities
- Applications
- Contexts

Let's create a basic dial plan. In subsequent sections of this book, I will devote a chapter exclusively to the dial plan. If you installed the sample files (make samples), the `extensions.conf` already exists. Save it with another name and start with a blank file.

The structure of the file `extensions.conf`

The `extensions.conf` file is separated into sections. The first is the `[general]` section followed by the `[globals]` section. The beginning of each section starts with its name definition (i.e., `[default]`) and finishes when another section is created.

The section `[general]`

The general section sits at the top of the file. Before starting to configure the dial plan, it is helpful to know the general options that control certain dial plan behaviors. These options are:

- **static and write protect:** If `static=yes` and `writeprotect=no`, you can use the CLI command `save dialplan`.

Warning: If you issue a `save dialplan` command from the CLI, you will end up losing any remarks and comments in the file.

- **autofallthrough:** If `autofallthrough` is set, then if an extension runs out of things to do, it will terminate the call with BUSY, CONGESTION, or HANGUP depending on Asterisk's best guess. This is the default. If `autofallthrough` is not set, then if an extension runs out of things to do, Asterisk will wait for a new extension to be dialed. In version 1.4, the default is yes.
- **clearglobalvars:** If `clearglobalvars` is set, global variables will be cleared and reparsed into an dialplan reload or Asterisk reload. If `clearglobalvars` is not set, then global variables will persist through reloads and—even if deleted from the `extensions.conf` or one of its included files—they will remain set to the previous value.

- **extenpatternmatchnew** (new in the 1.6 version): This uses a new algorithm to match the extension from 1.5 to 300 times faster than the existing one, particularly if you have a large number of extensions. It is a new feature and should be used with care; it defaults to **no**.
- **userscontext**: This is the context where the entries from the **users.conf** are registered.

The section [globals]

In the **[globals]** section you will define global variables and their initial values. You can access the variable in the dial plan using **`\${GLOBAL(variable)}`**. You can even access variables defined in the linux/unix environment using **`\${ENV(variable)}`**.

Global variables are not case sensitive. A few examples could be:

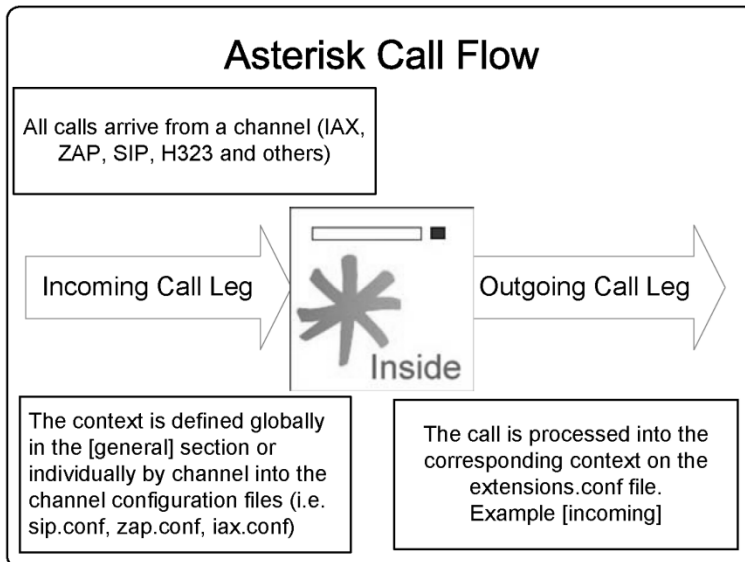
```
INCOMING>DAHDI/8&DAHDI/9
RINGTIME=>3
```

In the following example, you can set and test a global variable in the dial plan.

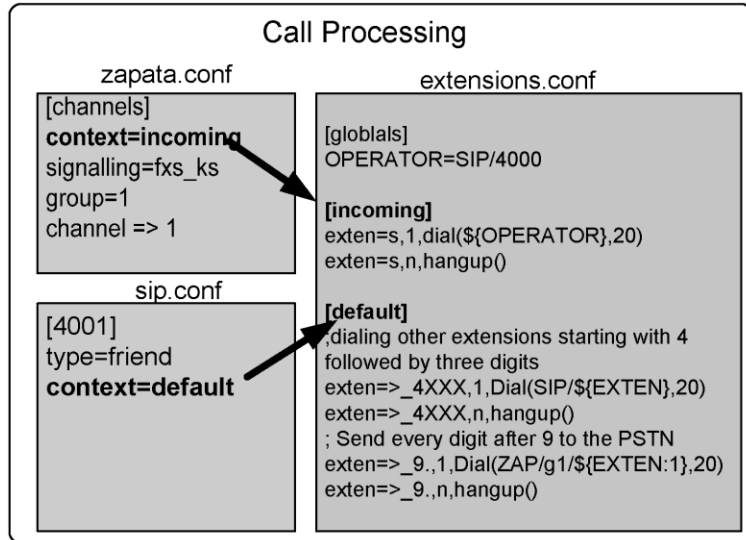
```
exten=9000,1,set(GLOBAL(RINGTIME)=4)
exten=9000,n,Noop(`${GLOBAL(RINGTIME)}`)
exten=9000,n,hangup()
```

Contexts

Context is the named partition of the dial plan. After the **[general]** and **[globals]** sections, the dial plan is a set of contexts in which each context has several extensions, each extension has several priorities, and each priority calls an application with several arguments.



You can build a simple dial plan to reach other phones and the PSTN. However, Asterisk is much more powerful than that. Our objective is to teach you more details of what is possible in the dial plan.



Extensions

Unlike the traditional PBX, where extensions are associated with phones, interfaces, menus, and so on, in Asterisk an extension is a list of commands to be processed when a specific extension number or name is triggered. The commands are processed in priority order.

Extensions syntax	
<i>exten => number (name), {priority/label{+/-}offset}{(alias)},application</i>	
Extension format	
8000	Numeric
Alexander	Alphanumeric
4321/1234	Numeric extension with callerID
_4XXX	Pattern matching
s	Standard
Priority Format	
1-9	Priority Number
n	next
s	same
n+/-x	n+x, n-x
s+/-x	s+x, s-x
hint	Used with presence

An extension can be literal, standard, or special. A standard extension includes only numbers or names and the characters `*` and `#`; `12#89*` is a valid literal extension. Names can be used for extension matching as well. Extensions are case sensitive. However, you cannot create two extensions with the same name but different cases.

When an extension is dialed, the command with the first priority is executed followed by the command with priority 2 and so on. This happens until the call is disconnected or some command returns the number one, indicating failure. What Asterisk does when the last priority is executed is regulated by the parameter `autofallthrough`. See the [\[general\]](#) section in this chapter.

Example:

```
exten=>123,1,Answer
exten=>123,n,Playback(tt-wease1s)
exten=>123,n,Hangup
```

Above you find the list of instructions to be processed when the extension `123` is dialed. The first priority is to answer the channel (necessary when the channel is in the ringing state: i.e., FXO channels). The second priority is to play back an audio file called `tt-wease1s`. The third priority hangs up the channel.

Another option is to handle the call according to the caller ID. You can use the `/` character to specify the caller ID to be processed.

Examples:

```
exten=>123/100,1,Answer()
exten=>123/100,n,Playback(tt-wease1s)
exten=>123/100,n,Hangup()
```

This example will trigger extension **123** and execute the following options only if the caller ID is 100. This can also be done by using the pattern described below:

```
exten=>1234/_256NXXXXXX,1,Answer()
```

hint: maps an extension to a channel. It is used to monitor the channel state. It is used in conjunction with presence. The phone has to support it.

Patterns

You can use patterns and literals in the dial plan. Patterns are very useful for reducing the dial plan size. All patterns start with the “_” character. The following characters may be used to define a pattern. The figure identifies the patterns available for use with Asterisk.

<u>Pattern Matching</u>	
_ (Underscore)	Start of a match
. (dot)	Matches one or more characters
! (exclamation)	Matches zero or more characters
[123-7]	Matches any digit in the brackets (1,2,3to 7)
X	Any digit from 0-9
Z	Any digit from 1-9
N	Any digit from 2-9
<u>Examples</u>	
Extension	Description
_61XX	Sao Paulo's Office (6100 - 6199)
_63XX	New York Office (6300-6399)
_62XX	San Francisco Office (6200 - 6299)
_7[1-3]XX	Bangalore Office (7100-7399)
_7[04-9]XX	Beijing Office (7000-7099, 7400-7999)
_9.	Any number starting with 9
_9XXXXXX	Any number with 8 digits starting with 9

Special extensions

Asterisk uses some extension names as standard extensions.

Asterisk Special Extensions

i	: Invalid
s	: Start
h	: Hangup
t	: Timeout
T	: AbsoluteTimeout
o	: Operator
a	: Called when user press * in voicemail
fax	: Used for fax detection
Talk	: Used with BackgroundDetect

Description:

s: Start. It is used to handle a call when there is no dialed number. It is useful for FXO trunks and in-menu processing.

t: Timeout. It is used when calls remain inactive after a prompt has been played. It is also used to hang up an inactive line.

T: AbsoluteTimeout. If you establish a call limit using the `absolutetimeout()` function, once the call exceeds the limit defined, it will be sent to the T extension.

h: Hangup. It is called after the user disconnects the call.

i: Invalid. It is triggered when you call a non-existent extension in the context. Using these extensions can affect the content of CDR records—specifically, the `dst` that does not contain the number dialed.

o: Operator. It is used to go to operator when the user presses “0” during the voicemail.

The use of these extensions can change the content of the billing records (CDR)—in particular, the field `dst` will not have the number dialed. To work around this problem, you should use the option `g` in the `dial()` application and consider the functions `resetcdr(w)` and/or `nocdr()`

Variables

In the Asterisk PBX, variables can be global, channel-specific, and environment-specific. You can use the `noop()` application to see the content of a variable in the console.

It can use a global variable or a channel-specific variable as applications arguments. A variable can be referenced as in the following example, where `varname` is the name of the variable.

```
/${varname}
```


A variable name can be an alphanumeric string starting with a letter. Global variable names are not case sensitive. However, system variables (Asterisk-defined are channel-defined) are case sensitive. Thus, the variable `${EXTEN}` is different from `${exten}`.

Global variables

Global variables can be configured in the `[global]` section in the `extensions.conf` file or using the application:

```
set(Global(variable)=content)
```

Channel-specific variables

Channel-specific variables are configured using the application `set()`. Each channel receives its own variable space. There is no chance of collisions between variables from different channels. A channel-specific variable is destroyed when the channel hangs up. Some of the most commonly used variables are:

- `${EXTEN}` Extension dialed
- `${CONTEXT}` Current context
- `${CALLERID(name)}`
- `${CALLERID(num)}`
- `${CALLERID(all)}` Current caller ID
- `${PRIORITY}` Current priority

Other channel-specific variables are all uppercase. You can see the content of several variables using the `dumpchan()` application. Below is a simple excerpt of dump-channel variables.

```
exten=9001,1,dumpchan()
exten=9001,n,echo()
exten=9001,n,hangup()
```

Dumpchan output:

```
Dumping Info For Channel: SIP/4400-08191828:
```

```
=====
Info:
Name=                SIP/4400-08191828
Type=                SIP
UniqueID=            1161186526.0
CallerID=            4400
CallerIDName=        laptop
DNIDDigits=          9001
RDNIS=               (N/A)
State=               Ring (4)
Rings=               0
NativeFormat=        0x4 (ulaw)
WriteFormat=         0x4 (ulaw)
ReadFormat=          0x4 (ulaw)
1stFileDescriptor=  16
Framesin=            0
Framesout=           0
```

```
-TimetoHangup=      0
ElapsedTime=       0h0m0s
Context=           default
Extension=         9001
Priority=          1
CallGroup=
PickupGroup=
Application=       DumpChan
Data=              (Empty)
Blocking_in=      (Not Blocking)
```

Variables:

```
SIPCALLID=500CEBC0-9483-4CED-B1E4-16D953655CFC@192.168.1.116
SIPUSERAGENT=SJphone/1.61.312b (SJ Labs)
SIPDOMAIN=192.168.1.133
SIPURI=sip:4400@192.168.1.116
```

Environment-specific variables

Environment-specific variables can be used to access variables defined in the operating system. You can set environment-specific variables using the function `ENV()`. For example:

```
 ${ENV(LANG)}
Set(ENV(LANG))=en_US
```

Application-specific variables

Some applications use variables for data input and output. You can set variables before calling the application or retrieve the variable after the application execution. For example:

The Dial application returns the following variables:

- `${DIALEDTIME}` ->This is the time from dialing a channel until it is disconnected.
- `${ANSWEREDTIME}` -> This is the amount of time for the actual call.
- `${DIALSTATUS}` This is the status of the call:
 - CHANUNAVAIL
 - CONGESTION
 - NOANSWER
 - BUSY
 - ANSWER
 - CANCEL
 - DONTCALL
 - TORTURE
- `${CAUSECODE}` -> Error message for the call.

Expressions

Expressions can be very useful in the dial plan. They are used to manipulate strings and perform math and logical operations.

Asterisk Expressions

`$(expression1 operator expression2)`

Math Operators

Addition (+), Subtraction (-), Multiplication(*), Division (/), Modulus (%)

Logical Operators

Logical "AND" (&), Logical "OR" (|), Unary not (!)

Comparison Operators

(=, >, >=, <, <=, !=)

Regular expression operators

Regular expression matching (:), Regular expression exact matching (=~)

Conditional operator

`expression1 ? expression2 :: expression3`

The expression syntax is defined as follows:

`$(expression1 operator expression2)`

Let's suppose that we have a variable called "I" and we want to add 100 to the variable:

`$$[I]+100`

When Asterisk finds an expression in the dial plan, it changes the entire expression by the resulting value.

Operators

The following operators can be used to build expressions. It is important to observe operator precedence.

1. Parentheses "()"
2. Unary operators "! -"
3. Regular expression ": =~"
4. Multiplicative operators "* / %"
5. Additive operators "+ -"
6. Comparison operators
7. Logical operators
8. Conditional operators

Math Operators

- Addition (+)
- Subtraction (-)

- Multiplication (*)
- Division (/)
- Modulus (%)

Logical Operators

- Logical “AND” (&)
- Logical “OR” (|)
- Logical Unary Complement (!)

Regular expression operators

- Regular expression matching (:)
- Regular expression exact matching (=~)

A regular expression is a special text string used to describe a search pattern. You can think of regular expressions as wildcards. Regular expressions are used to match a string to a pattern to check the matching. If the match succeeds and the regular expression contains at least one match, the first match is returned; otherwise, the result is the number of characters matched.

Comparison operators

The result of a comparison is 1 if the relation is true or 0 if it is false.

- = equal
- != not equal
- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to

LAB. Evaluate the following expressions:

Put these expressions in your dial plan and use the **NoOP()** application to evaluate the expressions. Dial **9002** and examine the results in the Asterisk console. Use verbose 15 to show the results.

```
exten=9002,1,set(NAME="FLAVIO") ;Set NAME=FLAVIO
exten=9002,n,set(I=4)
exten=9002,n,set(URI="40001@asteriskguide.com")
exten=9002,n,NoOP(${NAME})
exten=9002,n,NoOP(${I})
exten=9002,n,NoOP(${I}${I})
exten=9002,n,NoOP(${I}=4)
exten=9002,n,NoOP(${I}=4 & ${NAME}=FLAVIO)
exten=9002,n,NoOP(${URI} =~ "4[0-9][0-9][0-9][0-9]@.")
```

```
exten=9002,n,NoOp(${I}=4?"MATCH"::"DO NOT MATCH")
exten=9002,n,hangup
```

Functions

After version 1.2, some applications were replaced by functions to allow the processing of certain variables in a more advanced way than only expressions. You can see the full list of functions by issuing the following console command:

```
CLI>core show functions
```

String length: `{LEN(string)}` returns the string length

Example:

```
exten=>100,1,Set(Fruit=pear)
exten=>100,2,NoOp(${LEN(Fruit)})
exten=>100,3,NoOp(${LEN(${Fruit})})
```

In the first operation, the system shows 5 as the result (the number of letters in the word “fruit”). The second returns the number 4 (the number of letters in the word “pear”).

Substrings: Returns the substring, starting from the positing defined by the “offset” parameter, with the string length defined in the “length” parameter. If the offset is negative, it starts from right to left, beginning at the end of the string. If the length is omitted or negative, it takes the whole string starting with the offset.

```
{string:offset:length }
```

Example #1: Several substrings

```
{123456789:1}-returns 23456789
{123456789:-4}-returns 6789
{123456789:0:3}-returns 123
{123456789:2:3}-returns 345
{123456789:-4:3}-returns 678
```

Example #2: Take the area code from the first three digits.

```
exten=>_NXX.,1,Set(areacode=${EXTEN:0:3})
```

Example #3: Takes all digits from the variable `{EXTEN}`, except for the area code.

```
exten=>_516XXXXXXX,1,Dial(${EXTEN:3})
```

String concatenation

To concatenate two strings, simply write them together.

```
{foo}{bar}
555{number}
${longdistanceprefix}555{number}
```

Applications

To build a dial plan, we need to understand the concept of applications. You will use applications to handle the channel in the dial plan. Applications are implemented in several modules. Available applications depend on modules. You can show all Asterisk applications using the console command:

```
CLI>core show applications
```

Alternatively, you can show details of a specific application using the following example:

```
CLI>core show application dial
```

To build a simple dial plan, you need to know a few applications. We will discuss more advanced examples later in the book.

Simple applications to build a dialplan

- Answer – Answer a channel
- Dial – Dial other channel
- Hangup – Hang up a channel
- Playback – Play back an audio file
- Goto – Jump to a particular priority, extension or context

We will use these applications (above) to create a simple dial plan for two basic PBXs.

Answer()

[Synopsis]

Answers a channel if ringing

[Description]

Answer([delay]): If the call has not been answered, the application will answer it. Otherwise, it has no effect on the call. If a delay is specified, Asterisk will wait the number of milliseconds specified in 'delay' before answering the call.

Dial()

The following description can be obtained by issuing the show application dial in the dial plan. For easy searching, it is reproduced below. The syntax for the Dial application is also shown below:

```
;dial to a single channel
Dial(type/identifier,timeout,options, URL)
```

```
;Dialing to multiple channels
Dial(Technology/resource[&Tech2/resource2...][|timeout][|options][|URL]):
```

This application will place calls to one or more specified channels. As soon as one of the requested channels answers, the originating channel will be answered—if it has not already been answered. These two channels will then be active in a bridged call. All other requested channels will then be hung up.

Unless a timeout is specified, the Dial application will wait indefinitely until one of the called channels answers, the user hangs up, or all of the called channels are busy or unavailable. The execution of the dial plan will continue if no requested channels can be called or if the timeout expires. This application sets the following channel variables upon completion:

- DIALEDTIME - This is the time from dialing a channel until the time that it is disconnected.
- ANSWEREDTIME - This is the amount of time for an actual call.
- DIALSTATUS - This is the status of the call:
 - CHANUNAVAIL
 - CONGESTION
 - NOANSWER
 - BUSY
 - ANSWER
 - CANCEL
 - DONTCALL
 - TORTURE

For the Privacy and Screening Modes, the **DIALSTATUS** variable will be set to **DONTCALL** if the called party chooses to send the calling party to the 'Go Away' script. The **DIALSTATUS** variable will be set to **TORTURE** if the called party wants to send the caller to the 'torture' script.

This application will report normal termination if the originating channel hangs up or if the call is bridged and either of the parties in the bridge ends the call.

The optional URL will be sent to the called party if the channel supports it. If the **OUTBOUND_GROUP** variable is set, all peer channels created by this application will be included in that group (as in **Set(GROUP)=...**).

The following table summarizes some of the most frequently used options for the application dial. For the complete list, use the console command **core show application dial**.

A(x)	Plays an announcement to the called party, using 'x' as the file.
C	Resets the CDR for this call.
D	Allows the calling user to dial a 1-digit extension while waiting for a call to be answered. Exits to that extension if it exists in the current context or to the context defined in the EXITCONTEXT variable, if it exists.
D([called][:calling])	Sends the specified DTMF strings <u>after</u> the called party has answered, but before the call gets bridged. The 'called' DTMF

	string is sent to the called party, and the 'calling' DTMF string is sent to the calling party. Both parameters can be used alone.
f	Forces the caller ID of the <u>calling</u> channel to be set as the extension associated with the channel using a dial plan 'hint'. For example, some PSTNs do not allow caller ID to be set to anything other than the number assigned to the caller.
g	Proceeds with dial plan execution at the current extension if the destination channel hangs up.
G(context^exten^pri)	If the call is answered, transfers the calling party to the specified priority and the called party to the specified priority+1. Optionally, an extension—or extension and context—can be specified. Otherwise, the current extension is used.
h	Allows the called party to hang up by sending the '*' DTMF digit
H	Allows the calling party to hang up by hitting the '*' DTMF digit.
L(x[:y][:z])	Limits the call to 'x' ms. Plays a warning when 'y' ms are left. Repeats the warning every 'z' ms. The following special variables can be used with this option: LIMIT_PLAYAUDIO_CALLER yes no (default yes) Plays sounds for the caller. LIMIT_PLAYAUDIO_CALLEE yes no Plays sounds for the person called. LIMIT_TIMEOUT_FILE File to be played when time is up. LIMIT_CONNECT_FILE ->File to be played when the call begins. LIMIT_WARNING_FILE ->File to be played as a warning if 'y' is defined. The default is to say the time remaining.
m([class])	Provides hold music to the calling party until a requested channel answers. A specific MusicOnHold class can be specified.
r	Indicates ringing to the calling party. Passes no audio to the calling party until the called channel has answered.
S(x)	Hangs up the call 'x' seconds <u>after</u> the called party has answered the call.
t	Allows the called party to transfer the calling party by sending the DTMF sequence defined in features.conf .
T	Allows the calling party to transfer the called party by sending the DTMF sequence defined in features.conf .
w	Allows the called party to enable recording of the call by sending the DTMF sequence defined for one-touch recording in features.conf .
W	Allows the calling party to enable recording of the call by sending

	the DTMF sequence defined for one-touch recording in features.conf .
K	Allows the called party to enable parking of the call by sending the DTMF sequence defined for call parking in features.conf.
K	Allows the calling party to enable parking of the call by sending the DTMF sequence defined for call parking in features.conf.

Example:

```
exten=_4XXX,1,Dial(SIP/${EXTEN},20,tTm)
```

In the example above, the application will dial to the corresponding SIP channel. Both caller and called could transfer the call (**tT**). Music on hold will be heard instead of ring back. If nobody answers within 20 seconds, the extension will go to the next priority.

Hangup()

Hangs up the calling channel

[Description]

hangup([causecode]): This application will hang up the calling channel. If a cause code is given, the channel's hang-up cause will be set to the given value.

Goto()

Jump to a particular priority, extension, or context

[Description]

goto([[context]|extension]|priority): This application will cause the calling channel to continue the dial plan execution at the specified priority. If no specific extension (or extension and context) are specified, this application will jump to the specified priority of the current extension. If the attempt to jump to another location in the dial plan is not successful, the channel will continue at the next priority of the current extension.

Building a dial plan

To build a simple dial plan, you need to treat all incoming and outgoing calls by creating contexts and extensions. In this section, we will show you how to build the most common extensions.

Dialing between extensions

To enable dialing between extension, we could use the channel variable `${EXTEN}`, which refers to the dialed extension. For example, if the extension range is between 4000 and 4999 and all extensions use SIP, we could adopt the following command:

```
[from-internal]
exten=_4XXX,1,Dial(SIP/${EXTEN})
```

Dialing to an external destination

To dial an external destination you could precede the number dialed with a route. In North America, it is common to use 9 followed by the number to be dialed externally. If you are using an analog or digital channel to the PSTN, the command should look like the following:

If you want to use the SIP trunk instead of the DAHDI, use SIP/trunk as the channel

```
[from-internal]
exten=_9NXXXXXX,1,Dial(DAHDI/1/${EXTEN:1},20,tT)
```

or

```
exten=_9NXXXXXX,1,Dial(SIP/trunk/${EXTEN:1},20,tT)
```

The above line will permit you to dial 9 and the desired number. In the example given, you will use the first DAHDI channel (DAHDI/1). If you have several lines and this one is busy, the call will not be completed. However, you could use the following line to automatically choose the first available DAHDI channel. Optionally, you can use the SIP trunk instead of DAHDI.

```
[from-internal]
exten=_9NXXXXXX,1,Dial(DAHDI/g1/${EXTEN:1},20,tT)
```

The “g1” parameter will search for the first available channel in the group, allowing the use of all channels. Using the line below, you could dial a long distance number.

```
[from-internal]
exten=_91NXXNXXXXXX,1,Dial(DAHDI/g1/${EXTEN:1},20,tT)
```

Dialing 9 to get a PSTN line

If you do not have any restrictions to external dialing, you could simplify and use the following:

```
[from-internal]
exten=9,1,Dial(DAHDI/g1,20,tT)
```

Receiving a call in the operator extension

In the following example, the operator extension is 4000. The PSTN line is connected to an FXO interface. In the `chan_dahdi.conf` file, the context specified is `from-pstn`. Any call coming from the PSTN will be routed to the context `from-pstn` in the dial plan. This line does not have direct inward dialing (DID); as such, we will have to receive the call via the “s” extension. If receiving from the SIP trunk, use the context `[from-sip]`.

```
[globals]
OPERATOR=SIP/6000

[from-pstn]
exten = s,1,Dial(${OPERATOR},40,tT)
```

```
exten = s,n,Hangup()

[from-sip]
exten = s,1,Dial(${OPERATOR},40,tT)
exten = s,n,Hangup()
```

Receiving a call using direct inward dialing (DID)

If you have a digital line, you will receive the dialed extension. When this is the case, you don't need to forward the call to the operator; rather, you can forward the call directly to the destination. Suppose your DID range is from 3028550 to 3028599 and the last four numbers are passed in the DID. The configuration would look like the following example:

```
[from-pstn]
exten => _85[5-9]X,1,Answer()
exten => _85[5-9]X,n,Dial(SIP/${EXTEN},15,tT)
exten => _85[5-9]X,n,Hangup()
```

Playing several extensions simultaneously

You can set Asterisk to dial an extension and, if it is not answered, to dial several other extension simultaneously, as indicated in the following example:

```
exten => 0,1,Dial(DAHDI/1,15,tT)
exten => 0,n,Dial(DAHDI/1&DAHDI/2&DAHDI/3,15)
exten => 0,n,Hangup()
```

In this example, when someone dials the operator, the channel DAHDI/1 is initially tried. If nobody answers after 15 seconds (timeout), the channels DAHDI/1, DAHDI/2 and DAHDI/3 will ring simultaneously for another 15 seconds.

Routing by Caller ID

In this example, you could give different treatments based on the caller ID, which could be useful for call spammers. For example:

```
exten => 8590/4832518888,1,Playback(I-have-moved-to-china)
exten => 8590,1,Dial(DAHDI/1,20)
```

In this example, we have added a special rule that, if the caller ID is 4832518888, you play back a message from the previously recorded file "I-have-moved-to-china". Other calls are accepted as usual.

Using variables in the dial plan

Asterisk can use global and channel variables in the dial plan as arguments for certain applications. Look at the following examples:

```
[globals]
Flavio => DAHDI/1
Daniel => DAHDI/2&SIP/pingtel
```

```
Anna => DAHDI/3
Christian => DAHDI/4
```

```
[mainmenu]
exten => 1,1,Dial(${Daniel}&${Flavio})
exten => 2,1,Dial(${Anna}&${Christian})
exten => 3,1,Dial(${Anna}&${Flavio})
```

Using variables makes future changes easier. If you change the variable, all references are changed immediately.

Recording an announcement

In some of the options discussed later in this section, we will use recorded prompts. Here we show you an easy way to record them. We will use the application `Record()` to save the announcement using one's own phone.

```
[from-internal]
exten => _record.,1,Record(${EXTEN:6}:gsm)
exten => _record.,n,wait(1)
exten => _record.,n,Playback(${EXTEN:6})
exten => _record.,n,Hangup()
```

These instructions allow you to record any message from a soft-phone.

Example: dialing `recordmenu` from the softphone

The instructions will call the recording with the variable `${EXTEN:6}` without the first six letters. In other words, the instruction is equivalent to `record(menu:gsm)`. All you have to do is dial `record + name_of_the_file_to_be_recorded`, press `#` to finish the recording, and wait to hear the recording.

Receiving the calls in a digital receptionist

Now that we have some simple examples, let's expand our learning about the applications `background()` and `goto()`. The key for interactive systems in Asterisk is the application `background()`, which allows you to execute an audio file that, when the caller presses a key, is interrupted in order to send the call to the extension dialed.

Syntax of the `background()` application:

```
exten=>extension, priority, background(filename)
```

Another application very useful is `goto()`. As the name implies, it jumps to the context, extension, and priority indicated.

Syntax of the application `goto()`:

```
exten=>extension, priority, goto(context, extension, priority)
```

Valid formats for the `goto()` command:

```
goto(context,extension,priority)
goto(extension,priority)
```

```
goto(priority)
```

In the following example, we will create a digital receptionist. It is very simple to edit the file `extensions.conf` and configure the following extensions:

```
[globals]
OPERATOR=SIP/6000

[from-pstn]
include=aapstn

[from-sip]
include=aasip

[aapstn]
exten=>s,1,answer()
exten=>s,n,set(TIMEOUT(response)=10)
exten=>s,n,background(menu1)
exten=>s,n,waitExten(30)
exten=>s,n,Dial(${OPERATOR})
exten=>6000,1,Dial(SIP/6000)
exten=>6001,1,Dial(SIP/6001)
exten=>6003,1,Dial(IAX2/6003)
exten=>6004,1,Dial(IAX2/6004)

[aasip]
exten=>9999,1,answer()
exten=>9999,n,set(TIMEOUT(response)=10)
exten=>9999,n,background(menu1)
exten=>s,n,waitExten(30)
exten=>9999,n,Dial(${OPERATOR})
exten=>6000,1,Dial(SIP/6000)
exten=>6001,1,Dial(SIP/6001)
exten=>6003,1,Dial(IAX2/6003)
exten=>6004,1,Dial(IAX2/6004)
```

In the file `menu1.gsm`, record the message “press the extension or wait for the operator”. When the user dials the number 6000, he will be sent to extension 6000.

At this point, you should have a clear understanding of the use of several applications, including `answer()`, `background()`, `goto()`, `hangup()`, and `playback()`. If you do not have a clear understanding, please read this chapter again until you feel comfortable with the content. You will use the background application very often.

Once you understand the basics of extensions, priorities, and applications, it will be easy to create a simple dial plan. These concepts will be explored in greater depth later in the book, and you will see that the dial plan will become more powerful.

Summary

In this chapter, you've learned that configuration files are stored in the `/etc/asterisk directory`. To use Asterisk, it is first necessary to configure the channels (e.g., sip, dahdi, iax). Three different grammars exist for configuration files: simple group, object inheritance, and complex entity. The dial plan is created in the file `extensions.conf` and is a set of contexts and extensions. In the dial plan, each extension triggers an application. You've learned to use `playback`, `background`, `dial`, `goto`, `hangup`, and `answer` applications.

Quiz

1. The channel configuration files are:

- A. `/etc/dahdi/system.conf`
- B. `/etc/asterisk/chan_dahdi.conf`
- C. `sip.conf`
- D. `iax.conf`

2. It is important to define a context in the channel configuration file as this will define the incoming context for a call. In the extensions configuration file `extensions.conf`, a call from this channel will be processed in the matching incoming context.

- A. True
- B. False

3. The main differences between the `playback()` and `background()` applications are (choose two):

- A. Playback simply plays a prompt, but does not wait for digits.
- B. Background simply plays a prompt, but does not wait for digits.
- C. Background plays a message and waits for digits to be pressed.
- D. Playback plays message and waits for digits to be pressed.

4. When a call gets into Asterisk using a telephony interface card (FXO), this call is handled in the special extension:

- A. `'0'`
- B. `'9'`
- C. `'s'`
- D. `'i'`

5. Valid formats for the `goto()` application are (choose three):

- A. `Goto(context,extension, priority)`
- B. `Goto(priority, context, extension)`
- C. `Goto(extension,priority)`

- D. Goto(priority)
6. An extension cannot be defined as (choose all correct answers):
- A. An alphanumeric literal
 - B. A numeric literal
 - C. A pattern beginning with a “.” (dot) character
 - D. A pattern starting with a “_” (underscore) character
7. The pattern `_7[1-5]XX` matches (choose all correct answers):
- A. 7100
 - B. 7600
 - C. 7630
 - D. 7230
8. An incoming context for a DAHDI-compatible telephony interface is defined in the _____ configuration file:
- A. `/etc/dahdi/system.conf`
 - B. `/etc/asterisk/chan_dahdi.conf`
 - C. `/etc/asterisk/asterisk.conf`
 - D. `/etc/asterisk/modules.conf`
9. In the Options Inheritance grammar used by `chan_dahdi.conf`, you:
- A. Define the object in a single line.
 - B. Define options first and declare the objects below the defined options.
 - C. Define a context for each object.
10. Priorities must be consecutive!
- A. False
 - B. True

4

Analog channels

There are several ways to connect the public switched telephone network (PSTN). The best way depends on how the telephone company makes this connection available in your area. The simplest way is to use an analog line, similar to the line you use at home. In this section, we will show you how to configure analog cards from Digium™ and Xorcom™.

Objectives

By the end of this chapter you should be able to:

- Recognize the main telephony terms and acronyms;
- Understand when to use digital and analog circuits;
- Recognize the difference between FXS and FXO; and
- Configure Asterisk for FXS and FXO.

Telephony basics

Most analog implementations use a pair of copper lines named tip and ring. When a loop is closed, the phone receives the dial tone from the telecom switch (or the private PBX). The most frequently used signaling is loop-start; other, less common kinds of signaling including ground start, which is used in several countries. The three categories of signaling are:

- Supervision signaling
- Address signaling
- Information signaling

Supervision signaling

The main supervision signalings are on-hook, off-hook, and ringing.

On-Hook – When a user puts the phone on the hook, the PBX interrupts and does not allow the electric current to pass. In this state, the circuit is named on-hook. In this position, only the ringer is active.

Off-Hook – Before starting a phone call, the phone needs to pass to the off-hook state. Removing the handset from the hook closes the loop and indicates to the PBX that the user intends to make a call. Upon receiving this indication, the PBX generates a dial tone, indicating to the user that it is ready to accept the destination address (i.e., phone number).

Ringing – When a user calls another phone, it generates a voltage to the ringer that warns the other user about a call being received.

Signaling varies by country, with different tones for different countries. You can personalize Asterisk tones to your country by modifying the `indications.conf` file. For example:

```
[br]
description=Brazil
ringcadance=1000,4000
dial=425
busy=425/250,0/250
ring=425/1000,0/4000
congestion=425/250,0/250,425/750,0/250
callwaiting=425/50,0/1000
```

Address Signaling

You can use two kinds of signaling for dialing. The first and most common is dual tone multi-frequency (dtmf) while the other is pulse dialing (used in old rotary dial phones). Phones have a keypad for dialing, and each button is associated with two frequencies: one high and one low. In the case of dtmf signaling, the combination of these tones indicates what digit is being pressed. MFC/R2 uses a multi-frequency tone different from dtmf.

Information signaling

Information signaling shows the call's progress and different events.

- Dial tone
- Busy Tone
- Ringback
- Congestion
- Invalid number
- Confirmation tone

PSTN interfaces

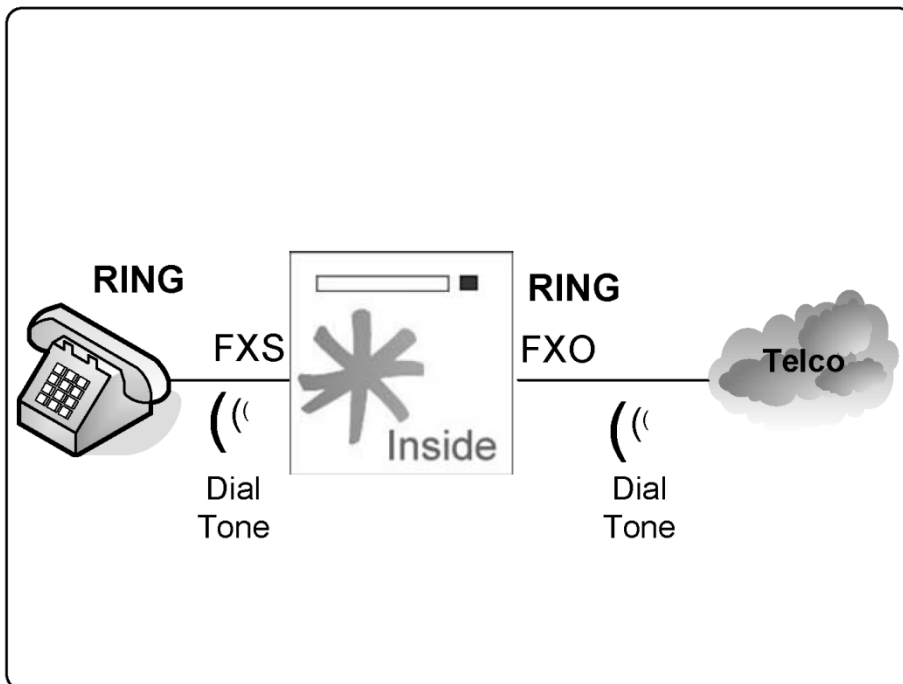
As in the case of old PBXs, it is often required to connect the Asterisk PBX to the PSTN. Here we'll show you how to do it. Usually you have three options for telephone lines.

- **Analog:** The most common form for home and small business, usually delivered with a metallic pair of copper lines.

- **Digital:** Used when many lines are necessary. A digital line is usually delivered by a CSU/DSU or a Fiber multiplexer. The end user connector is usually a RJ45. In some countries, E1 lines are delivered using two coaxial BNC connectors; in this case you will need a balloon to connect to the RJ45 jack to the telephony board.
- **SIP:** This option has been recently developed. The telephone line is delivered using a data connection with SIP signaling (VoIP). This is a good option to use with Asterisk since you will not need to buy a telephony card. Phone calls will be delivered directly to the Ethernet port. Another advantage is that you may be able to free resources from your CPU by avoiding codec transcoding.

Analog FXS, FXO, and E&M interfaces

Several types of analog interfaces are available. It is fundamental to understand the differences between these interfaces to learn how to connect to the phone network as well as to other PBXs. Here, we will show you the E&M interface. Although it is not currently available for Asterisk and has been discontinued by several vendors, you may find routers and PBXs with this kind of interface, so it is better to know what you are dealing with.



Foreign eXchange (FX) Interfaces

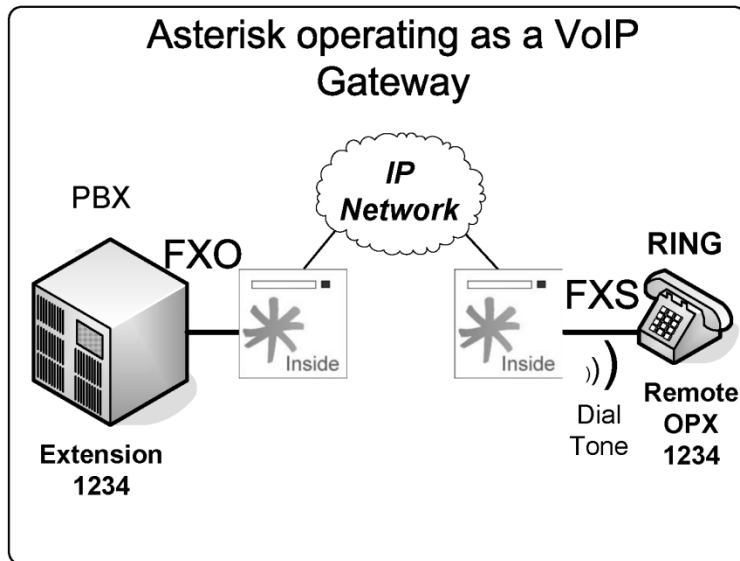
FX interfaces are analog. The term “Foreign eXchange” is applied to access trunks to a PSTN central office (CO).

Foreign eXchange Office (FXO)

The FXO interface is used to connect to a central office (CO) or another PBX's extension. It communicates directly with a telephone line coming from the PSTN. Another option is to connect the FXO interface to an existing PBX, allowing communication between Asterisk and the legacy PBX. Connecting Asterisk to a PBX port and delivering a remote extension using VoIP is often referred to as an off-promises extension (OPX). An FXO interface receives a dial tone.

Foreign eXchange Station (FXS)

The FXS interface feeds an analog phone, modem, or fax. The FXS provides the dial tone and power for a phone.



Trunk signaling

- Loop-Start
- Ground-Start
- Kewlstart

The use of kewlstart signaling in Asterisk is almost default. Kewlstart is not signaling itself, but adds intelligence to the circuit by monitoring what is happening on the other side. Kewlstart is based in loop-start. Most switches do not support this feature, which is used to get the hang-up notification.

- Loopstart: Used in most analog lines, it allows the telephone to indicate “on-hook” and “off-hook” and the switch to indicate “ring” and “no-ring”. This is probably what most people have at home. The name comes from the fact that the line is always open. When you close the loop, the switch provides you with a dial tone. An incoming call is signaled by a 100V ringing voltage over the open pair.

- **Groundstart:** Similar to Loopstart. When you want to make a call, one side of the line is short-circuited. When the switch identifies this state, it reverses the voltage through the open pair, and then the loop is closed. Consequently, the line first becomes occupied before being offered to the caller.
- **Kewlstart:** Adds intelligence to the circuits, allowing monitoring of the other side. Kewlstart incorporates many advantages from loop-start.

Asterisk telephony channels setup

To configure a telephony interface card, several steps are necessary. In this chapter, we will show three of the most common scenarios:

- Analog connection using FXS
- Analog connection using FXO
- Connection of an Astribank™ with FXS and FXO interfaces

Configuration Procedure (valid in both cases)

Before choosing hardware for Asterisk, you should consider the number of simultaneous calls, services, and codecs that are going to be installed and enabled. Asterisk is a CPU-intensive application, which is why we recommend a dedicated machine for Asterisk.

The number of interface cards installed within the computer is limited by the number of slots and interruptions available. It is preferable to install a single card with eight voice interfaces than two cards with four. Another option is to use a USB channel bank, such as the Xorcom Astribank. Recently, some manufacturers (e.g., CIANET) have started producing TDMoE channel banks, making it even easier to connect dozens of analog interfaces.



Astribank 19" with 32 FXS/FXO ports

Example 1: One FXO, one FXS installation

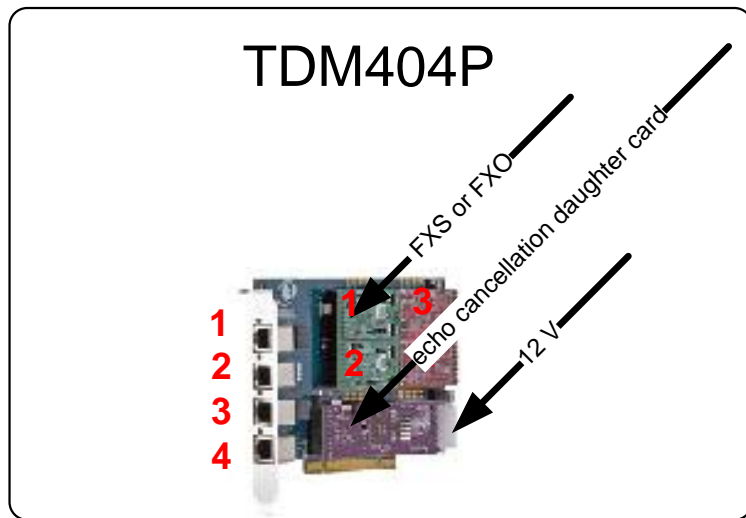
Several devices (e.g., Digium cards, Astribanks) use a set of kernel modules formerly known as Zaptel. Because of trademark disputes, the Zaptel drivers were renamed Digium Asterisk Hardware Device Interface (DAHDI) in 2008. The Zaptel drivers are still available, but they will likely be discontinued, which is why I have chosen to use the new DAHDI drivers in this book. In this example, we will use a Digium TDM400 telephony interface card with one FXS and one FXO module. The required steps are listed below:

1. Install the analog card FXS, FXO, or both.
2. Configure the file `/etc/DAHDI/system.conf` (formerly `/etc/zaptel.conf`).
3. Generation of the configuration files using `DAHDI_genconf`.
4. Load the driver for the DAHDI interface.

5. Execute `dahdi_test` to verify interrupt misses.
 6. Execute `dahdi_cfg` to configure the driver.
 7. Configure the channel DAHDI in `chan_dahdi.conf` file.
- Load Asterisk.

Step 1: Install the TDM400 Board.

The TDM404P card contains FXS and FXO modules. Connect the FXS (S110M, green) and FXO (X100M, red) modules. If you are using FXS modules, connect the card directly to the power source using a molex connector. Please wear electrostatic protection before handling interface cards to avoid damage to the hardware. Digium analog cards also support a hardware echo cancellation module VPMADT032.



Step 2: The good news about the configuration is the new utility `dahdi_genconf`, which automatically detects and generates the configuration for DAHDI interfaces. The utility generates two files:

- `/etc/dahdi/system.conf`
- `/etc/asterisk/dahdi-channels.conf`
- `/etc/asterisk/users.conf` (option: users)
- All these files use the option `chan_dahdi full`

Before you can execute the `dahdi_genconf`, it is important to configure the file `gen_parameters.conf`

```
#
# /etc/dahdi/genconf_parameters
#
# This file contains parameters that affect the
# dahdi_genconf configurator generator.
```

```

#
#base_exten      4000
#fxs_immediate   no
#fxs_default_start ks
#lc_country      il
#context_lines   from-pstn
#context_phones  from-internal
#context_input   astbank-input
#context_output  astbank-output
#group_phones    0
#group_lines     5
#brint_overlap
#bri_sig_style   bri_ptmp
#
# The echo canceller to use. If you have a hardware echo canceller, just
# leave it be, as this one won't be used anyway.
#
# The default is mg2, but it may change in the future. E.g: a packager
# that bundles a better echo canceller may set it as the default, or
# dahdi_genconf will scan for the "best" echo canceller.
#
#echo_can        hpec
#echo_can        oslec
#echo_can        none # to avoid echo cancellers altogether

# bri_hardhdlc: If this parameter is set to 'yes', in the entries for
# BRI cards 'hardhdlc' will be used instead of 'dchan' (an alias for
# 'fcshdlc').
#
#bri_hardhdlc    yes

# For MFC/R2 Support
#pri_connection_type R2
#r2_idle_bits    1101

# pri_types contains a list of settings:
# Currently the only setting is for TE or NT (the default is TE)
#
#pri_termtype
# SPAN/2        NT
# SPAN/4        NT
O arquivo gen_parameters.conf permite a personalização da sua configuração. Os
parâmetros mais importantes para linhas analógicas são:
base_exten      4000
#fxs_immediate   no
fxs_default_start ks
lc_country      br
context_lines    from-pstn

```

```
context_phones    from-internal
context_input     astbank-input
context_output    astbank-output
group_phones      0
group_lines       5
#echo_can         hpec
#echo_can         oslec
echo_can          MG2
```

Warning: It is required that you configure at least the echo cancellation algorithm for the channels.

The `base_exten` parameter defines the basic dial plan for FXS extensions. In this case, the first FXS channel will receive the extension number 4000, the second 4001, and so on. The context in which the lines (`context_phones`) and trunks (`context_lines`) are created is very important. After generating the files, you should include the file `/etc/asterisk/dahdi-channels.conf` in the file `/etc/asterisk/chan_dahdi.conf`.

```
#include dahdi-channels.conf
```

Note: Analog signaling is a bit confusing; it is always the inverse of the card. FXS cards are signaled with FXO whereas FXO cards are signaled with FXS. Asterisk talks to these devices as if it was on the opposite side.

Step 3: Load kernel drivers.

Now you have to load the `chan_dahdi` module and the related card kernel driver. Use `dahdi_hardware` to detect your card and the driver name. For example:

Card	Driver	Description
TE410P	wct4xxp	4xE1/T1-3.3V PCI
TE405P	wct4xxp	4xE1/T1-5V PCI
TDM400P	wctdm	4 FXS/FXO
T100P	wctlxxp	1 T1
E100P	wctlxxp	1 E1
X100P	wcfxo	1 FXO

Commands to load the drivers:

```
modprobe dahdi
modprobe wctdm
```

Step 4: Use the `dahdi_test` utility.

An important utility is `dahdi_test`, which is used to verify interrupt misses in the DAHDI card. Audio quality problems are often related to interrupt conflicts.

To verify that your DAHDI card is not sharing an interrupt with other cards, use the following command:

```
#cat /proc/interrupts
```

You can verify the number of interrupt misses using the `dahdi_test` utility compiled with the DAHDI cards. A number below 99.987% indicates possible problems.

Step 5: Use the `dahdi_cfg` utility to configure the driver.

DAHDI has an unusual system for loading the drivers. First configure the `/etc/system/dahdi.conf`, and then apply those configurations to the DAHDI driver using `dahdi_cfg`.

In this case, `dahdi_cfg` is used to configure the signaling for the FX interfaces. To see the results, you can append “-vvvvv” to the command for verbose.

```
#
/sbin/dahdi_cfg -vv
Dahdi Configuration
=====
Channel map:
Channel 01: FXS Kewlstart (Default) (Slaves: 01)
Channel 02: FXO Kewlstart (Default) (Slaves: 02)
2 channels configured.
```

If the channels were loaded successfully, you will see an output similar to the one shown above.

Users often incorrectly configure `chan_dahdi.conf` with inverted signaling between channels. If this happens, you will see a message like the one shown below:

```
DAHDI_CHANCONFIG failed on channel 1: Invalid argument (22)
Did you forget that FXS interfaces are configured with FXO signalling
and that FXO interfaces use FXS signalling?
```

After successfully configuring the hardware, you can proceed to Asterisk configuration.

Step 6: `/etc/dahdi/system.conf` configuration file.

It sounds strange, but after configuring the `/etc/dahdi/system.conf`, you configured the card itself. DAHDI can be used for other purposes, like routing and SS7. To use it with Asterisk, you must configure the Asterisk DAHDI channels. Every channel in Asterisk has to be defined; SIP channels are defined in `sip.conf` while TDM channels are defined in `chan_dahdi.conf`. This creates the logical TDM channels to be used in your dial plan.

```
signalling=fxs_ks;
group=1;          channel group
context=incoming ; context
channel => 1;     channel number
signalling=fxo_ks; FXO signaling for FXS interfaces
group=2;          channel group
context=extensions; context
channel=> 2       channel number
```

Configuration options

Several options are available in the `chan_dahdi.conf` file. A description of all options would be boring and counterproductive; instead, we will focus on the main option groups available for easy understanding.

General options (channel independent)

These options work for any channel:

context: Defines the incoming context.

```
context=default
```

channel: Defines channel or channel range. Each channel definition will inherit options defined before the declaration. Channels can be identified individually or in the same line by comma separation. Ranges can be defined using “-”.

```
Channel=>1-15
```

```
Channel=>16
```

```
Channel=>17,18
```

group: Allows channels to be handled as a group. If you dial a group number instead of a channel number, the first channel available is used. If channels are phones, when you call a group, all phones will ring simultaneously. With commas, you can specify more than one group for the same channel.

```
group=1
```

```
group=3,5
```

language: Turns on the internationalization and configures a language. This feature will configure system messages for a specific language. English is the only language with complete prompts available through standard installation.

musiconhold: Selects music on hold class.

Caller ID options

There are many **callerid** options. Some can be disabled, although most are enabled by default.

usecallerid: Enables or disables the **callerid** transmission for the subsequent channels (Yes/No).

Note: If your system gets two rings before answering, try disabling this feature. It should answer immediately.

hidecallerid: Defines whether or not to hide the outgoing **callerid** (Yes/No).

callerid: Configures a **callerid** string for a specific channel. The caller can be configured with **asreceived**. This is mostly used in trunk interfaces to indicate the incoming **callerid**.

```
callerid = "Flavio Eduardo Gonçalves" <48 30258500>
```

callwaitingcallerid: Supports **callerid** during call waiting.

useincomingcalleridondahditransfer: Uses the incoming **callerid** in a transfer.

Call Waiting

Asterisk supports call waiting in FXS channels. The user will receive a waiting tone if someone tries the extension. To enable call waiting:

```
callwaiting=yes
```

To support `callerid` in call waiting:

```
callwaitingcallerid=yes
```

Audio quality options

Adjusting the echo cancellation is half technical, half art. These options adjust certain Asterisk parameters that affect audio quality in the DAHDI channels. They can help improve audio quality in analog interfaces.

The `fxotune` utility

The `fxotune` is a utility used to fine-tune certain parameters for FXO modules. This fine-tuning is required to adjust impedance mismatch caused by the hybrid. The utility has three operation modes:

- Detection (-i): detects and fixes the existing FXO channels and saves the configuration to `fxotune.conf`
- Dump mode (-d): generates the waveform files to `fxotune_dump.vals`
- Startup mode (-s): reads the file `fxotune.conf` and applies it to the FXO modules

It is important to understand that you will have to insert the instruction `fxotune -s` in the system load before starting Asterisk:

```
#modprobe dahdi
#modprobe wctdm
#fxotune-s
```

Echo cancellation

Most echo cancellation algorithms operate by generating multiple copies of the received signal, in which each one is delayed by a specific amount of time. The number of taps of the filter determines the size of the echo delay that needs to be cancelled. These delayed copies are then adjusted and subtracted from the received signal. The trick is to adjust only the delayed signal to remove the echo without using too many CPU cycles.

From the users' perspective, it is important to choose an appropriate echo cancellation algorithm. The default is MG2; however, recently two new possibilities have emerged. The commercially licensed High Performance Echo Cancellation (HPEC) from Digium and the open-source echo cancellation (OSLEC) developed by David Rowe. OSLEC has received significant attention lately. Check the official page for more information <http://www.rowetel.com/ucasterisk/oslec.html>. To change the echo cancellation algorithm, change the parameter `echo_can` to `/etc/dahdi/system.conf`.

For example:

```
echo_can=oslec
```

The echo cancellation in Asterisk is controlled by three parameters in the file `/etc/asterisk/channel-dahdi.conf`.

echocancel: Disables or enables echo cancellation. You should keep this feature enabled. It accepts “yes” or the number of taps.

Explanation: How does echo canceling work?

Most echo canceling algorithms operate by generating multiple copies of a received signal, with each being delayed by a small interval. This little flow is called a “tap”. The number of taps determines the echo delay that can be cancelled. These copies are delayed, adjusted, and subtracted from the original signal. The trick is to adjust the delayed signal exactly to what is necessary to remove the echo.

echocancelwhenbridged: Enables or disables the echo canceller during a pure TDM call. This is usually not necessary.

rxgain: Adjusts the audio reception gain to either increase or decrease reception volume (-100% to 100%).

txgain: Adjusts audio transmission gain to either increase or decrease the transmission volume (-100% to 100%).

For example:

```
echocancel=yes
echocancelwhenbridged=yes
txgain=-10%
rxgain=10%
```

Billing options

These options change how call information is recorded in the call detail records (CDR) database.

amaflags: Configures the AMA flags affecting the CDR categorization. It accepts the following values:

- billing
- documentation
- omit
- default

accountcode: Configures an account code for a specific channel. It can contain any alphanumeric value—usually the department or user name.

```
accountcode=finance
amaflags=billing
```

Call progress options

These items are used to acquire information about the progress of the call. In public interfaces, it may be useful to detect the call progress and determine if it was answered or busy. The busy detection is highly experimental and regulated by specific parameters.

```
busydetect=yes
busycount=4
```

```

busypattern=500,500
callprogress=yes
progzone=br

```

These parameters (above) specify whether the interface will try to detect the busy tone, how many tones will be used for successful detection, and what is the busy pattern. The busy detection is largely experimental, and some additional parameters can be changed in the [Makefile](#).

To detect the answer of a call, which is essential for precise billing, it is possible to use the polarity reversal to signal the exact answer time. This is important if you plan to charge for the call or just wish to have precise billing for comparison. Usually you have to contact the phone company to request this service.

```
answeronpolaritieswitch=yes
```

In some countries, it is possible to detect the hang up of the call using polarity reversal as well.

```
hanguponpolaritieswitch=yes
```

Options for phones

These options are used for phones connected to the FXS interfaces. All the functionalities delivered to analog phones connected directly to the DAHDI interfaces are controlled by Asterisk.

Adsi (Analog Display Services Interface): This is a set of telecom standards used by some telcos to offer services such as ticket buying.

cancallforward: Enables or disables call forwarding (*72 to enable and *73 to disable).

calleridcallwaiting: Enables `callerid` received during a call waiting indication (Yes/No).

immediate: In *immediate* mode, instead of providing a dial tone, the channel jumps immediately to the “s” extension in the defined context. This is used to create hotlines.

threewaycalling: Enables or disables three-way conferencing.

mailbox: Warns the user about available voicemail messages. It can be an audible sign or a visual indicator (if the telephone supports this feature). The argument is the mailbox number.

callgroup: Group phones to dial or to pick up.

pickupgroup: Group of phones for call pickup.

DAHDI channel format.

DAHDI channels use the following format in the dial plan:

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

<identifier>- Physical channel numeric identifier

[g] - Group identifier

[c] - Answer confirmation. A number is not considered until the callee press “#”

[r] - customized ringing

[cadence] Integer from 1 to 4

For example:

```
DAHDI/2 - channel 2
DAHDI/g1 - First available channel in group 1
```

Quiz

- Supervision signaling includes:
 - On-hook
 - Off-hook
 - Ringing
 - Dtmf
- Information signaling includes:
 - Dtmf
 - Dial tone
 - Invalid number
 - Ringback
 - Congestion
 - Busy
 - Pulse
- There are two types of analog interfaces available for Asterisk: FXS and FXO. Mark the correct answers.
 - FXS: Foreign Exchange Station can be connected directly to the company's PBX extension port.
 - FXO: Foreign Exchange Office can be connected to the public switched telephony network.
 - FXS: Foreign Exchange Station provides a dial tone and can be connected to a standard analog phone.
- To configure DAHDI hardware, you should first edit the _____ file.
 - /etc/dahdi/system.conf
 - /etc/asterisk/chan_dahdi.conf
 - /etc/asterisk/unicall.conf
 - serial.conf
- The DAHDI hardware is independent of Asterisk. In the chan_dahdi.conf, you configure Asterisk channels, not the hardware itself.
 - True

- B. False
6. When using a TDM400 with an ___ port, is necessary to connect the PC power source to the card using a specific connector (similar to the one used to power the hard disk).
- A. FXO
 - B. FXS
 - C. E+M
 - D. ISDN
7. Echo, pops, and noise in a DAHDI card are often related to the:
- A. Asterisk compilation
 - B. Cable problems
 - C. PCI Interrupt conflicts
 - D. Electromagnetic interference
8. When a card presents problems with echo, what you can do? (check all that apply)
- A. Change tx and rx gains
 - B. Change the echo cancellation algorithm (oslec, mg2)
 - C. Use hardware echo cancellation
 - D. Activate call progress detection
 - E. Invert the tip and ring
9. In some cases, when you want a precise billing using analog channels, it is important to activate a feature that allows the precise detection of the moment when the call answer occurred. To do this, you should activate _____ on Asterisk and at the phone company.
- A. Answer reversal
 - B. Billing reversal
 - C. Charge reversal
 - D. Polarity reversal
 - E. Dial tone generation
10. Caller ID identification on analog lines is country dependent. The most frequently used standard for North America is:
- A. v.23
 - B. dtmf
 - C. polarity reversal
 - D. battery reversal

5

Digital channels

Digital channels are extremely common, so you will need to learn how to implement these channels if you want to focus on large customers. When the number of channels is high—usually more than 8—it is fairly common to use digital interfaces such as T1/E1/J1. T1 is very common in the US, whereas E1 is common in Europe and J1 in Japan. These types of channels allow for a good density of circuits—24 per T1 channel and 30 for E1 channels. In Latin America, China, and Africa, it is common to use a type of channel associated signaling (CAS) known as MFC/R2. This chapter will examine how to implement MFC/R2 using the library OpenR2. In the US and Europe, Integrated Services Digital Networks (ISDN) PRI is the most common signaling. The chapter will also discuss ISDN Basic Rate Interface (BRI), which is very common in Europe in mid-range applications. All examples in the book concentrate on DAHDI channels. Some cards are implemented using proprietary channels, so please check with your manufacturer for further details on how to configure your specific card.

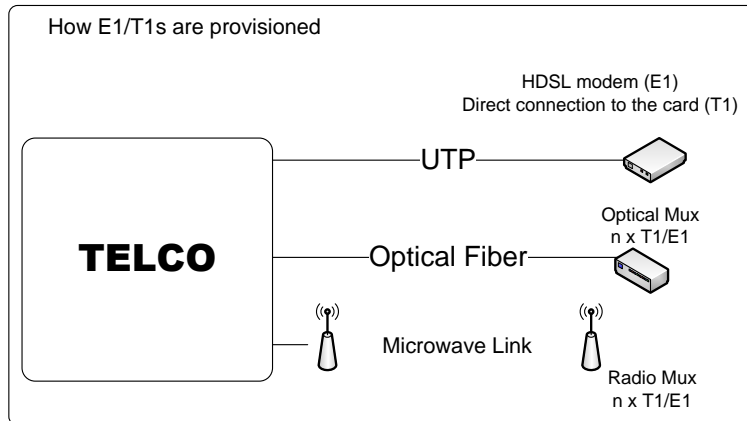
Objectives

By the end of this chapter you will be able to:

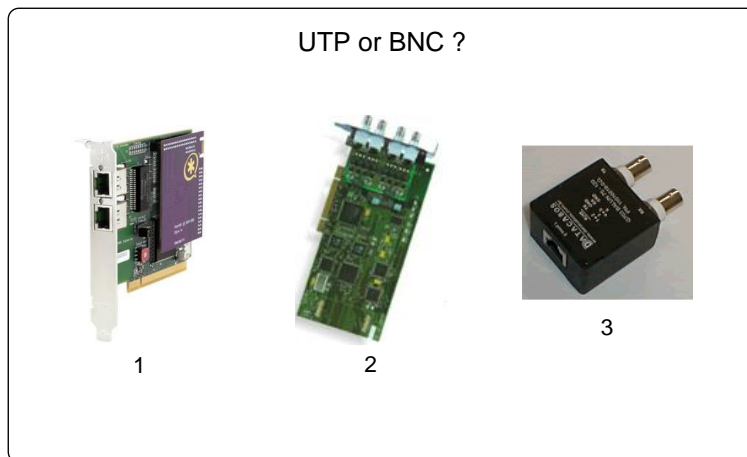
- Recognize the main terms used in digital telephony
- Differentiate CAS and CCS signaling
- Differentiate R2 and ISDN signaling
- Configure interfaces with ISDN signaling
- Configure interfaces with R2 signaling

E1/T1 digital lines

Digital lines E1/T1 are an option whenever you need to implement a large number of channels. A single E1 circuit is capable of 30 simultaneous calls, and you may have features such as direct inward dial (DID), Caller ID (caller identification), and advanced signaling. The E1/T1 line may arrive at your company in several ways using twisted pair, fiber, and microwaves, depending on your country.

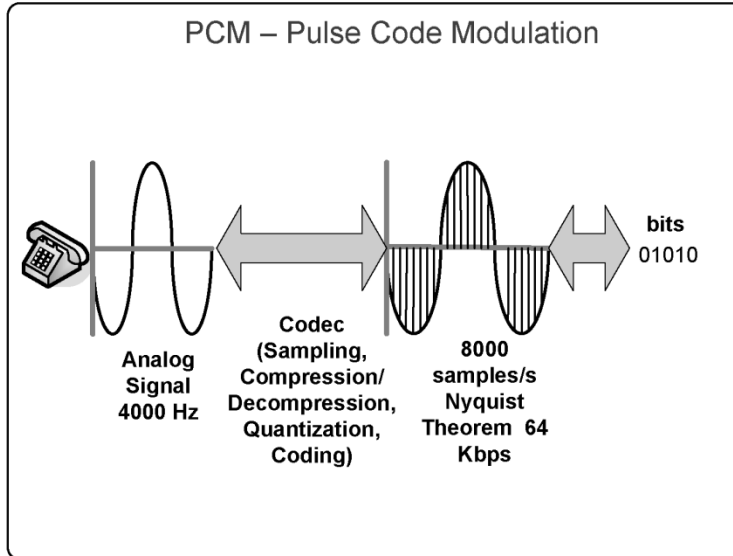


Digital lines are delivered to your company using UTP, fiber, or microwaves. Modems and multiplexors (MUX) are used to deliver the physical line. The connection to a T1 line is always based in an RJ45 connector. However, E1 lines may be provisioned as well using BNC. It is very important to know the type of connector you are going to receive in advance, mainly in E1 lines. Usually all the equipment up to the RJ45 is provided by the TELCO.



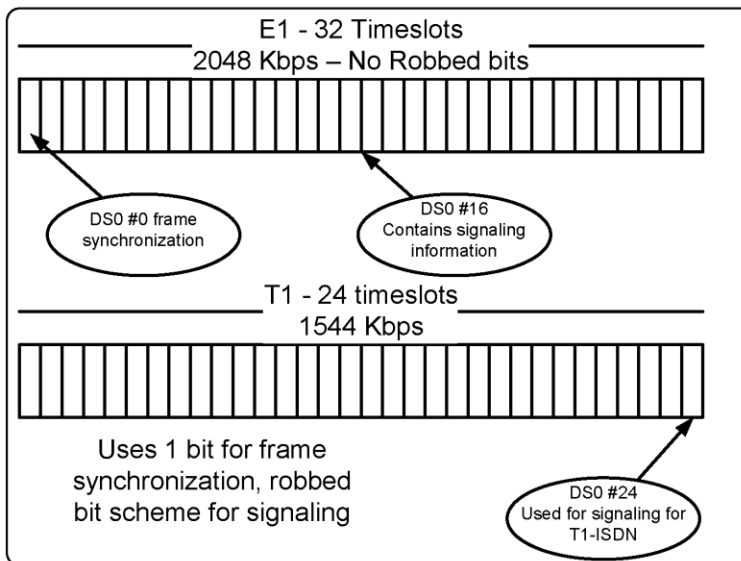
How is the voice converted to bits?

The analog signal is sampled 8,000 times per second to create a digital version of the analog voice. This encoding is known as pulse code modulation (PCM). In the US and Japan, the signal is encoded using μ law (in Asterisk, referred to as **u1aw**). In the rest of the world, the encoding is **a1aw**.



Time Division Multiplexing

Analog lines make sense when you need just a few channels. When using time division multiplexing (TDM), it is possible to stuff multiple channels into a single data connection. When you want a large number of circuits, the phone company will usually provide you with a digital trunk, which is a data circuit in which the voice is transported in a digital format using PCM. Each timeslot uses 64 Kbps of bandwidth to transport a single voice channel.



In the US, the most common digital trunk is T1, which has 24 available lines; in Europe and Latin America, E1 trunks have 30 lines. Some companies provide a fractional T1/E1 with fewer channels.

Robbed bit signaling

Sometimes a T1 trunk uses a robbed bit scheme where one bit is borrowed for signaling. On T1 trunks, the data/voice channel is transmitted with 56 Kbps on each timeslot. As you may observe, when you use the robbed bit, the T1 circuit does not lose two slots for synchronization and signaling.

T1/E1 Line code

T1s and E1s are actually data circuits and have a data coding that determines the way in which the bits are interpreted. For E1s, the most common line code is HDB3 for layer 1 and CCS for layer 2. The easiest way to know how your digital trunk is configured is to ask the TELCO about this information. You will need this information to configure the file `/etc/dahdi/system.conf`.

T1/E1 Signaling

It is important to understand that T1/E1 lines may be delivered using different kinds of signaling, such as:

- T1 with robbed bit signaling
- T1 with ISDN signaling
- E1 with MFC/R2 (CAS - Channel Associated Signaling)
- E1 with ISDN signaling

ISDN is often used in Europe and the US. It is a digital voice network, standardized by the International Telecommunications Union (ITU) in 1984. ISDN provides two kinds of channels:

- Bearer channels
 - Voice
 - Data
- Data channels
 - Out of band signaling
 - LAPD signaling
 - Q.931

Usually, an ISDN line is provided using two physical means:

- Basic rate interface (BRI)
 - Known as 2B+D
 - Two bearer (64K) channels and a data (16K) channel
 - Uses a pair of copper wires with 148Kbps.
- Primary rate interface (PRI)
 - Delivered using a T1/E1 trunk
 - 23B+D for T1s

- 30B+D for E1s

Sometimes, E1 circuits use a CAS signaling scheme called MFC/R2, which was defined by the ITU as a standard known as Q.421/Q441. This is frequently found in Latin America and Asia. Several telephony companies in these countries use customized variants of MFC/R2. Hence, you will need to know the correct country variation in order to make it work.

ISDN BRI

Channels using ISDN BRI signalling are very popular in Europe. Most ISDN BRI cards for Asterisk supports an S/T interface with NT and TE capabilities. The TE (terminal) connection is the one used to connect to the TELCO or to other PBXs configured as network termination (NT). The NT is used to connect phones and PBXs configured as TE. ISDN BRI provides two data/voice channels and one signalling channel. ISDN BRI cards are available from several vendors of interface cards for Asterisk.

Choosing a telephony card for your Asterisk server

There are several manufacturers for digital cards compatible with Asterisk. The choice of a card depends on some of the following factors:

Data bus

There are several types of bus on your PC. It is very important that you have the right card for your server. The following overview outlines the most frequently used cards:

- 32 Bits PCI 5V found in most computers, including desktops
 - Digium TE405, TE407, TE205, TE207, TE120, TE122, B410, TDM2400, TDM800, TDM410, and TC400
 - Sangoma A101, A102, and A104
- 32/64 bits PCI 3.3V, basically found in servers
 - Digium TE410, TE412, TE210, TE212, TE120, TE122, B410, TDM2400, TDM800, TDM410, and TC400
- PCI Express found on desktops and servers
 - Digium TE420, TE220, TE121, AEX2400, and AEX800
 - Sangoma A101, A102, and A104
- MiniPCI found on embedded systems
 - OpenVOX A100M(FXO), B100M(ISDN BRI), B200M(ISDN BRI), and B400M(ISDN BRI)
- USB 2.0 found in most modern PCs. Solutions based on USB allow a great density of analog and digital channels. This bus supports 480 Mbps, and each voice channel occupies 64 Kbps. When using USB hubs, it is possible to get densities up to a thousand analog ports in a single port.

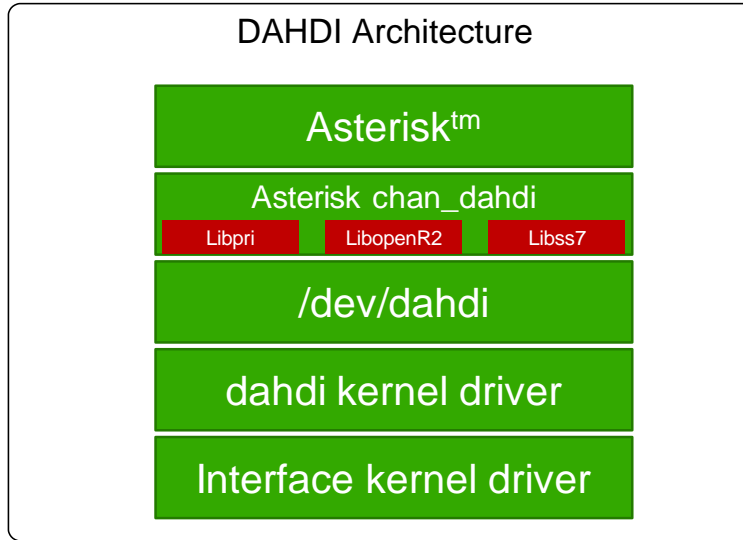
- Xorcom Atribank (FXS, FXO, E1-ISDN, E1-R2)
- Ethernet. The biggest advantage of Ethernet is to allow the card to be connected by more than one server. High availability solutions are usually the core application for these devices. The strength of this solution is the use of servers without free PCI slots or blade servers.
 - Redfone FoneBridge (up to four E1 circuits)

Using hardware echo cancellation

Hardware echo cancellation reduces the load in the host CPU. For cards with more than a single E1 interface, hardware echo cancellation can help alleviate your processor. New enhanced software echo cancellers such as the OSLEC are reducing the need for a hardware echo canceller. To choose between hardware and software echo cancellers, you should consider the amount of processing power available in your server and the number of E1 circuits. An echo cancellation process may use up to nine MIPS (millions of instructions per second) per voice channel with 128 taps of amplitude using OSLEC (Reference: Xorcom Ltd.). If you consider 1 CPU cycle per each instruction (which is not always correct based on the processor and software implementation itself), we are speaking of 1.080 Ghz for four E1s.

Type of signaling

Selecting the type of signaling (e.g., T1 CAS, T1 PRI, E1 CAS R2, or E1 CAS ISDN) is not an easy task. It really depends on what you have available in your area and at what price. Common Channel Signaling (CCS) is often better than channel associated signaling (CAS). However, it is often not available. In the US, you can usually choose, as most TELCOS offer T1 CAS for regular users and T1 PRI for advanced users (e.g., call centers). In Latin America, E1 CAS R2 is prevalent, but ISDN PRI is available in some cities.



Implementing R2 is necessary for installing a library known as OpenR2 (www.libopenr2.org), developed by Moises Silva, and to patch Asterisk before the installation—a simple procedure shown later in this chapter. The library has passed several tests and is in production in several of our customers. ISDN is, in my opinion, always the best choice, if available. Some providers can have access to signaling system 7 (SS7), which is a CCS signaling available between phone companies. Proprietary and open source solutions are available for SS7. Library libss7 is used to support SS7 on Asterisk.

Zaptel and DAHDI

Recently, because of a dispute over the trademark Zaptel™, Digium has changed the name of its drivers. In this version of the book, we use the new DAHDI drivers because the old ZAPTEL drivers will no longer be maintained. The file UPGRADE.txt in the source code details the differences.

Asterisk telephony channels setup

Configuring a telephony interface card involves several necessary steps. In this chapter, we will show three of the most common scenarios:

- Digital connection using ISDN PRI
- Digital connection using ISDN BRI
- Digital connection using MFC/R2

There are two ways to configure DAHDI channels. The first one is to configure it manually with full control of all parameters. The second way is to use the utility `dahdi_genconf` to detect and configure the cards.

Automatic detection and configuration

Thanks to the DAHDI development team, we now have automatic detection and configuration of the cards.

Step 1: To generate the configuration automatically, use the utility `dahdi_genconf`, which will detect the card and generate the files `/etc/dahdi/system.conf` and `dahdi-channels.conf`.

`dahdi_genconf`

Step 2: In the last line of the file `chan_dahdi.conf`, include the file `dahdi-channels.conf`

```
#include dahdi_channels.conf
```

Step 3: Comment on all the unused modules in the file `modules` or simply use:

`dahdi_genconf modules`

Manual configuration

Another option is to configure the interfaces manually. Below are some examples of the configuration for DAHDI channels.

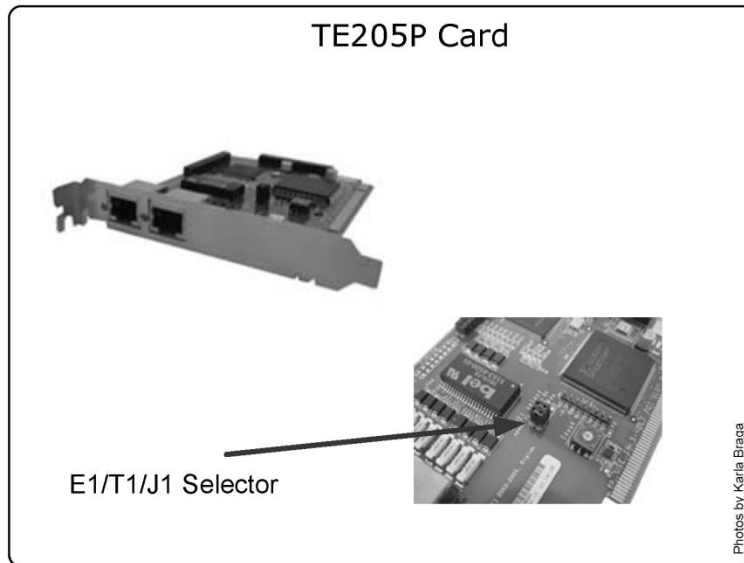
Example #1 – Two T1/ E1 channels using ISDN

Required steps:

- TE205P or TE210P installation
- `/etc/dahdi/system.conf` file configuration
- dahdi driver loading
- `dahdi_test` utility
- `dahdi_cfg` utility
- `chan_dahdi.conf` file configuration
- Asterisk load and testing

Step 1: TE205P installation

Before installing TE205P, it is important to understand the differences between the TE205P and TE210P cards. The TE210P card uses a 64-bit bus powered by 3.3 volts found almost only in the server's motherboards. Be careful if you specify this interface card; make sure your hardware supports a 64-bit, 3.3V bus. The TE205P card uses a 5V PCI, which is often found in desktop computers.



We have chosen the TE205P interface card with two spans for this example because it is easier to reduce it to one-span card or to expand it to the four-span card.

Step 2: `/etc/dahdi/system.conf` configuration file

The configuration of TDM digital cards is a bit different from the configuration of their analog counterparts. First, we will need to configure the board spans and then the channels. Spans are numbered sequentially depending on the recognizing order of the cards. In other words, if you have more than one interface card, it is hard to know what span belongs to each one.

Use `dahdi_hardware` to check which hardware is installed on each span.

Example #1 (2xT1 PRI)

```
span=1,1,0,esf,b8zs
span=2,0,0,esf,b8zs
bchan=1-23
dchan=24
bchan=25-47
dchan=48
defaultzone=us
loadzone=us
```

Example #2 (2xE1 PRI)

```
span=1,1,0,ccs,hdb3,crc4 # not always necessary, consult Telco.
span=2,0,0,ccs,hdb3,crc4
bchan=1-15, 17-31
dchan=16
bchan=33-47, 49-63
dchan=48
defaultzone=br
```



```
loadzone=br
```

Example #3 (4xBRI)

```
loadzone=de
defaultzone=de
span=1,1,0,ccs,ami
bchan=1,2
hardhdlc=3
span=2,0,0,ccs,ami
bchan=4,5
hardhdlc=6
span=3,0,0,ccs,ami
bchan=7,8
hardhdlc=9
span=4,0,0,ccs,ami
bchan=10,11
hardhdlc=12
```

Step 3: Loading kernel drivers

Check which driver you need to install using [dahdi_hardware](#).

dahdi_hardware

```
pci:0000:04:02.0      wcte2xxp      e159:0001 Digium wilcard TE205P T1/E1 Board
```

To load use:

```
modprobe dahdi
modprobe wct2xxp
```

Step 4: Using [dahdi_test](#), check the missing interrupts

You may verify the number of interrupt misses using the [dahdi_test](#) utility compiled with the DAHDI cards. A number below 99.987% indicates possible problems. You will find [dahdi_test](#) in [/usr/sbin](#).

#!/dahdi_test

```
Opened pseudo zap interface, measuring accuracy...
99.987793% 100.000000% 100.000000% 100.000000% 100.000000% 100.000000%
100.000000%
100.000000% 100.000000% 100.000000% 100.000000% 100.000000% 100.000000%
100.000000% 100.000000%
100.000000% 100.000000% 100.000000% 100.000000% 99.987793% 100.000000%
100.000000% 100.000000%
100.000000% 100.000000% 100.000000%
--- Results after 26 passes ---
Best: 100.000000 -- worst: 99.987793 -- Average: 99.999061
```

Step 5: Using the [dahdi_cfg](#) utility

This is the correct output for [dahdi_cfg](#) for one fractional E1 (15 ports) span and two FXO ports.

#!/dahdi_cfg -vvvv

```
Dahdi configuration
```

```

=====
SPAN 1: CCS/HDB3 Build-out: 0 db (CSU)/0-133 feet (DSX-1)
Channel map:
Channel 01: Clear channel (Default) (Slaves: 01)
Channel 02: Clear channel (Default) (Slaves: 02)
Channel 03: Clear channel (Default) (Slaves: 03)
Channel 04: Clear channel (Default) (Slaves: 04)
Channel 05: Clear channel (Default) (Slaves: 05)
Channel 06: Clear channel (Default) (Slaves: 06)
Channel 07: Clear channel (Default) (Slaves: 07)
Channel 08: Clear channel (Default) (Slaves: 08)
Channel 09: Clear channel (Default) (Slaves: 09)
Channel 10: Clear channel (Default) (Slaves: 10)
Channel 11: Clear channel (Default) (Slaves: 11)
Channel 12: Clear channel (Default) (Slaves: 12)
Channel 13: Clear channel (Default) (Slaves: 13)
Channel 14: Clear channel (Default) (Slaves: 14)
Channel 15: Clear channel (Default) (Slaves: 15)
Channel 16: D-channel (Default) (Slaves: 16)
16 channels configured.

```

Step 6: Configuration of DAHDI into the file `/etc/asterisk/chan_dahdi.conf`

Example #1 (2xT1)

```

callerid="John Doe"<(555)555-1111>
switchtype=national
signalling =pri_cpe
context=from-pstn
group = 1
channel => 1-23
group =2
channel => 25-47

```

Example #2 (2xE1)

```

callerid="Flavio Eduardo" <4830258580>
switchtype=euroisdn
signalling = pri_cpe
group = 1
channel => 1-15;17-31
group =2
channel => 32-46;48-62

```

Example #3 (4xBRI)

```

signaling=bri_cpe
switchtype=euroisdn
group=1
context=from-pstn
channel=>1,2,4,5,7,8,10,11

```

Use `signaling=bri_cpe_ptmp` for point to multipoint BRI. Currently, BRI point to multipoint is not supported in NT mode.

Loading the kernel drivers

After configuring the drivers, you may simply restart the server. If you have installed DAHDI with `make config`, you won't need to do anything extra. The kernel driver will be automatically loaded and configured. However, sometimes it is useful to load and unload the drivers manually.

Example:

```
modprobe wct11xp
dahdi_cfg -vvvvv
```

The first command loads the driver and the second, `dahdi_cfg`, applies the configuration to the kernel driver.

Troubleshooting

Sometimes things don't work the first time. Let's check some resources for troubleshooting DAHDI.

Step 1: Check if the card is being recognized by the operation system. Digium cards are usually recognized as the ISDN modem.

`lspci -v`

```
00:00.0 Host bridge: Intel Corporation E7230/3000/3010 Memory Controller Hub
00:01.0 PCI bridge: Intel Corporation E7230/3000/3010 PCI Express Root Port
00:1c.0 PCI bridge: Intel Corporation 82801G (ICH7 Family) PCI Express Port 1 (rev 01)
00:1c.4 PCI bridge: Intel Corporation 82801GR/GH/GHM (ICH7 Family) PCI Express Port 5 (rev 01)
00:1c.5 PCI bridge: Intel Corporation 82801GR/GH/GHM (ICH7 Family) PCI Express Port 6 (rev 01)
00:1d.0 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #1 (rev 01)
00:1d.1 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #2 (rev 01)
00:1d.2 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #3 (rev 01)
00:1d.7 USB Controller: Intel Corporation 82801G (ICH7 Family) USB2 EHCI Controller (rev 01)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev e1)
00:1f.0 ISA bridge: Intel Corporation 82801GB/GR (ICH7 Family) LPC Interface Bridge (rev 01)
00:1f.1 IDE interface: Intel Corporation 82801G (ICH7 Family) IDE Controller (rev 01)
00:1f.2 IDE interface: Intel Corporation 82801GB/GR/GH (ICH7 Family) SATA IDE Controller (rev 01)
00:1f.3 SMBus: Intel Corporation 82801G (ICH7 Family) SMBus Controller (rev 01)
01:00.0 PCI bridge: Intel Corporation 6702PXH PCI Express-to-PCI Bridge A (rev 09)
01:00.1 PIC: Intel Corporation 6700/6702PXH I/OxAPIC Interrupt Controller A (rev 09)
02:08.0 SCSI storage controller: LSI Logic / Symbios Logic SAS1068 PCI-X Fusion-MPT SAS (rev 01)
03:00.0 PCI bridge: Intel Corporation 6702PXH PCI Express-to-PCI Bridge A (rev 09)
04:02.0 Network controller: Tiger Jet Network Inc. Tiger3XX Modem/ISDN interface
05:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5721 Gig. Eth.PCI Express (rev 11)
07:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8139/8139C/8139C+ (rev 10)
```

```
07:05.0 VGA compatible controller: ATI Technologies Inc ES1000 (rev 02)
```

Step 2: Check if the kernel driver is loading correctly using:

```
modprobe wct11xp
dmesg
```

```
TE110P: Setting up global serial parameters for E1 FALC V1.2
TE110P: Successfully initialized serial bus for card
TE110P: Span configured for CAS/HDB3
Calling startup (flags is 4099)
Found a Wildcard: Digium Wildcard TE110P T1/E1
TE110P: Span configured for CCS/HDB3/CRC4

Calling startup (flags is 4099)
dahdi: Registered tone zone 0 (United States / North America)
wctelxxp: Setting yellow alarm
```

Step 3: Verify the status of alarms related to the physical layer of the connection.

To verify the physical layer of the E1 connection, you may use the following console command.

```
dahdi show status
```

The alarms indicate problems with the port:

Red Alarm: Cannot maintain synchronization with the remote switch. This is usually a physical problem, such as line code or framing mismatch.

Yellow alarm: Signals that the remote switch is in the red alarm. This indicates that the remote switch is not receiving your transmissions.

Blue Alarm: Receives all unframed 1s on all timeslots; `dahdi_tool` currently does not detect a blue alarm.

Loopback: The port is either in local or remote loopback

```
vtsvoffice*CLI> dahdi show status
Description                               Alarms   IRQ      bpviol   CRC4
Digium Wildcard E100P E1/PRA Card 0      OK       0        0        0
Wildcard X100P Board 1                   OK       0        0        0
Wildcard X100P Board 2                   RED      0        0        0
```

Step 4: To detect problems with DAHDI on the Asterisk server, first check if the channels are being recognized using:

```
CLI dahdi show channels
pabxip01*CLI> dahdi show channels
  Chan Extension Context      Language MOH Interpret
pseudo          default
  1             from-pstn default
  2             from-pstn default
  3             from-pstn default
  4             from-pstn default
  5             from-pstn default
  6             from-pstn default
  7             from-pstn default
  8             from-pstn default
  9             from-pstn default
 10            from-pstn default
```

11	from-pstn	default
12	from-pstn	default
13	from-pstn	default
14	from-pstn	default
15	from-pstn	default
17	from-pstn	default
18	from-pstn	default
19	from-pstn	default
20	from-pstn	default
21	from-pstn	default
22	from-pstn	default
23	from-pstn	default
24	from-pstn	default
25	from-pstn	default
26	from-pstn	default
27	from-pstn	default
28	from-pstn	default
29	from-pstn	default
30	2171 from-pstn	default
31	2171 from-pstn	default

Step 5: Check the status of the ISDN layer 3, also known as q.931.

You can check if the ISDN layer 3 is up using:

pri show span:

```
vtsvoffice*CLI> pri show span 1
Primary D-channel: 16
Status: Provisioned, Up, Active
Switchtype: EuroISDN
Type: CPE
Window Length: 0/7
Sentrej: 0
SolicitFbit: 0
Retrans: 0
Busy: 0
Overlap Dial: 0
T200 Timer: 1000
T203 Timer: 10000
T305 Timer: 30000
T308 Timer: 4000
T313 Timer: 4000
N200 Counter: 3
```

Check a specific channel.

dahdi show channel x:

```
vtsvoffice*CLI> dahdi show channel 1
Channel: 1*CLI>
File Descriptor: 21
Span: 1
Extension:
Dialing: no
Context: entrada
Caller ID: 4832341689
Calling TON: 33
Caller ID name:
Destroy: 0
InAlarm: 0
```

```

Signalling Type: PRI Signalling
Radio: 0
Owner: <None>
Real: <None>
Callwait: <None>
Threeway: <None>
Confno: -1
Propagated Conference: -1
Real in conference: 0
DSP: no
Relax DTMF: no
Dialing/CallwaitCAS: 0/0
Default law: alaw

```

debug pri span x: If after everything you still have problems, start debugging the pri span. This command enables a detailed debugging of ISDN calls. It is an important command when you think that something is not correct. You can detect digits being misdialed and other problems. Below we present the example of a debugging output for a successful call. Refer to this example if you need to compare an unsuccessful call to one without problems. One tip is using core `set verbose=0` to receive just the ISDN q.931 messages.

```

-- Making new call for cr 32833
> Protocol Discriminator: Q.931 (8) len=57
> Call Ref: len= 2 (reference 65/0x41) (Originator)
> Message type: SETUP (5)
> [04 03 80 90 a3]
> Bearer Capability (len= 5) [ Ext: 1 Q.931 Std: 0 Info transfer capability: Speech (0)
>                               Ext: 1 Trans mode/rate: 64kbps, circuit-mode (16)
>                               Ext: 1 User information layer 1: A-Law (35)
> [18 03 a9 83 81]
> Channel ID (len= 5) [ Ext: 1 IntID: Implicit, PRI Spare: 0, Exclusive Dchan: 0
>                               ChanSel: Reserved
>                               Ext: 1 Coding: 0 Number Specified Channel Type: 3
>                               Ext: 1 Channel: 1 ]
> [28 0e 46 6c 61 76 69 6f 20 45 64 75 61 72 64 6f]
> Display (len=14) @h@[ Flavio Eduardo ]
> [6c 0c 21 80 34 38 33 30 32 35 38 35 39 30]
> Calling Number (len=14) [ Ext: 0 TON: National Number (2) NPI: ISDN/Telephony Numbering
Plan (E.164/E.163) (1)
>                               Presentation: Presentation permitted, user number not screened
(0) '4830258590' ]
> [70 09 a1 33 32 32 34 38 35 38 30]
> Called Number (len=11) [ Ext: 1 TON: National Number (2) NPI: ISDN/Telephony Numbering
Plan (E.164/E.163) (1) '32248580' ]
> [a1]fice*CLI>
> Sending Complete (len= 1)
< Protocol Discriminator: Q.931 (8) len=10
< Call Ref: len= 2 (reference 65/0x41) (Terminator)
< Message type: CALL PROCEEDING (2)
< [18 03 a9 83 81]
< Channel ID (len= 5) [ Ext: 1 IntID: Implicit, PRI Spare: 0, Exclusive Dchan: 0
<                               ChanSel: Reserved
<                               Ext: 1 Coding: 0 Number Specified Channel Type: 3
<                               Ext: 1 Channel: 1 ]
-- Processing IE 24 (cs0, Channel Identification)
< Protocol Discriminator: Q.931 (8) len=9
< Call Ref: len= 2 (reference 65/0x41) (Terminator)
< Message type: ALERTING (1)
< [1e 02 84 88]

```

```

< Progress Indicator (len= 4) [ Ext: 1 Coding: CCITT (ITU) standard (0) 0: 0 Location:
Public network serving the remote user (4)
<
      Ext: 1 Progress Description: Inband information or
appropriate pattern now available. (8) ]
-- Processing IE 30 (cs0, Progress Indicator)
< Protocol Discriminator: Q.931 (8) len=64
< Call Ref: len= 2 (reference 5720/0x1658) (Originator)
< Message type: SETUP (5)
< [04 03 80 90 a3]
< Bearer Capability (len= 5) [ Ext: 1 Q.931 Std: 0 Info transfer capability: Speech (0)
<
      Ext: 1 Trans mode/rate: 64kbps, circuit-mode (16)
<
      Ext: 1 User information layer 1: A-Law (35)
< [18 03 a1 83 82]
< Channel ID (len= 5) [ Ext: 1 IntID: Implicit, PRI Spare: 0, Preferred Dchan: 0
<
      ChanSel: Reserved
<
      Ext: 1 Coding: 0 Number Specified Channel Type: 3
<
      Ext: 1 Channel: 2 ]
< [1c 15 91 a1 12 02 01 bc 02 01 0f 30 0a 02 01 01 0a 01 00 a1 02 82 00]
< Facility (len=23, codeset=0) [ 0x91, 0xa1, 0x12, 0x02, 0x01, 0xbc, 0x02, 0x01, 0x0f, '0',
0x0a, 0x02, 0x01, 0x01, 0x0a, 0x01, 0x00, 0xa1, 0x02, 0x82, 0x00 ]
< [1e 02 82 83]
< Progress Indicator (len= 4) [ Ext: 1 Coding: CCITT (ITU) standard (0) 0: 0 Location:
Public network serving the local user (2)
<
      Ext: 1 Progress Description: Calling equipment is non-ISDN.
(3) ]
< [6c 0c 21 83 34 38 33 32 32 34 38 35 38 30]
< Calling Number (len=14) [ Ext: 0 TON: National Number (2) NPI: ISDN/Telephony Numbering
Plan (E.164/E.163) (1)
<
      Presentation: Presentation allowed of network provided number (3)
'4832248580' ]
< [70 05 c1 38 35 38 30]
< Called Number (len= 7) [ Ext: 1 TON: Subscriber Number (4) NPI: ISDN/Telephony Numbering
Plan (E.164/E.163) (1) '8580' ]
< [a1]
< Sending Complete (len= 1)
-- Making new call for cr 5720
-- Processing Q.931 Call Setup
-- Processing IE 4 (cs0, Bearer Capability)
-- Processing IE 24 (cs0, Channel Identification)
-- Processing IE 28 (cs0, Facility)
Handle Q.932 ROSE Invoke component
-- Processing IE 30 (cs0, Progress Indicator)
-- Processing IE 108 (cs0, Calling Party Number)
-- Processing IE 112 (cs0, Called Party Number)
-- Processing IE 161 (cs0, Sending Complete)
> Protocol Discriminator: Q.931 (8) len=10
> Call Ref: len= 2 (reference 5720/0x1658) (Terminator)
> Message type: CALL PROCEEDING (2)
> [18 03 a9 83 82]
> Channel ID (len= 5) [ Ext: 1 IntID: Implicit, PRI Spare: 0, Exclusive Dchan: 0
>
      ChanSel: Reserved
>
      Ext: 1 Coding: 0 Number Specified Channel Type: 3
>
      Ext: 1 Channel: 2 ]
> Protocol Discriminator: Q.931 (8) len=14
> Call Ref: len= 2 (reference 5720/0x1658) (Terminator)
> Message type: CONNECT (7)
> [18 03 a9 83 82]
> Channel ID (len= 5) [ Ext: 1 IntID: Implicit, PRI Spare: 0, Exclusive Dchan: 0
>
      ChanSel: Reserved
>
      Ext: 1 Coding: 0 Number Specified Channel Type: 3
>
      Ext: 1 Channel: 2 ]

```

```

> [1e 02 81 82]
> Progress Indicator (len= 4) [ Ext: 1 Coding: CCITT (ITU) standard (0) 0: 0 Location:
Private network serving the local user (1)
> Ext: 1 Progress Description: Called equipment is non-ISDN.
(2) ]
< Protocol Discriminator: Q.931 (8) len=5
< Call Ref: len= 2 (reference 5720/0x1658) (Originator)
< Message type: CONNECT ACKNOWLEDGE (15)
< Protocol Discriminator: Q.931 (8) len=9
< Call Ref: len= 2 (reference 65/0x41) (Terminator)
< Message type: PROGRESS (3)
< [1e 02 84 82]
< Progress Indicator (len= 4) [ Ext: 1 Coding: CCITT (ITU) standard (0) 0: 0 Location:
Public network serving the remote user (4)
< Ext: 1 Progress Description: Called equipment is non-ISDN.
(2) ]
-- Processing IE 30 (cs0, Progress Indicator)
< Protocol Discriminator: Q.931 (8) len=5
< Call Ref: len= 2 (reference 65/0x41) (Terminator)
< Message type: CONNECT (7)
> Protocol Discriminator: Q.931 (8) len=5
> Call Ref: len= 2 (reference 65/0x41) (Originator)
> Message type: CONNECT ACKNOWLEDGE (15)
NEW_HANGUP DEBUG: Calling q931_hangup, ourstate Active, peerstate Connect Request
> Protocol Discriminator: Q.931 (8) len=9
> Call Ref: len= 2 (reference 65/0x41) (Originator)
> Message type: DISCONNECT (69)
> [08 02 81 90]
> Cause (len= 4) [ Ext: 1 Coding: CCITT (ITU) standard (0) 0: 0 Location: Private network
serving the local user (1)
> Ext: 1 Cause: Unknown (16), class = Normal Event (1) ]
< Protocol Discriminator: Q.931 (8) len=5
< Call Ref: len= 2 (reference 65/0x41) (Terminator)
< Message type: RELEASE (77)
NEW_HANGUP DEBUG: Calling q931_hangup, ourstate Null, peerstate Release Request
> Protocol Discriminator: Q.931 (8) len=9
> Call Ref: len= 2 (reference 65/0x41) (Originator)
> Message type: RELEASE COMPLETE (90)
> [08 02 81 90]
> Cause (len= 4) [ Ext: 1 Coding: CCITT (ITU) standard (0) 0: 0 Location: Private network
serving the local user (1)
> Ext: 1 Cause: Unknown (16), class = Normal Event (1) ]
NEW_HANGUP DEBUG: Calling q931_hangup, ourstate Null, peerstate Null
NEW_HANGUP DEBUG: Destroying the call, ourstate Null, peerstate Null
< Protocol Discriminator: Q.931 (8) len=9
< Call Ref: len= 2 (reference 5720/0x1658) (Originator)
< Message type: DISCONNECT (69)
< [08 02 82 90]
< Cause (len= 4) [ Ext: 1 Coding: CCITT (ITU) standard (0) 0: 0 Location: Public network
serving the local user (2)
< Ext: 1 Cause: Unknown (16), class = Normal Event (1) ]
-- Processing IE 8 (cs0, Cause)
NEW_HANGUP DEBUG: Calling q931_hangup, ourstate Disconnect Indication, peerstate Disconnect
Request
> Protocol Discriminator: Q.931 (8) len=9
> Call Ref: len= 2 (reference 5720/0x1658) (Terminator)
> Message type: RELEASE (77)
> [08 02 81 90]
> Cause (len= 4) [ Ext: 1 Coding: CCITT (ITU) standard (0) 0: 0 Location: Private network
serving the local user (1)

```



```
>                               Ext: 1 Cause: Unknown (16), class = Normal Event (1) ]
< Protocol Discriminator: Q.931 (8) len=5
< Call Ref: len= 2 (reference 5720/0x1658) (Originator)
< Message type: RELEASE COMPLETE (90)
NEW_HANGUP DEBUG: Calling q931_hangup, ourstate Null, peerstate Null
NEW_HANGUP DEBUG: Destroying the call, ourstate Null, peerstate Null
```

Configuration options in chan_dahdi.conf

Several options are available in the file `chan_dahdi.conf`. A description of all options would be boring and counterproductive. Here, we will detail the main option groups available to provide a better understanding.

General options (channel independent)

context: Defines the incoming context.

```
context=default
```

channel: Defines channel or channel range. Each channel definition will inherit options defined before the declaration. Channels can be identified individually or in the same line with comma separation. Ranges can be defined using “-”.

```
Channel=>1-15
Channel=>16
Channel=>17,18
```

group: Allows channels to be treated as a group. If you dial a group number instead of a channel number, the first channel available is used. If channels are phones, when you call a group, all phones will ring simultaneously. Using commas, you can specify more than one group for the same channel.

```
group=1
group=3,5
```

language: Turns on the internationalization and configures a language. This feature will configure system messages for a specific language. English is the only language with complete prompts available from the standard installation.

musiconhold: Select music on hold class.

ISDN options

switchtype: Is dependent on the PBX or switch used. In Europe and Latin America, EuroISDN is common.

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

- **Qsig: Q.SIG**
switchtype = EuroISDN

pridialplan: Required for some switches that need a dial plan specification. This option is ignored by many switches. The valid options are private, national, international, and unknown.

pridialplan = unknown

prilocaldialplan: Necessary for some switches, usually unknown.

prilocaldialplan = unknown

overlapdial: Overlap dialing is used when you pass digits after the connection is established. You can use block mode numbering (**overlapdial=no**) or digit mode (**overlapdial=yes**). Block mode is often used by operators.

signaling: Configures the signaling type for the subsequent channels. These parameters should correspond to those in the **chan_dahdi.conf** file. Correct choices are based on the available channel. For ISDN you might choose five options:

- **pri_cpe:** Used when the device is a CPE, sometimes referred to as client, user, or slave. This is the simplest and most used form of signaling. Sometimes, when you try to connect to a private PBX, the PBX has commonly been configured as a CPE as well. In this case, use **pri_net** signaling in Asterisk.
- **pri_net:** Used when Asterisk is connected to a private PBX configured as a CPE. The signaling is often referred to as host, master, or network.
- **bri_cpe:** Used when Asterisk is connected as a CPE to a ISDN BRI trunk
- **bri_net:** Used when Asterisk is connected to an ISDN phone or PBX configured as a terminal (TE).
- **bri_cpe_ptmp:** Sames as bri_cpe, but in a point-to-multipoint architecture.

CallerID options

Many Caller ID options are available. Some can be disabled, although most are enabled by default.

usecallerid: Enables or disables the Caller ID transmission for the subsequent channels (Yes/No).

Note: If your system requires two rings before answering, try disabling this feature so that it will answer immediately.

hidecallerid: Hides the Caller ID (Yes/No).

calleridcallwaiting: Enables receiving Caller ID during a call waiting indication (Yes/No).

callerid: Configures a Caller ID string for a specific channel. The caller can be configured with “asreceived” in trunk interfaces to pass the Caller ID forward.

callerid = "Flavio Eduardo Gonçalves" <48 30258500>

Note: Most TELCOs mandate that you configure your correct caller ID. If you do not pass the right caller ID, you shouldn't be able to dial out over the TELCO. On the other hand, you will be able to receive calls even without configuring the caller ID.

Audio quality options

These options adjust certain Asterisk parameters that affect audio quality in DAHDI channels.

echocancel: Disable or enable echo cancellation. You should keep this feature enabled. It accepts “yes” or the number of taps.

Explanation: How does echo canceling work?

Most echo canceling algorithms operate by generating multiple copies of a received signal, with each being delayed by a small interval. This little flow is named “tap”. The number of taps determines the echo delay that can be cancelled. These copies are delayed, adjusted, and subtracted from the original signal. The trick is to adjust the delayed signal exactly to what is necessary to remove the echo.

echocancelwhenbridged: Enables or disables the echo canceller during a pure TDM call. This is usually not required.

rxgain: Adjusts the audio reception gain to either increase or decrease reception volume (-100% to 100%).

txgain: Adjusts audio transmission gain to either increase or decrease the transmission volume (-100% to 100%).

Example:

```
echocancel=yes
echocancelwhenbridged=yes
txgain=-10%
rxgain=10%
```

Billing options

These options change the way in which call information is recorded in the call detail records (CDR) database.

amaflags: Affects the categorization of CDR. It accepts these values:

- billing
- documentation
- omit
- default

accountcode: It configures an account code for a specific channel. It can contain any alphanumeric value, usually the department or user name.

```
accountcode=finance
```

```
amaflags=billing
```

MFC/R2 configuration

MFC/R2 is used in several countries in Latin America, China, and Africa as well as some European countries. ISDN is superior and preferred if available in your area.

Understanding the problem

The card used to signal MFC/R2 is the same used to signal ISDN. It's possible to use MFC/R2 on DAHDI channels using the library called libopenR2 (www.libopenr2.com). This library was not part of versions of Asterisk prior to 1.6.2; to install it, patch the Asterisk code before compiling—an easy procedure shown in this section.

Understanding the MFC/R2 protocol

The MFC/R2 protocol combines in-band and out-of-band signaling. Address signaling is forwarded in-band using a set of tones while channel information is transmitted over timeslot 16 as out-of-band signaling.

Line Signaling (ITU-T Q.421)

In timeslot 16, each voice channel uses four ABCD bits to signal its states and call control. Bits C and D are rarely used. In some countries, they can be used for metering (pulse metering for billing). In a normal conversation, we have both sides working: the caller and the called side. Signaling from the caller side is referred to as forward signaling while the called side uses backward signaling. We will designate Af and Bf for forwarding signaling and Ab and Bb for backward signaling.

State	ABCD forward	ABCD backward
Idle/Released	1001	1001
Seized	0001	1001
Seize Ack	0001	1101
Answered	0001	0101
ClearBack	0001	1101
ClearFwd (Before clear-back)	1001	0101
ClearFwd (disconnection confirmation)	1001	1001
Blocked	1001	1101

MFC/R2 was defined by the ITU. Unfortunately, several countries customized the standard to their own needs. As a result, variations emerged in standards between countries.

Inter-register signals (ITU-T Q.441)

MFC/R2 signaling uses a combination of two tones. The table below shows the ITU standard.

Signal group I (Forward)

Signal	Description	Forward signal
1	Digit 1	I-1
2	Digit 2	I-2
3	Digit 3	I-3
4	Digit 4	I-4
5	Digit 5	I-5
6	Digit 6	I-6
7	Digit 7	I-7
8	Digit 8	I-8
9	Digit 9	I-9
10	Digit 0	I-10
11	Country code indicator, outgoing half-echo suppressor required	I-11
12	Country code indicator, no echo suppressor required	I-12
13	Test call indicator	I-13
14	Country code indicator, outgoing half-echo suppressor inserted	I-14
15	Not used	I-15

Signal group II (Forward)

Signal	Description	Forward signal
1	Subscriber without priority	II-1
2	Subscriber with priority	II-2
3	Maintenance equipment	II-3
4	Spare	II-4
5	Operator	II-5
6	Data Transmission	II-6
7	Subscriber or operator without forward transfer facility	II-7
8	Data transmission	II-8
9	Subscriber with priority	II-9
10	Operator with forward transfer facility	II-10
11	Spare	II-11

12	Spare	II-12
13	Spare	II-13
14	Spare	II-14
15	Spare	II-15

Signal group A (backwards)

Signal	Description	Backward signal
1	Send next digit (n+1)	A-1
2	Send last but one digit (n-1)	A-2
3	Address complete, changeover to reception of Group B signals	A-3
4	Congestion in the national network	A4
5	Send calling party's category	A5
6	Address complete, charge, set-up speech conditions	A6
7	Send last but two digit (n-2)	A7
8	Send last but three digit (n-3)	A8
9	Spare	A9
10	Spare	A10
11	Send country code indicator	A11
12	Send language or discrimination digit	A12
13	Send nature of circuit	A13
14	Request information on use of echo suppressor	A14
15	Congestion in an international exchange or at its output	A15

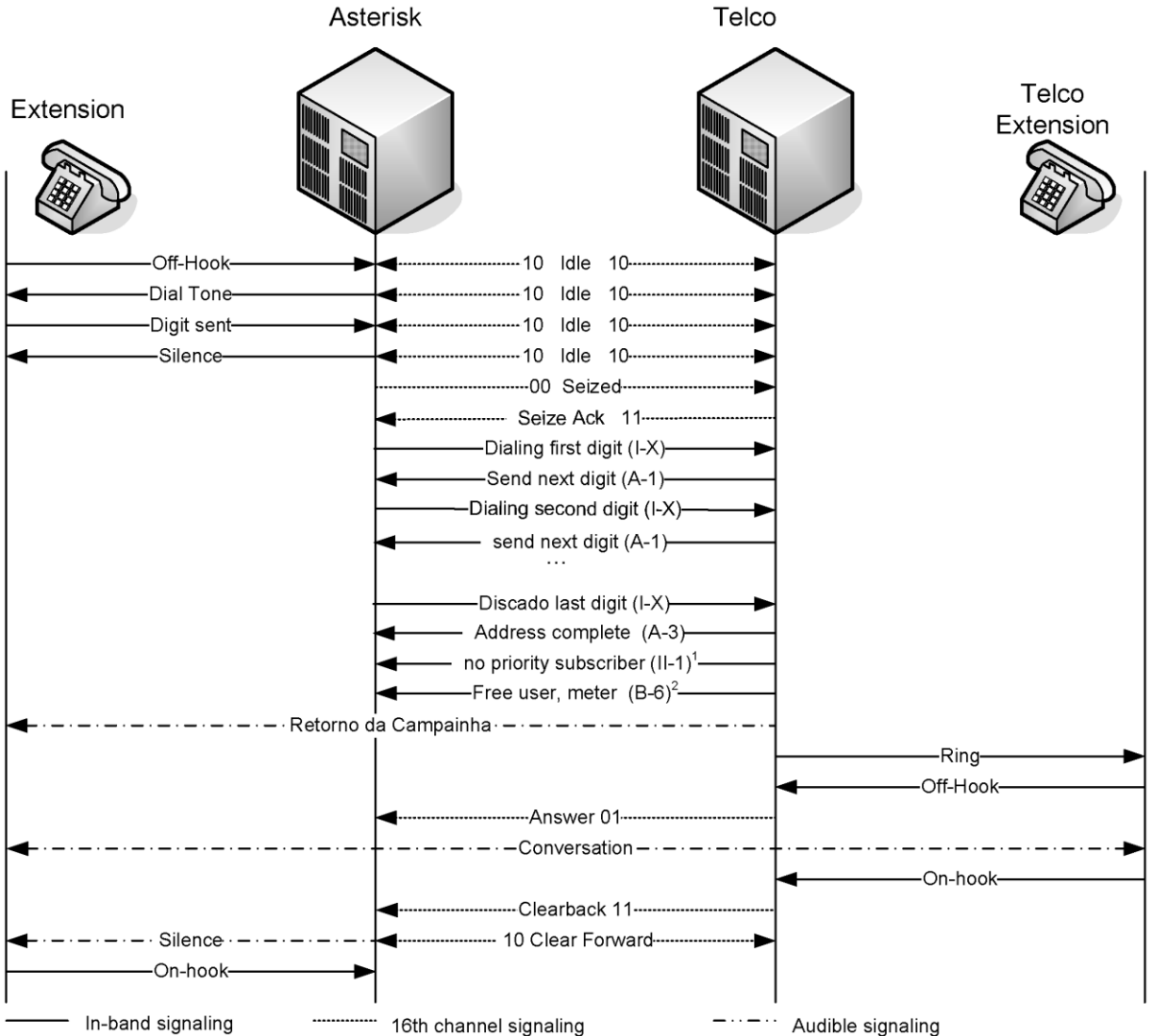
Signal group B (backwards)

Signal	Description	Backward signal
1	Spare	B1
2	Send special information tone	B2
3	Subscriber's line busy	B3
4	Congestion (after changeover group A to B)	B4
5	Unallocated number	B5
6	Subscriber's line free, charge	B6
7	Subscriber's line free, no charge	B7

8	Subscriber's line out of order	B8
9	Spare	B9
10	Spare	B10
11	Spare	B11
12	Spare	B12
13	Spare	B13
14	Spare	B14
15	Spare	B15

MFC/R2 sequence

The following sequence illustrates a call originating from an Asterisk's extension to a terminal in the PSTN. The PSTN drops the call and ends the communication.



How to use the driver libopenr2

The project initiated by Moises Silva was inspired on the Unicall channel driver written by Steve Underwood. The OpenR2 library is currently the most stable software solution for Asterisk. With this solution, we may use any digital card compatible with DAHDI. Previously, only proprietary solutions were available for MFC/R2, one of the best I have used is the one made available by Khomp, www.khomp.com.br. In the previous installation of Asterisk 1.6.2, we installed the OpenR2, so you

should have it installed and may go directly to the step four. If you want to install OpenR2 for Asterisk 1.4, follow the instructions below:

Step 1: Check the patches for the version of Asterisk you want to install.

apt-get install subversion

Step2: Download the modified Asterisk code with the patch installed.

cd /usr/src

svn checkout <http://svn.digium.com/svn/asterisk/team/moy/mfcr2/asterisk-1.4-openr2>

Step 3: Compile and install

Please, BACK UP your server before proceeding.

cd asterisk-1.4-openr2

./configure && make && make install

Note: Do not execute “make samples” to avoid overwriting your configuration files.

Step 4: Changing the file **/etc/daohdi/system.conf**:

vim /etc/daohdi/system.conf

Let’s suppose you have a card with one E1 interface.

```
span=1,1,0,cas,hdb3
cas=1-15:1101
cas=17-31:1101
dchan=16
loadzone=br
defaultzone=br
```

Step 5: Run the command **dahdi_cfg** to apply the changes to the driver:

dahdi_cfg -vvvvvvv

```
Dahdi Version:SVN-branch-1.4-r4348
Echo Canceller: MG2
Configuration
=====
SPAN 1: CAS/HDB3 Build-out: 0 db (CSU)/0-133 feet (DSX-1)

Channel map:

Channel 01: CAS / User (Default) (Slaves: 01)
Channel 02: CAS / User (Default) (Slaves: 02)
Channel 03: CAS / User (Default) (Slaves: 03)
Channel 04: CAS / User (Default) (Slaves: 04)
Channel 05: CAS / User (Default) (Slaves: 05)
Channel 06: CAS / User (Default) (Slaves: 06)
Channel 07: CAS / User (Default) (Slaves: 07)
Channel 08: CAS / User (Default) (Slaves: 08)
Channel 09: CAS / User (Default) (Slaves: 09)
Channel 10: CAS / User (Default) (Slaves: 10)
Channel 11: CAS / User (Default) (Slaves: 11)
```

```

Channel 12: CAS / User (Default) (Slaves: 12)
Channel 13: CAS / User (Default) (Slaves: 13)
Channel 14: CAS / User (Default) (Slaves: 14)
Channel 15: CAS / User (Default) (Slaves: 15)
Channel 16: D-channel (Default) (Slaves: 16)
Channel 17: CAS / User (Default) (Slaves: 17)
Channel 18: CAS / User (Default) (Slaves: 18)
Channel 19: CAS / User (Default) (Slaves: 19)
Channel 20: CAS / User (Default) (Slaves: 20)
Channel 21: CAS / User (Default) (Slaves: 21)
Channel 22: CAS / User (Default) (Slaves: 22)
Channel 23: CAS / User (Default) (Slaves: 23)
Channel 24: CAS / User (Default) (Slaves: 24)
Channel 25: CAS / User (Default) (Slaves: 25)
Channel 26: CAS / User (Default) (Slaves: 26)
Channel 27: CAS / User (Default) (Slaves: 27)
Channel 28: CAS / User (Default) (Slaves: 28)
Channel 29: CAS / User (Default) (Slaves: 29)
Channel 30: CAS / User (Default) (Slaves: 30)
Channel 31: CAS / User (Default) (Slaves: 31)

```

```
31 channels to configure.
```

Step 5: Change the file `chan_dahdi.conf`

vim /etc/asterisk/chan_dahdi.conf

```

[channels]
usecallerid=yes
callwaiting=yes
usecallingpres=yes
callwaitingcallerid=yes
threewaycalling=yes
transfer=yes
canpark=yes
cancallforward=yes
callreturn=yes
echocancel=yes
echotraining=yes
echocancelwhenbridged=yes

signalling=mfcr2
mfcr2_variant=br
mfcr2_get_ani_first=no
mfcr2_max_ani=20
mfcr2_max_dnis=4
mfcr2_category=national_subscriber
mfcr2_logdir=span1
mfcr2_logging=all

group=1
callgroup=1
pickupgroup=1
callerid=asreceived
context=from-mfcr2
channel => 1-15,17-31

```

Step 6: Change the dial plan in the file `extensions.conf`

vim /etc/asterisk/extensions.conf

```
[default]
exten => _XXXXXXX,1,Set(CALLERID(num)=1145678990)
exten => _XXXXXXX,n,Dial(ZAP/g1/${EXTEN},60,tT)
```

Note: Some TELCOS do not accept calls without the caller ID. Please set the caller ID to one of the DID numbers assigned by the operator. In some countries, this step is not required.

Step 7: Test the solution:

Now, with an extension in the context **from-internal**, call any number and observe the console. Check to see if any errors are occurring.

```
-- Executing Set("SIP/8564-081ca5d8", "CALLERID(num)=1145678990") in new stack
-- Executing Dial("SIP/8564-081ca5d8", "ZAP/g1/35678899|60|tT") in new stack
```

Debugging OpenR2

To detect errors in the calls, you can activate the debug. To do this, follow the steps below.

Step 1: Edit the file **chan_dahdi.conf** and add the following three lines to the configuration:

```
mfc2_logdir=span1
mfc2_logging=all
mfc2_call_files=yes
```

Step 2: Restart the Asterisk server

Step 3: Test the call and check the call files at **/var/log/asterisk/mfc2/span1**

Below is a trace for a normal call. Compare it to what you receive in your call.

```
[15:05:47:710] [Thread: 3078019984] [Chan 1] - Call started at Mon Jul 6 15:05:47 2009 on
chan 1
[15:05:47:710] [Thread: 3078019984] [Chan 1] - CAS Tx >> [SEIZE] 0x00
[15:05:47:710] [Thread: 3078019984] [Chan 1] - CAS Raw Tx >> 0x01
[15:05:47:951] [Thread: 3078019984] [Chan 1] - Bits changed from 0x08 to 0x0C
[15:05:47:951] [Thread: 3078019984] [Chan 1] - CAS Rx << [SEIZE ACK] 0x0C
[15:05:47:951] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 2
[15:05:47:951] [Thread: 3078019984] [Chan 1] - timer id 2 found, cancelling it now
[15:05:47:951] [Thread: 3078019984] [Chan 1] - Sending DNIS digit 3
[15:05:47:951] [Thread: 3078019984] [Chan 1] - MF Tx >> 3 [ON]
[15:05:48:070] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [ON]
[15:05:48:070] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:48:070] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:48:070] [Thread: 3078019984] [Chan 1] - MF Tx >> 3 [OFF]
[15:05:48:150] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [OFF]
[15:05:48:150] [Thread: 3078019984] [Chan 1] - Sending DNIS digit 0
[15:05:48:150] [Thread: 3078019984] [Chan 1] - MF Tx >> 0 [ON]
[15:05:48:150] [Thread: 3078019984] [Chan 1] - Group A DNIS request handled
[15:05:48:250] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [ON]
[15:05:48:250] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:48:250] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:48:250] [Thread: 3078019984] [Chan 1] - MF Tx >> 0 [OFF]
[15:05:48:350] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [OFF]
[15:05:48:350] [Thread: 3078019984] [Chan 1] - Sending DNIS digit 2
[15:05:48:350] [Thread: 3078019984] [Chan 1] - MF Tx >> 2 [ON]
[15:05:48:350] [Thread: 3078019984] [Chan 1] - Group A DNIS request handled
[15:05:48:450] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [ON]
```

```
[15:05:48:450] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:48:450] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:48:450] [Thread: 3078019984] [Chan 1] - MF Tx >> 2 [OFF]
[15:05:48:550] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [OFF]
[15:05:48:550] [Thread: 3078019984] [Chan 1] - Sending DNIS digit 5
[15:05:48:550] [Thread: 3078019984] [Chan 1] - MF Tx >> 5 [ON]
[15:05:48:550] [Thread: 3078019984] [Chan 1] - Group A DNIS request handled
[15:05:48:650] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [ON]
[15:05:48:650] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:48:650] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:48:650] [Thread: 3078019984] [Chan 1] - MF Tx >> 5 [OFF]
[15:05:48:750] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [OFF]
[15:05:48:750] [Thread: 3078019984] [Chan 1] - Sending DNIS digit 8
[15:05:48:750] [Thread: 3078019984] [Chan 1] - MF Tx >> 8 [ON]
[15:05:48:750] [Thread: 3078019984] [Chan 1] - Group A DNIS request handled
[15:05:48:850] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [ON]
[15:05:48:850] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:48:850] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:48:850] [Thread: 3078019984] [Chan 1] - MF Tx >> 8 [OFF]
[15:05:48:950] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [OFF]
[15:05:48:950] [Thread: 3078019984] [Chan 1] - Sending DNIS digit 5
[15:05:48:950] [Thread: 3078019984] [Chan 1] - MF Tx >> 5 [ON]
[15:05:48:950] [Thread: 3078019984] [Chan 1] - Group A DNIS request handled
[15:05:49:050] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [ON]
[15:05:49:050] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:49:050] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:49:050] [Thread: 3078019984] [Chan 1] - MF Tx >> 5 [OFF]
[15:05:49:150] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [OFF]
[15:05:49:150] [Thread: 3078019984] [Chan 1] - Sending DNIS digit 8
[15:05:49:150] [Thread: 3078019984] [Chan 1] - MF Tx >> 8 [ON]
[15:05:49:150] [Thread: 3078019984] [Chan 1] - Group A DNIS request handled
[15:05:49:250] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [ON]
[15:05:49:250] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:49:250] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:49:250] [Thread: 3078019984] [Chan 1] - MF Tx >> 8 [OFF]
[15:05:49:330] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [OFF]
[15:05:49:330] [Thread: 3078019984] [Chan 1] - Sending DNIS digit 4
[15:05:49:330] [Thread: 3078019984] [Chan 1] - MF Tx >> 4 [ON]
[15:05:49:330] [Thread: 3078019984] [Chan 1] - Group A DNIS request handled
[15:05:49:590] [Thread: 3078019984] [Chan 1] - MF Rx << 5 [ON]
[15:05:49:590] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:49:590] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:49:590] [Thread: 3078019984] [Chan 1] - MF Tx >> 4 [OFF]
[15:05:49:670] [Thread: 3078019984] [Chan 1] - MF Rx << 5 [OFF]
[15:05:49:670] [Thread: 3078019984] [Chan 1] - Sending category National Subscriber
[15:05:49:670] [Thread: 3078019984] [Chan 1] - MF Tx >> 1 [ON]
[15:05:49:770] [Thread: 3078019984] [Chan 1] - MF Rx << 5 [ON]
[15:05:49:770] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:49:770] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:49:770] [Thread: 3078019984] [Chan 1] - MF Tx >> 1 [OFF]
[15:05:49:850] [Thread: 3078019984] [Chan 1] - MF Rx << 5 [OFF]
[15:05:49:850] [Thread: 3078019984] [Chan 1] - Sending ANI digit 4
[15:05:49:850] [Thread: 3078019984] [Chan 1] - MF Tx >> 4 [ON]
[15:05:49:930] [Thread: 3078019984] [Chan 1] - MF Rx << 5 [ON]
[15:05:49:930] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:49:930] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:49:930] [Thread: 3078019984] [Chan 1] - MF Tx >> 4 [OFF]
[15:05:50:030] [Thread: 3078019984] [Chan 1] - MF Rx << 5 [OFF]
[15:05:50:030] [Thread: 3078019984] [Chan 1] - Sending ANI digit 8
[15:05:50:030] [Thread: 3078019984] [Chan 1] - MF Tx >> 8 [ON]
```



```

[15:05:51:810] [Thread: 3078019984] [Chan 1] - Sending more ANI unavailable
[15:05:51:810] [Thread: 3078019984] [Chan 1] - MF Tx >> F [ON]
[15:05:51:990] [Thread: 3078019984] [Chan 1] - MF Rx << 3 [ON]
[15:05:51:990] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:51:990] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:51:990] [Thread: 3078019984] [Chan 1] - MF Tx >> F [OFF]
[15:05:52:090] [Thread: 3078019984] [Chan 1] - MF Rx << 3 [OFF]
[15:05:52:090] [Thread: 3078019984] [Chan 1] - Sending category National Subscriber
[15:05:52:090] [Thread: 3078019984] [Chan 1] - MF Tx >> 1 [ON]
[15:05:53:350] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [ON]
[15:05:53:350] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:05:53:350] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:05:53:350] [Thread: 3078019984] [Chan 1] - MF Tx >> 1 [OFF]
[15:05:53:430] [Thread: 3078019984] [Chan 1] - MF Rx << 1 [OFF]
[15:06:03:322] [Thread: 3078019984] [Chan 1] - Attempting to cancel timer timer 0
[15:06:03:322] [Thread: 3078019984] [Chan 1] - Cannot cancel timer 0
[15:06:03:322] [Thread: 3078019984] [Chan 1] - CAS Tx >> [CLEAR FORWARD] 0x08
[15:06:03:322] [Thread: 3078019984] [Chan 1] - CAS Raw Tx >> 0x09
[15:06:03:569] [Thread: 3085228944] [Chan 1] - Bits changed from 0x0C to 0x08
[15:06:03:569] [Thread: 3085228944] [Chan 1] - CAS Rx << [IDLE] 0x08
[15:06:03:569] [Thread: 3085228944] [Chan 1] - Call ended
[15:06:03:569] [Thread: 3085228944] [Chan 1] - Attempting to cancel timer timer 0
[15:06:03:569] [Thread: 3085228944] [Chan 1] - Cannot cancel timer 0

```

MFC/R2 Configuration

The options are documented within the file `chan_dahdi.conf`. Some of the most important options are detailed here.

Mandatory parameters: `mfc_r2_variant`, `mfc_r2_max_ani` and `mfc_r2_max_dnis`.

mfc_r2_variant: Country variant.

r2test -l

Variant Code	Country
AR	Argentina
BR	Brazil
CN	China
CZ	Czech Republic
CO	Colombia
EC	Ecuador
ITU	International Telecommunication Union
MX	Mexico
PH	Philippines
VE	Venezuela

mfc_r2_max_ani: Max amount of ANI digits to ask for

mfc_r2_max_dnis: Max amount of DNIS digits to ask for

mfc_r2_get_ani_first: Whether or not to get ANI before DNIS (required by some TELCOS)

mfc_r2_category: Caller category. You can set the variable `MFCR2_CATEGORY` before starting the call

mfc2_logdir: Directory to log the call files. (/var/log/asterisk/mfc2/**directory**)

mfc2_call_files: Whether or not to log the calls

mfc2_logging: logging values

- cas – ABCD bits for tx and rx
- mf – Multifrequency tones
- stack – verbose output of the channel and context stack
- all – all activities
- nothing – do not log anything

mfc2_mfback_timeout: This value deserves to be mentioned. Sometimes if you are calling a cell phone or any call that takes a long time to complete, this parameter can time out, so it is often changed for fine tuning. If some of your calls are not being completed, this is the parameter you should change first.

mfc2_metering_pulse_timeout: Pulses are used by some R2 variants to indicate costs

mfc2_allow_collect_calls: In Brazil, the tone II-8 is used to indicate a collect call; this parameter allows you to block collect calls.

mfc2_double_answer: Also used to avoid collect calls when a double answer is required. With **doub1e_answer=yes** you actually block the collect calls.

mfc2_immediate_accept: Allows you to skip the use of group B/II signals and go directly to the accepted state.

mfc2_forced_release: Allows you to speed up the release of the call; works for the Brazilian variant.

ANI and DNIS

Automatic Number Identification (ANI) is the caller's number. Dialed Number Identification Service (DNIS) is the number called or, in other words, the number dialed.

When a call is received, usually the last four numbers are passed to the PBX in a process referred to as direct inward dial (DID). The ANI number is actually the Caller ID. ANI will have the caller's extension when dialing while DNIS will contain the call destination. It is important that these parameters be configured correctly. Some switches send just the last four digits while others send the complete number.

DAHDI channel format

DAHDI channels use the following format in the dial plan:

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

<identifier>- Physical channel numeric identifier

[g] - Group identifier
 [c] - Answer confirmation. A number is not considered until the callee press "#"
 [r] - customized ringing
 [cadence] Integer from 1 to 4

Examples:

```
DAHDI/2 - channel 2
DAHDI/g1 - First available channel in group 1
[g] - Group identifier
[c] - Answer confirmation; A number is not considered until the callee press "#"
[r] - customized ringing
[ cadence] Integer from 1 to 4
```

Questions

- 1 – In regard to T1 and E1 signaling, mark the correct affirmations.
 - A. E1 is digital signaling that uses 1.544 Mbits/s bandwidth.
 - B. T1 is often used in Latin America and Europe.
 - C. It is possible to use 30 channels for an E1 trunk and 23 channels for a T1 trunk in an ISDN PRI configuration.
 - D. ISDN is an example of CCS signaling while MFC/R2 is an example of CAS signaling.
- 2 – To configure the hardware with a DAHDI interface, you should first edit the _____ file.
 - A. system.conf
 - B. chan_dahdi.conf
 - C. unicall.conf
 - D. serial.conf
- 3 – The DAHDI hardware is independent of Asterisk. In `chan_dahdi.conf`, you configure Asterisk channels and not the hardware itself.
 - A. False
 - B. True
- 4 – R2 signaling defined by ITU is standardized throughout the world, and no variations to the standard exist based on the country.
 - A. True
 - B. False
- 5 – The utility to detect and configure the DAHDI channel automatically is:
 - A. dahdi_generator

- B. dahdi_genconf
- C. dahdigenconf
- D. generate_dahdi

6 – ISDN BRI is common on Europe. An ISDN BRI line supports ___ voice/data channels and ___ signaling channel(s).

- A. 15, 2
- B. 30,2
- C. 23,1
- D. 2,1

7 – If you have a server with a PCI interface type of 64 bits 3.3V and you need 4E1s, what card should you use?

- A. TE410P
- B. TE405P
- C. TE110P
- D. TE205P

8 – When using a USB 2.0 connection, you can support only 32 channels.

- A. True
- B. False

9 – The support for OpenR2 was included in Asterisk version ___; all other versions need to be patched if you want to use this signaling.

- A. 1.6.0
- B. 1.4.25
- C. 1.2.1
- D. 1.6.2

10 – You can improve Asterisk’s echo cancellation by installing OSLEC.

- A. True
- B. False

6

Designing a VoIP network

Voice over IP is quickly growing in the telephony market. The convergence paradigm is changing the way in which we communicate, reducing costs and enhancing the way in which we trade information. Voice is just the beginning of a full multimedia communication era, including voice, video, and presence. In the future, we are not going to transport people to work, but work to people because it is cleaner, faster, and cheaper. VoIP is just part of this revolution. Our challenge in this chapter is to design a VoIP network. To do this, we will have to understand concepts such as session protocols and codecs as well as how to dimension the number of circuits and bandwidth.

Objectives

By the end of this chapter, you should be able to:

- Understand the benefits of VoIP
- Describe how Asterisk handles VoIP
- Describe the concepts of the SIP, IAX, and H323 channels
- Choose the most adequate protocol for a specific data channel
- Choose the most adequate codec for a specific data channel
- Dimension the required number of channels
- Calculate the required bandwidth

VoIP benefits

Why would you care about VoIP? VoIP provides benefits to both companies and individuals. Cost reduction is certainly one of them, but in some environments VoIP simplifies the integration of computer systems. Several of the benefits are detailed here:

Convergence

The primary benefit of VoIP is the combination of data and voice networks to reduce costs (convergence). However, analyzing just voice minute costs may not be enough to justify the adoption of VoIP. The price of the minutes sold by phone companies is quickly becoming cheaper and is something to be considered before adopting VoIP.

Infrastructure costs

The use of a single network infrastructure reduces the costs associated with additions, removals, and changes. As IP has become pervasive, it has brought VoIP-related technology to several new devices, such as cell phones, PDAs, embedded systems, and laptops.

Open Standards

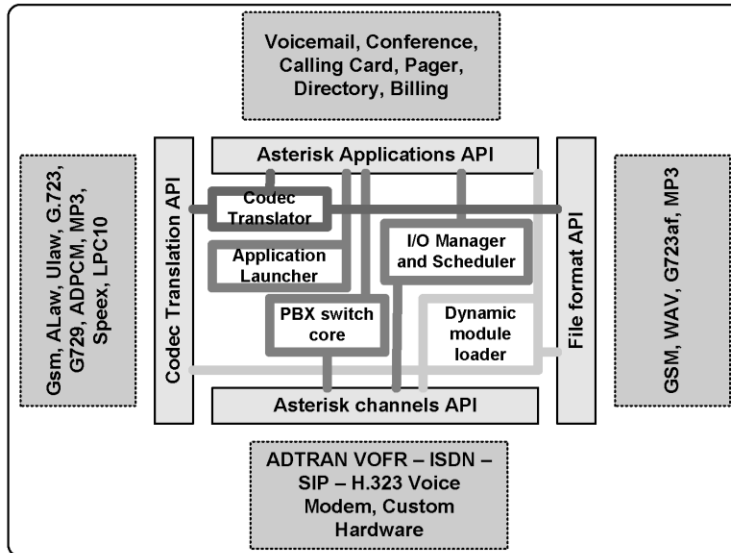
Finally, the open standards upon which VoIP is built provide the freedom to choose from different vendors. This single benefit makes the customer king instead of a subordinate to TELCOS and PBX manufacturers.

Computer Telephony Integration

Telephony is far older than computing. Telephony PBXs are circuit-switch based, and you usually do not have more than a computer for supervision. With VoIP, telephony is from the ground up created based in computer standards. This makes the use of Computer Telephony applications cheaper and easier than in the old model. You can quickly create a long list of telephony applications based on Asterisk. You can develop IVRs, ACDs, CTI, dialers, screen popups, and other applications in a fraction of the time required for traditional PBXs.

Asterisk VoIP architecture

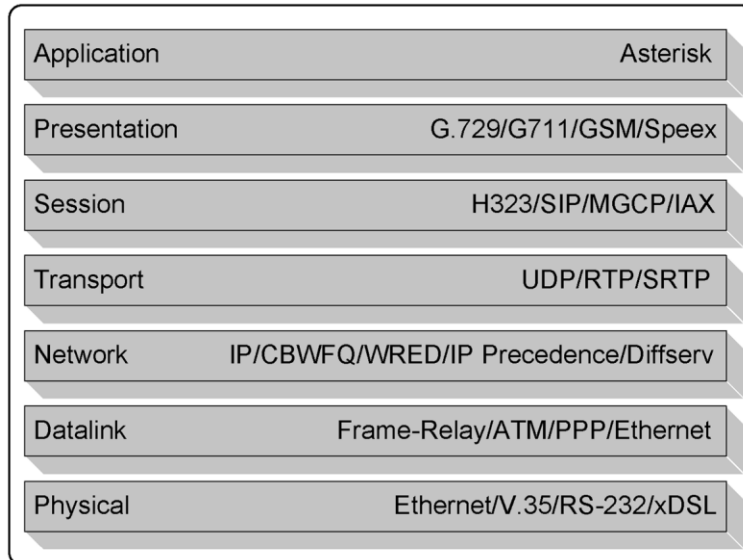
Asterisk's architecture is shown below. Asterisk treats all VoIP protocols as channels. You can use any codec or any protocol. The concept to be learned here is that Asterisk bridges any type of channel to any other. Thus, you can translate signaling protocols such as H.323, SIP, and IAX to one another and even with different codecs. For example, you can translate a call from a SIP phone in the local area network using the G.711 codec to a H323 connection to your VoIP provider using the G.729 codec.



In the next chapters, we will explain the details of the SIP and IAX architecture. As H.323 is not part of Asterisk (although available as an add-on), we will not cover it in this book.

VoIP protocols and the ISO Open Systems Interconnect (OSI) model

As you can see below, VoIP uses a set of different protocols working together. Different OSI layers are present in VoIP communication. The figure below will help you understand the role of each protocol and their relationships.



The first four layers represent a data network, just like the Internet you have in your business or home. You can use some QoS protocols like “diffserv” or “cbwfq” to prioritize voice packets and enhance voice quality. Most VoIP protocols use real-time protocol (RTP) as the transport protocol of choice.

In the session layer, protocols are responsible for setting up and closing the calls. H.323 is one of the oldest and mature protocols in this area. SIP is now pervasive in the VoIP provider market, putting aside H.323. Signaling protocols use TCP or UDP to transport the packets.

In the presentation layer, the codecs transform the multimedia stream from one format to another based on different characteristics. For example:

SIP: SIP uses UDP or TCP in port 5060 to transport signaling. RTP transports the audio stream using ports 1000 to 2000 in Asterisk (as defined in [rtp.conf](#)). For example, a call can be coded in g.711. A soft-phone in the application layer will use the lower layers to communicate.

H.323: H.323 uses TCP in ports 1720 and 1719 to transport signaling. RTP usually transports audio in UDP ports 16383 to 32768.

How to choose a protocol

Given the many protocols, how can you choose the best one for your network? In this section, we will highlight the advantages and drawbacks of each protocol.

SIP - Session Initiated Protocol

SIP is an Internet Engineering Task Force (IETF) open standard, largely defined in RFC 3261. Most modern VoIP providers use SIP; indeed, it is becoming the most popular VoIP standard. The strength of SIP is that it is an IETF-based standard. SIP is light when compared to the older H.323. SIP’s main

weakness is the NAT traversal—a challenge to most SIP VoIP providers. IETF did not create SIP with billing in mind, but for open communications between peers. Billing is usually a concern for VoIP providers.

IAX – Inter Asterisk eXchange

IAX is an open protocol defined by Digium and is currently in a draft form. You can download it from www.ietf.org/internet-drafts/drafts-guy-iax00.txt. IAX is an all-in-one protocol as it transports signaling and media through the same UDP port (4569). Mark Spencer developed IAX as a binary protocol for reduced bandwidth. The main strength of IAX is its reduced bandwidth usage (it does not use RTP); it is also very easy for NAT and firewall traversal since it uses only one UDP port (4569). If a traditional PBX manufacturer were to have created IAX, it would probably have marketed the protocol as the “best thing since ice cream”; in some situations, IAX in trunk mode can reduce voice bandwidth use by one third. As the time of this writing, this protocol was in version 2.

MGCP – Media Gateway Control Protocol

MGCP is a protocol used in conjunction with H.323, SIP, and IAX. Its greatest advantage is scalability. It is configured in the call agent instead of the gateways. This simplifies the configuration process and permits centralized management. However, Asterisk implementation is not complete, and it seems that not many people use it.

H.323

H.323 is largely being used in VoIP. It is one of the first VoIP protocols and is essential for connecting older VoIP infrastructures based in gateways. H.323 is still the standard in the gateway market, although the market is slowly migrating to SIP. H.323’s strengths include the large market adoption and maturity. H.323’s weaknesses are related to the complexity of implementation and standard bodies’ associated costs.

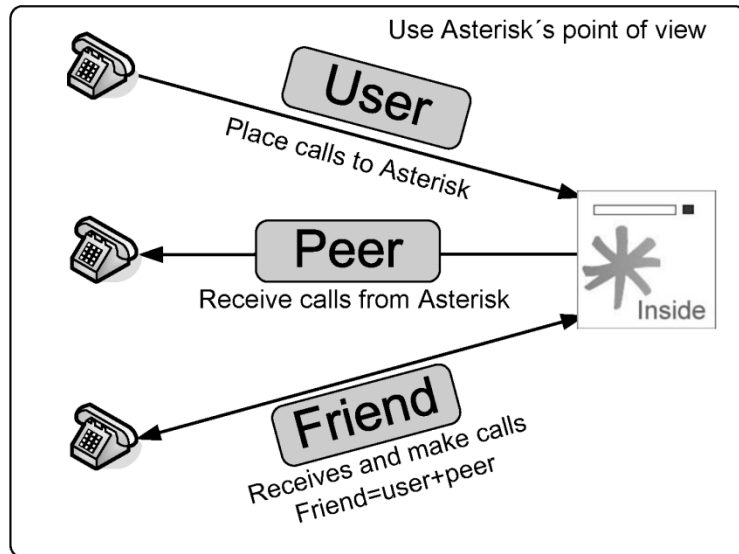
Protocol comparison table

The following table summarizes the differences among the session protocols.

Protocol	Standard body	Is used for:
IAX2	IETF draft	Asterisk trunks IAX2 phones Connection to IAX service providers
SIP	IETF standard	SIP phones Connection to SIP service providers
MGCP	IETF/ITU standard	MGCP phones Currently does not support connecting to a MGCP gateway or service provider
H.323	ITU standard	H.323 phones H.323 gateways Currently does not support being a gatekeeper, but can connect to an external gatekeeper.

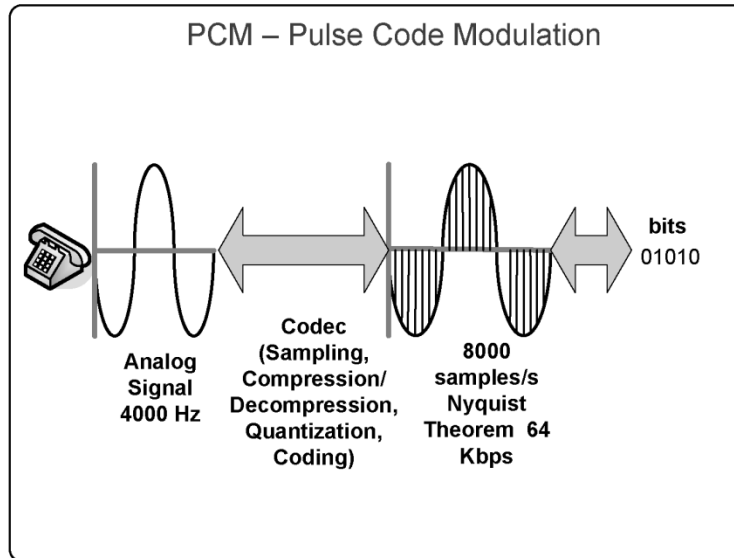
Peers, Users, and Friends

Three kinds of SIP and IAX clients exist. The first one is “user”. Users can make calls to an Asterisk server, but they cannot connect to receive calls from this server. The second one is a “peer”. You can make calls to a peer, but you will not receive calls from them. Usually a server or a device will require both concepts at the same time. A “friend” is a shortcut to a “user” + “peer”. A phone would probably fall into this category as it is needed to make and receive calls.



Codecs and codec translation

You will use a codec to convert the voice from an analog wave to a digital signal. Codecs differ from one another in aspects such as sound quality, compression rate, bandwidth, and computing requirements. Services, phones, and gateways usually support several of these aspects. The codec g729 is very popular and requires licensing.



Asterisk supports the following codecs:

- GSM: 13 Kbps
- iLBC: 13.3 Kbps
- ITU G.711: 64 Kbps
- ITU G.723.1: 5.3/6.3 Kbps
- ITU G.726: 16/24/32/40 Kbps
- ITU G.729: 8 Kbps
- Speex - 2.15 to 44.2 Kbps
- LPC10 - 2.5 Kbps

In addition, Asterisk permits translation among codecs. In some cases, this is not possible, such as the case of g723, which is supported only in pass-thru mode. Translating from one codec to another consumes many resources from the CPU. Thus, avoid this altogether whenever possible.

How to choose a Codec

Codec selection depends on several options, such as:

- Sound quality
- Licensing costs
- CPU-processing consumption
- Bandwidth requirements
- Packet-loss concealment

- Availability for Asterisk and phone devices

The following table compares the most popular codecs. The quality of these codecs is considered “toll”—in other words, similar to PSTN.

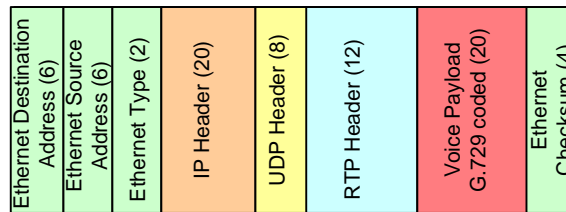
Codec	g.711	g.729A (20 ms)	iLBC (30 ms)	GSM 06.10 RTE/LTP
Bandwidth (Kbps)	64	8	13.33	13
Costs	Free	~ USD10.00 (per channel)	Free	Free
Resistance to Frame Erasure ¹	No mechanism	3%	5%	3%
Complexity MIPS ²	~0.35	~13	~18	~5

¹ Resistance to packet loss refers to the rate when MOS is next to 0.5 worst from peak quality for the specific codec.

² Complexity refers to quantities in millions of instructions per second spent to code and decode the codec using a reference design in a Texas Instruments DSP (TMS320C54x). A direct relationship exists between processor frequency and MIPS, but it is not possible to draw a precise relationship among such diverse hardware platforms. Use this table just for comparison.

Overhead caused by protocol headers

Despite the fact that codecs make little use of bandwidth, we have to consider the overhead caused by protocol headers like Ethernet, IP, UDP, and RTP. As such, we could say that bandwidth depends upon the headers used. If we are in an Ethernet network, the bandwidth requirement is higher than in a PPP network because the PPP header is shorter than the Ethernet one. Let’s look through some examples:



Example: Voice packet coded in g.729 20 ms sampling rate.

20 Bytes Payload/58 Bytes headers

Using simple proportion, if 20 bytes is 8 Kbps, 98 Bytes is 31.2 Kbps

A g.729 conversation in an Ethernet Network consumes 31.2 Kbps

Codec g.711 (64 Kbps)

- **Ethernet** (Ethernet+IP+UDP+RTP+G.711) = 95.2 Kbps
- **PPP** (PPP+IP+UDP+RTP+G.711) = 82.4 Kbps
- **Frame-Relay** (FR+IP+UDP+RTP+G.711) = 82.8 Kbps

Codec G.729 (8 Kbps)

- **Ethernet** (Ethernet+IP+UDP+RTP+G.729) = 31.2 Kbps
- **PPP** (PPP+IP+UDP+RTP+G.729) = 26.4 Kbps
- **Frame-Relay** (FR+IP+UDP+RTP+G.729) = 26.8 Kbps

You can easily calculate other bandwidth requirements using the calculator at the following website:
<http://www.asteriskguide.com/bandcalc/bandcalc.php>.

Traffic Engineering

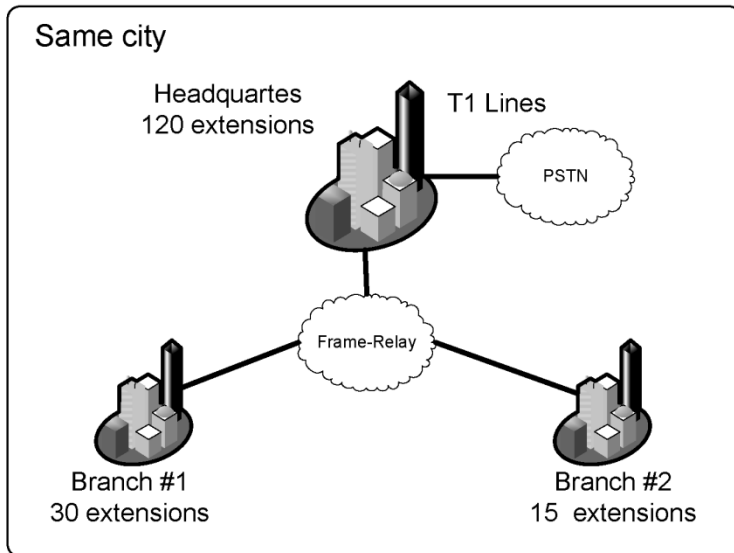
A main issue in the design of VoIP networks is dimensioning the number of lines and the required bandwidth to a specific destination, like a remote office or a service provider. It is also important to dimension the number of Asterisk's simultaneous calls (main parameter for Asterisk's dimensioning).

Simplifications

The primary and most widely used simplification is to estimate the number of calls by user type. For example:

- Business PBXs (one simultaneous call for every five extensions)
- Residential users (one simultaneous call for every sixteen users)

Example #1



The company's headquarters have 120 extensions and two branches—the first with 30 extensions and the second with 15 extensions. Our objective is to dimension the number of E1 trunks in the headquarters and the bandwidth required for the Frame-Relay network.

1a Number of T1 lines

- Total number of extensions using T1 lines: $120+30+15=165$ lines
- Using one trunk for each five extensions for business use
- Total number of lines = 33 or approximately 2xT1 lines

1b Bandwidth requirements

We choose the g.729 codec because of bandwidth requirements, sound quality, and medium CPU consumption.

Bandwidth Calculator for VOIP	
SIMULTANEOUS CALLS: 1	PAYLOAD: 160 BYTES SAMPLING: 0.125 MS
CODEC: g.711 64 Kbps (all variants)	MOS: 4.41 MIPS: ~0.35 DURATION: 20 MS
FRAMES PER PACKET: 160	L2 HEADER: 0 BYTES ATM CELLS: NA
L2 TECHNOLOGY: Layer2 (IP/UDP/RTP)	L3 HEADER: 40 BYTES
PROTOCOL: SIP	VPN HEADER: 0 TOTAL PAYLOAD: BYTES
VPN: NONE	BANDWIDTH (ONE CALL): Kbps
PROT. OVERHEAD: 5 %	BANDWIDTH (ALL CALLS): Kbps
<input type="checkbox"/> Compressed RTP	BANDWIDTH WITH OVERHEAD: Kbps
<p>1-Some codecs presented here are not supported on Asterisk™ PBX. 2-Payload size can be increased at the endpoint. 3-To enable iLBC 15.2 is necessary to edit the Asterisk™ source code. 4-Asterisk GSM™ is also known by GSM-FR. 5-The results obtained from the calculator should be considered only in one direction. In half-duplex networks, like 802.11b(WiFi), you should double the estimates. 6-Network design and other protocols running in the same network should be considered too.</p> <p>INFORMATIVE REFERENCES:</p> <p>PacketCable Audio/Video Codecs Specification, PKT-SP-CODEC-106-050812, August 12, 2005 PacketCable 2.0 Codec and Media Specification, PKT-SP-CODEC-MEDIA-102-061013, October 13, 2006 Cisco Voice Over IP - Per Call Bandwidth Consumption, Tech Notes, Feb 02, 2008</p> <p>DISCLAIMER:</p> <p>Information obtained from using this Bandwidth Calculator is believed to be accurate. Nonetheless, the author makes no guarantee that the results will provide any particular level of accuracy. Results obtained from using this calculator should not be relied upon as a sole source of information.</p> <p>TRADEMARKS:</p>	

With one trunk for every five extensions:

- Required bandwidth for branch #1 (Frame-relay): $26.8 \times 6 = 160.8$ Kbps
- Required bandwidth for branch #2 (Frame-relay): $26.8 \times 3 = 80.4$ Kbps

Erlang B method

1.a Number of VoIP simultaneous calls

Sometimes, simplification is not the best approach. When you have previous data, you can adopt a more scientific approach. We will use the work of Agner Karup Erlang (Copenhagen Telephone Company, 1909), who developed a formula to calculate lines in a trunk group between two cities. Erlang is a traffic measurement unit usually found in telecom. It is used to describe the volume of traffic for one hour.

For example: 20 calls occur in an hour, averaging 5 minutes of conversation each.

You can calculate the number of Erlangs as shown below:

Traffic minutes in the hour: $20 \times 5 = 100$ minutes

Hour of traffic inside one hour: $100/60 = 1.66$ Erlangs

You can determine these measures from a call logger and use it to design your network to calculate the number of lines required. Once the number of lines is known, it is possible to calculate the bandwidth requirements.

Erlang B is the most commonly used method for calculating the number of lines in a trunk group. It assumes that calls arrive randomly (Poisson distribution) while blocked calls are immediately cleared. This method requires that you know the Busy Hour Traffic (BHT), which you can obtain from a call logger or by the following simplification:

$BHT = 17\%$ of the call minutes of one day.

Another important variable is Grade of Service (GoS), which defines the probability of blocking calls by line shortage. You can arbitrate this parameter, which is usually 0.05 (5% calls lost) or 0.01 (1% calls lost).

Example #1:

Using the same example from 5.10.1, we will give you some data about traffic patterns. From the call logger, we discovered these data:

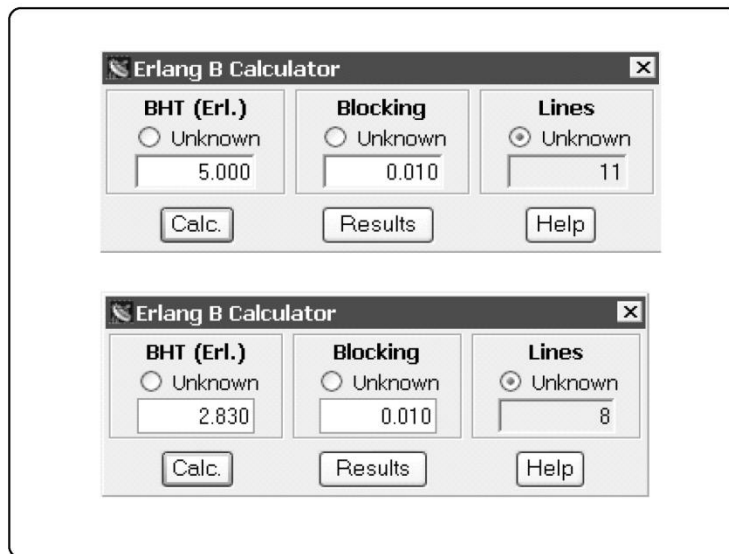
Data from call logger (Call minutes and BHT):

- Headquarters to Branch #1 = 2,000 minutes, BHT = 300 minutes
- Headquarters to Branch #2 = 1,000 minutes, BHT = 170 minutes
- Branch #1 to Branch #2 = 0, BHT=0

Let's arbitrate GoS=0.01

- Headquarters to Branch #1 - BHT=300 minutes/60 = 5 Erlangs
- Headquarters to Branch #2 - BHT=170 minutes/60 = 2.83 Erlangs

Using an Erlang Calculator (www.erlang.com)



- For the Headquarters to Branch #1, 11 lines are required.
- For the Headquarters to Branch #2, 8 lines are required

1.b Bandwidth Required

We are using a WAN where packet loss is rare. We will choose the g729 codec because of its good sound quality and data compression (8 Kbps).

Selected codec: g729

Datalink layer: Frame-Relay

- Estimated voice bandwidth for Branch #1: $26.8 \times 11 = 294.8$ Kbps
- Estimated voice bandwidth for Branch #2: $26.8 \times 8 = 214.40$ Kbps

Reducing the bandwidth required for VoIP

Three methods can be used to reduce the bandwidth required for VoIP calls:

- RTP header compression
- IAX Trunked
- VoIP payload

RTP Header Compression

In Frame-Relay and PPP networks, you can use RTP header compression. RTP header compression was defined in RFC 2508. It is an IETF standard available in several routers. However, be cautious, as some routers require a different feature set in order for this resource to be available.

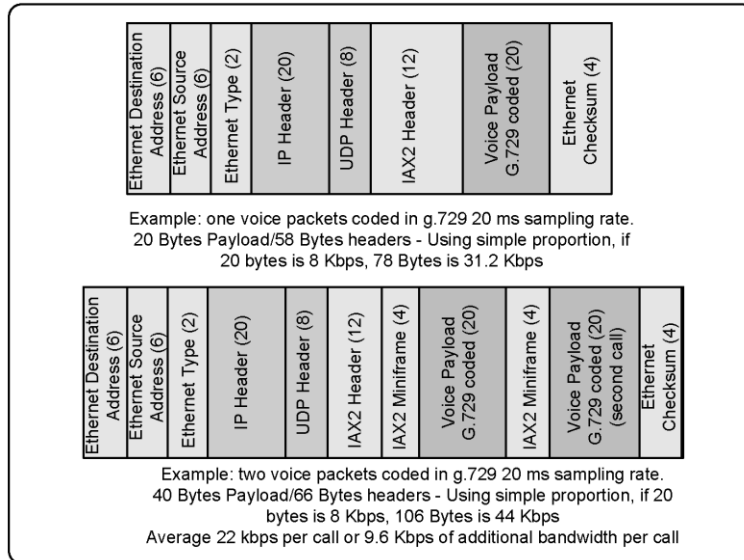
The impact of using RTP header compression is fabulous as it reduces the bandwidth required in our example from 26.8 Kbps per voice conversation to 11.2 Kbps—a 58.2% reduction!

The screenshot shows the Asterisk Guide Bandwidth Calculator for VoIP. The interface includes a header with the AsteriskGUIDE logo and navigation links. The main content area is titled 'Bandwidth Calculator for VOIP' and contains several input fields and a results section.

Parameter	Value
SIMULTANEOUS CALLS	1
CODEC	g.729a 8 Kbps
FRAMES PER PACKET	2
L2 TECHNOLOGY	Frame-Relay
PROTOCOL	SIP
VPN	NONE
PROT. OVERHEAD	5 %
<input checked="" type="checkbox"/> Compressed RTP	
PAYLOAD	20 BYTES
SAMPLING	10 MS
MOS	4.14
MIPS	~13
DURATION	20 MS
L2 HEADER	7 BYTES
L3 HEADER	2 BYTES
ATM CELLS	0
VPN HEADER	0
TOTAL PAYLOAD	29 BYTES
BANDWIDTH (ONE CALL)	11.6 Kbps
BANDWIDTH (ALL CALLS)	11.6 Kbps
BANDWIDTH WITH OVERHEAD	12.18 Kbps

IAX2 trunk mode

If you are connecting two Asterisk servers, you can use the IAX2 protocol in the trunk mode. This revolutionary technology does not need any special routers and can be applied to any kind of data link.



The IAX2 trunk mode reuses the same headers from the second call and over. Using g729 in a PPP link, the first call will consume 30 Kbps of bandwidth, whereas the second call will use the same header as the first and reduce the necessary bandwidth for the additional call to 9.6 Kbps. We can calculate the required bandwidth in trunk mode as follows:

Branch #1 (11 calls)

$$\text{Bandwidth} = 31.2 + (11-1) * 9.6 \text{ Kbps} = 127.2 \text{ Kbps}$$

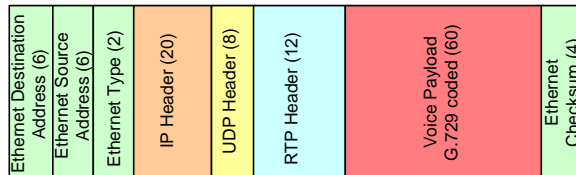
Branch #2 (8 calls)

$$\text{Bandwidth} = 31.2 + (8-1) * 9.6 \text{ Kbps} = 98.4 \text{ Kbps}$$

The first call uses 31.2 Kbps, the next 9.6, and so on.

Increasing the Voice Payload

This method is very common when using VoIP gateways over the Internet. When using a bigger payload, you will sacrifice latency in favor of reduced bandwidth. You can change the RTP packetization by appending the frame size to the codec in the `allow` instruction.



Example: Voice packet coded in g.729 20 ms sampling rate.
60 Bytes Payload/58 Bytes headers

Using simple proportion, if 60 bytes is 8 Kbps, 138 Bytes is 16.05 Kbps

A g.729 conversation in na Ethernet Network consumes 16.05 Kbps

Example:

allow=ulaw:30

The permitted values are:

Name	Min	Max	Default	Increment
g723	30	300	30	30
gsm	20	300	20	20
ulaw	10	150	20	10
alaw	10	150	20	10
g726	10	300	20	10
ADPCM	10	300	20	10
SLIN	10	70	20	10
lpc10	20	20	20	20
g729	10	230	20	10
speex	10	60	20	10
ilbc	30	30	30	30

Summary

In this chapter, you have learned that Asterisk treats VoIP using channels. It supports SIP, IAX, H.323, MGCP, and Skinny protocols. You compared and learned how to choose a signaling protocol and a codec for VoIP channels. The IAX2 is more bandwidth efficient and can traverse NAT easily. SIP is the most supported protocol by third-party phone and gateways vendors. The H.323 protocol is the oldest one and should be used to connect to legacy VoIP infrastructures. In section 5.11, we learned how to design and dimension a VoIP network.

Quiz

1. Please, list at least four benefits of VoIP.

2. Convergence is the integration of voice, data, and video in a single network; its primary benefit is the cost reduction in the implementation and maintenance of separate networks.
- A. False
 - B. True
3. Asterisk cannot use resources from PSTN and VoIP simultaneously because the codecs are not compatible.
- A. False
 - B. True
4. Asterisk is a SIP proxy with integration to other protocols
- A. False
 - B. True
5. Using the OSI reference model, SIP, H.323, and IAX2 are in the _____ layer.
- A. Presentation
 - B. Application
 - C. Physical
 - D. Session
 - E. Datalink
6. SIP is the most adopted protocol for IP phones and is an open standard ratified by IETF.
- A. False
 - B. True
7. H.323 is an inexpressive protocol with very few applications, abandoned by the market, which is moving to SIP.
- A. False
 - B. True
8. IAX is a proprietary Digium™ protocol. Despite its limited adoption by phone vendors, IAX is excellent when you need: (check all that apply)
- A. To reduce bandwidth usage
 - B. Video media format
 - C. NAT traversal
 - D. Protocols standardized by IETF or ITU.
9. “Users” can receive calls from Asterisk.

- A. False
 - B. True
10. Regarding codecs: (check all the true affirmations)
- A. G711 is the equivalent to PCM and uses 64 Kbps of bandwidth.
 - B. G.729 is free for commercial use and uses 8 Kbps of bandwidth.
 - C. GSM is growing because it uses approximately 13 Kbps and does not need a license.
 - D. G711 u-law is common in the US whereas a-law is common in Europe and Latin America.
 - E. G.729 is light and uses very few CPU resources in their coding/decoding process.

7

The IAX Protocol

In this chapter, we will learn about the Inter-Asterisk eXchange (IAX) protocol, including its strengths and weaknesses. Details such as trunk mode and the interconnection of two Asterisk servers will also be covered. All references in this document correspond to IAX version 2. The IAX protocol provides media transport and signaling for voice and video. IAX is very innovative; it saves bandwidth in trunk mode and is much simpler than SIP when you need to traverse NAT. The primary use for IAX nowadays is to interconnect Asterisk servers. IAX was created primarily for voice, but it can also accommodate video and other multimedia streams. IAX was inspired from other VoIP protocols, such as SIP and MGCP. Instead of using two separate protocols for signaling and media, IAX unified them to make a unique protocol. IAX does not use RTP for media transport; instead, it embeds the media in the same UDP connection.

Objectives

By the end of this chapter, you should be able to:

- Identify strengths and weakness of IAX protocol
- Describe usage scenarios for the IAX protocol
- Describe the advantages of IAX trunk mode
- Configure `iax.conf` for phones
- Configure `iax.conf` for connection to a VoIP provider
- Configure `iax.conf` for Asterisk interconnection
- Understand IAX authentication

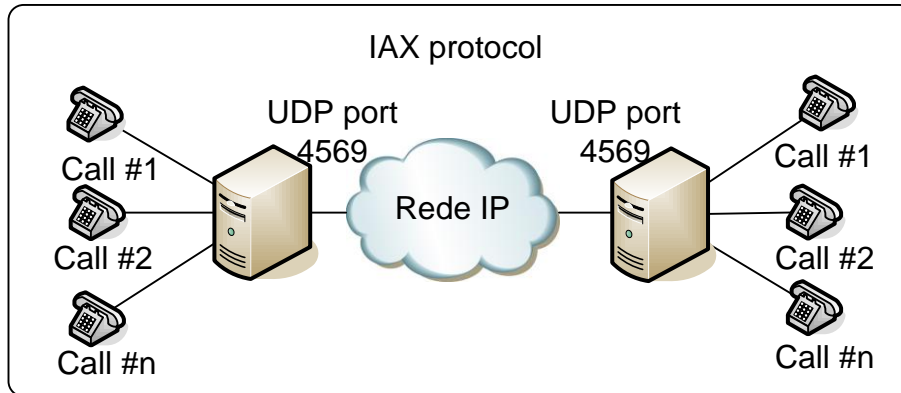
IAX design

The main objectives for IAX design are:

- To reduce the bandwidth required for media transport and signaling
- To provide NAT transparency
- To be able to transmit the dial plan information

- To support the efficient use of paging and intercom

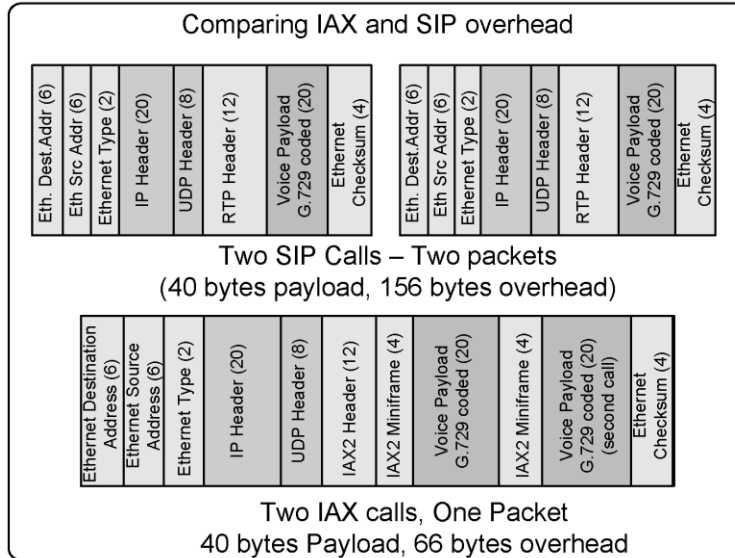
IAX is a peer-to-peer signaling and media protocol that is similar to SIP without using RTP. The basic approach is to multiplex the multimedia streams over a single UDP connection between two hosts. The greatest benefit of this approach is its simplicity when traversing connections over NAT, regularly found in xDSL modems. IAX uses a single port, UDP 4569 by default, and then uses a call number with 15 bits to multiplex all streams.



The IAX protocol uses registration and authentication processes similar to the SIP protocol. A description of the protocol can be found at <http://www.ietf.org/internet-drafts/draft-guy-iax-05.txt>

Bandwidth usage

The bandwidth used in VoIP networks is affected by several factors; codecs and protocol headers are the most important. The IAX protocol has a surprising feature called trunk mode, whereby it multiplexes several calls using a single header. By playing with the Asterisk bandwidth calculator, you will see how IAX trunks can save you up to 80% of the traffic with multiple calls.



Channel naming

It is important to understand channel-naming conventions as you will use these names when specifying a channel in the dial plan. The format of an IAX channel name used for outbound channels is:

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

<user>	UserID on remote peer, or name of client configured in iax.conf
<secret>	The password. Alternatively it can be the filename for an RSA key without the trailing extension (.key or .pub) and enclosed in square brackets
<peer>	Name of server to connect to
<portno>	Port number for connection
<exten>	Extension in the remote Asterisk server
<context>	Context in the remote Asterisk server
<options>	The only option available is 'a' meaning 'request autoanswer'

Outbound channels example:

Outbound channels are seen in the Asterisk console.

IAX2/8590:secret@myserver/8590@default	Call the 8590 extension in myserver. It uses 8590:secret as the name/password pair
--	--

IAX2/iaxphone	Call "iaxphone"
IAX2/judy:[judyrsa]@somewhere.com	Call somewhere.com using judy as the username and a RSA key for authentication

The format of an incoming IAX channel is:

Inbound channels are seen in the Asterisk console.

```
IAX2/[<username>@]<host>]-<callno>
```

<username>	Username if known
<host>	Host connecting
<callno>	Local call number

Incoming channel example:

IAX2[flavio@8.8.30.34]/10	Call number 10 from IP address 8.8.30.34 using flavio as the user.
IAX2[8.8.30.50]/11	Call number 11 from IP address 8.8.30.50.

Using IAX

You may use IAX in several ways. In this section, we will show you how to configure IAX for several scenarios, including:

- Connecting a soft-phone using IAX
- Connecting IAX to a VoIP provider using IAX
- Connecting two servers using IAX
- Connecting two servers using IAX in trunk mode
- Debugging an IAX connection
- Using RSA pair keys for authentication

Connecting a soft-phone using IAX

Asterisk supports IP phones based on IAX such as the ATCOM and the old ATA from Digium (called IAXy) as well as soft-phones such as Zoiper. The process for soft-phones, ATAs, and hard-phones is similar. To configure an IAX device, you need to edit the `iax.conf` file in `/etc/asterisk` directory.

We will use the Zoiper (www.zoiper.com) as an example. It is a full-featured and free soft-phone.

Step 1: Make a backup of the original `iax.conf` file using:

```
#cd /etc/asterisk
#mv iax.conf iax.conf.backup
```

Step 2: Start editing a new `iax.conf` file:

```
[general]
bindport=4569
bindaddr=8.8.1.4
bandwidth=high      ; Very important parameter, it changes the codecs available
disallow=all
allow=ulaw
jitterbuffer=no
forcejitterbuffer=no
tos=lowdelay
autokill=yes

[guest]
type=user
context=guest
callerid="Guest IAX User"

; Trust Caller*ID Coming from iaxtel.com
;
[iaxtel]
type=user
context=default
auth=rsa
inkeys=iaxtel

;
; Trust Caller*ID Coming from iax.fwdnet.net
;
[iaxfwd]
type=user
context=default
auth=rsa
inkeys=freeworlddialup

;
; Trust callerid delivered over DUNDi/e164
;
;
;[dundi]
;type=user
;dbsecret=dundi/secret
;context=dundi-e164-local

[2003]
type=friend
context=default
secret=senha
```

```
host=dynamic
```

I've tried to preserve the default (non-commented) lines of the sample file. The following parameters were modified:

```
bandwidth=high
```

This line affects the codec selection. Using the **high** setting allows for the selection of a high bandwidth and a high quality codec such as g.711 defined by the **ulaw** keyword. If you keep the default parameter, you will not be able to choose **ulaw**. In this case, Asterisk will give you the message “no codec available” for the configuration below.

```
disallow=all
allow=ulaw
```

In the commands described above, we disabled all codecs and enabled just **ulaw**. In LANs, most people prefer to use **ulaw** because it is not processor-intensive and saves CPU cycles. Even using more bandwidth, this codec is preferable because in LANs you usually have a 100-megabits Ethernet or even a Gigabit. A voice call using **ulaw** uses almost 100 kilobits per second of bandwidth from your network, which is a very light use for today's high-speed LANs. In WAN or Internet networks, you will usually disable **ulaw**, trading some available CPU cycles by voice compression for better bandwidth use. The codecs **gsm**, **g729**, and **ilbc** provide a good compression factor as well.

```
[2003]
type=friend
context=default
secret=senha
host=dynamic
```

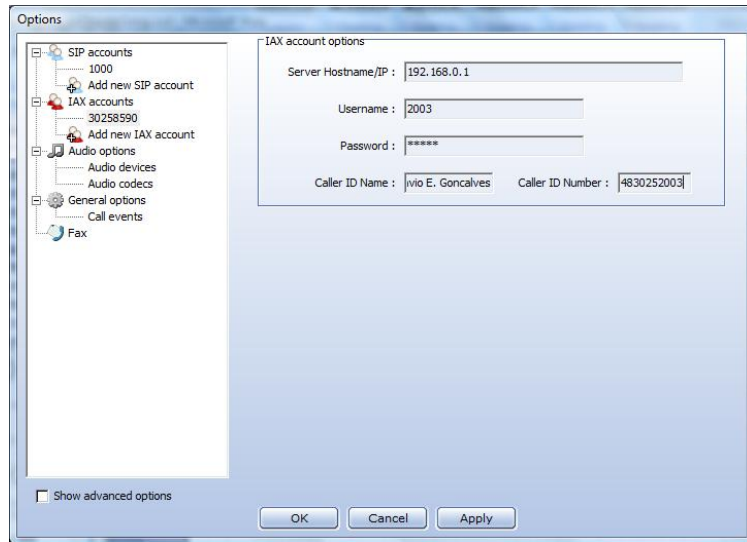
In the above commands, we have defined a friend named [2003]. The context is the default (in the first labs we always use the default context to avoid confusion; this context will be fully explained in chapter 9). The line “host=dynamic” provides a dynamic registration of the phone's IP address.

Step 3: Download and install Zoiper™ from the following URL:

<http://www.zoiper.com/>

Note: URLs frequently change. Please resort to “googling” if you cannot find the file at this specific URL. You can choose other soft-phones for the lab as well.

Step 4: Configure an Asterisk account by clicking the right button over the Zoiper. You should see a screen similar to the one below:



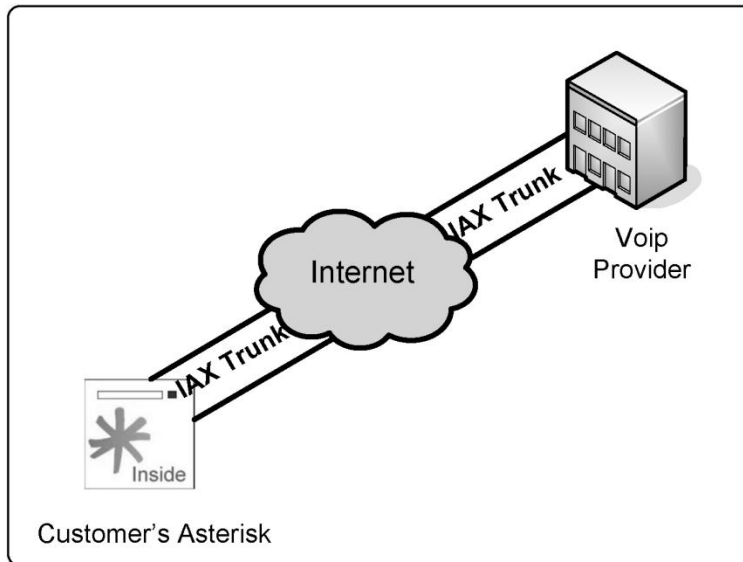
Step 5: Configure the `extensions.conf` file to test your IAX device.

```
[default]
exten=>2000,1,Dial(SIP/2000)
exten=>2001,1,Dial(SIP/2001)
exten=>2003,1,Dial(IAX2/2003)
```

Now you can dial between the SIP phones created in Chapter 3 and the IAX phone created in the lab.

Connecting to a VoIP provider using IAX

A few VoIP providers support IAX. You can easily find an IAX provider by “googling” the words “IAX providers”. Using an IAX provider makes a lot of sense as IAX can save a lot of bandwidth, easily traverses NAT, and can authenticate using RSA key pairs.



Connecting to a provider using IAX

Step 1: Open an account in your favorite provider. Your provider will provide you three things.

- Name
- Secret
- IP address or Host name
- RSA public key

Step 2: Configure the `iax.conf` file to register your Asterisk with your provider. Add the following lines to the `[general]` section of the file.

```
[general]
register=>name:secret@hostname/2003
```

In the instructions described above, you registered with your provider using your account and password. The moment you receive a call, it will be forwarded to the 2003 extension.

```
[name]          ; Your account name or number
type=peer
secret=secret   ; Your password
host=hostname
```

In the instructions described above, we have created a peer corresponding to the provider for dialing purposes.

```
[nameiax]
type=user
context=default
auth=rsa
inkeys=hostname
```

This is required for RSA authentication. Using the public key from your provider allows you to be sure that the call being received is really from the true provider. If anyone else tries to use the same path, they will not be able to authenticate it because they do not have the corresponding private key.

Step 4: Try the connection.

To test the connection, call any number. Some vendors provide an echo test. To accomplish this, please edit the file `extensions.conf`.

```
[default]
exten=>*98,1,Dial(IAX2/name:secret@hostname/*98,20,r)
```

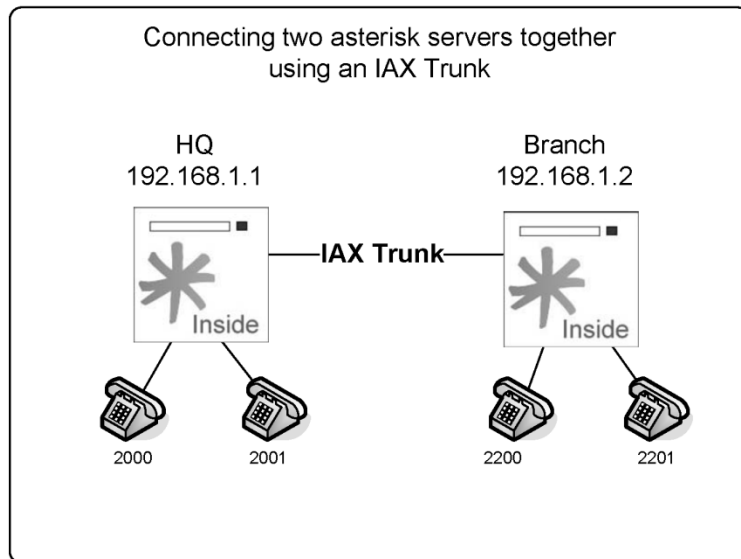
Go to the Asterisk CLI and issue a reload. To verify if Asterisk is registered with the provider, use the next command.

```
CLI>reload
CLI>iax2 show register
```

Now simply dial *98 on the soft-phone connected to the Asterisk server.

Connecting two Asterisk servers through an IAX trunk

It is very easy to connect one server to another. You won't need to register them because the IP addresses are already known.



You will have to create the peers and users in the `iax.conf` file. All extensions in the HQ site start with 20 followed by two digits (e.g., 2000). In the Branch, all extensions start with 22 followed by two digits (e.g., 2200). We will use the trunk. You will need a DAHDI timing source to enable this feature.

Step 1: Edit the `iax.conf` file in the Branch server.

```
[general]
bindport=4569                ; bindport and bindaddr may be specified
bindaddr=0.0.0.0            ; more than once to bind to multiple
disallow=all
allow=ulaw
;allow=gsm
```

```
[Branch]
type=user
context=default
secret=password
host=192.168.2.10
trunk=yes
notransfer=yes
```

```
[HQ]
type=peer
context=default
username=HQ
secret=password
host=192.168.2.10
callerID='HQ'
trunk=yes
notransfer=yes
```

```
[2200]
type=friend
auth=md5
context=default
secret=password
host=dynamic
callerid='2000'
```

```
[2201]
type=friend
auth=md5
context=default
secret=password
host=dynamic
callerid='2001'
```

Step 2: Configure the file `extensions.conf` in the Branch server

```
[general]
static=yes
writeprotect=no
autofallthrough=yes
clearglobalvars=no
priorityjumping=no
```

```
[default]
exten=>_20XX,1,dial(IAX2/HQ/${EXTEN},20)
exten=>_20XX,2,hangup

exten=>_22XX,1,dial(IAX2/${EXTEN},20)
exten=>_22XX,2,hangup
```

Step 3: Configure the **iax.conf** file in the HQ server

```
[general]
bindaddr=0.0.0.0
bindport=4569
disallow=all
allow=ulaw
allow=gsm

[Branch]
type=peer
context=default
username=Branch
secret=password
host=192.168.2.9
callerid="Branch"
trunk=yes
notransfer=yes

[HQ]
type=user
secret=password
context=default
host=192.168.2.9
callerid="HQ"
trunk=yes
notransfer=yes

[2000]
type=friend
auth=md5
context=default
secret=password
callerid="2200"
host=dynamic

[2001]
type=friend
auth=md5
context=default
secret=password
```

```
callerid="2201"  
host=dynamic
```

Step 4: Configure the `extensions.conf` file in the HQ server.

```
[general]  
static=yes  
writeprotect=no  
autofallthrough=yes  
clearglobalvars=no  
priorityjumping=no  
  
[default]  
exten=>_22XX,1,Dial(IAX2/Branch/${EXTEN})  
exten=>_22XX,2,hangup  
  
exten=>_20XX,1,Dial(IAX2/${EXTEN})  
exten=>_20XX,2,hangup
```

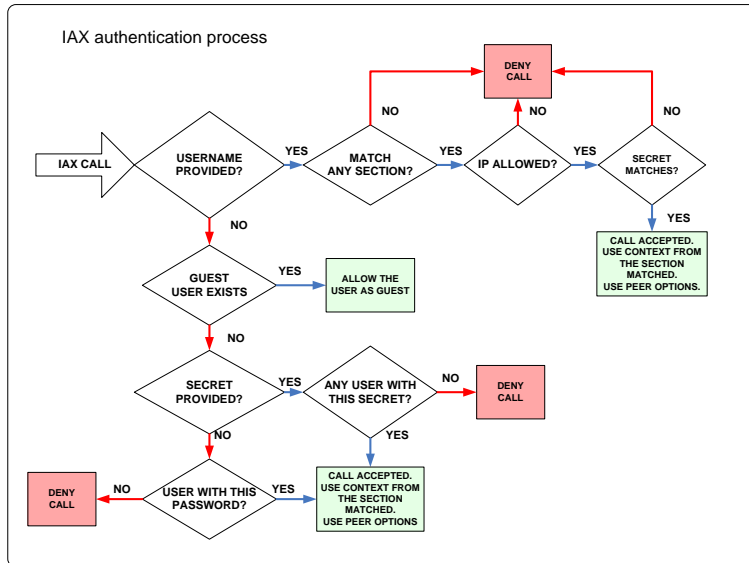
Step 5: Test a call from the phone 2000 in the HQ server to the phone 2200 in the Branch server.

IAX authentication

Now let's analyze the IAX authentication process from the practical standpoint to help you choose the best method for each specific requirement.

Incoming connections

When Asterisk receives an incoming connection, the initial information can include a user name (from the field "username=") or not. The incoming connection has an IP address too, which Asterisk uses for authentication as well.



If a user is provided, Asterisk:

1. Searches `iax.conf` for an entry with `type=user` (or `type=friend` with a section name matching the username). If it did not find it, Asterisk refuses the connection.
2. If the entry found has `deny/allow` configurations, it compares the IP address from the caller to determine whether to accept the call or not depending on the deny/allow clauses.
3. It checks the password (secret) using `plaintext`, `md5`, or `RSA`.
4. It accepts the connection and sends the call to the context specified in the line “context=” from the `iax.conf` file.

If a username is not provided, Asterisk:

1. Searches for an entry containing `type=user` (or `type=friend`) in the `iax.conf` file without a specified secret. It checks deny/allow clauses as well. If an entry is found, the connection is accepted and the section name is used as the user’s name.
2. Searches for an entry containing `type=user` (or `type=friend`) in the `iax.conf` file with a secret or RSA key specified. It checks `deny/allow` clauses. If an entry is found, it tries to authenticate the caller using the specified secret; if it matches, it accepts the connection. Section name is the user’s name.

Let’s suppose your `iax.conf` file has the following entries:

```

[guest]
type=user
context=guest

[ixte1]
type=user
context=incoming
auth=rsa
  
```

```

inkeys=iaxtel

[iax-gateway]
type=friend
allow=192.168.0.1
context=incoming
host=192.168.0.1

[iax-friend]
type=user
secret=this_is_secret
auth=md5
context=incoming

```

If a call has a specified username, such as:

- guest
- iaxtel
- iax-gateway
- iax-friend

Asterisk will try to authenticate the call using only the corresponding entry in the `iax.conf` file. If any other names are specified, the call would be rejected.

If no user is specified, Asterisk will try to authenticate the connection as guest. However, if guest does not exist, it will try any other connections with a matching secret. In other words, if you don't have a guest section in your `iax.conf` file, a malicious user could try to guess any matching secret by not specifying the user name. IP addresses' deny/allow restrictions apply too.

A good way to avoid secret guessing is to use RSA authentication. Another method is to restrict the IP addresses allowed to call in.

IP address restrictions

```

permit = <ipaddr>/<netmask>
deny = <ipaddr>/<netmask>

```

Rules are interpreted in sequence, and all are evaluated (this concept is different from ACLs usually found in routers and firewalls).

Example #1

```
permit=0.0.0.0/0.0.0.0
```

```
deny=192.168.0.0/255.255.255.0
```

Will deny any packet from 192.168.0.0/24 network

Example #2

```
deny=192.168.0.0/255.255.255.0
```

```
permit=0.0.0.0/0.0.0.0
```

It will permit any packet. The last instruction supersedes the first.

Outbound connections

Outbound connections acquire authentication information using the following methods:

- The IAX2 channel description passed by the `dial()` application.
- An entry with `type=peer` or `type=friend` in the `iax.conf` file.
- A combination of both methods.

Connecting two Asterisk servers using RSA keys

It is possible to use IAX with strong authentication using asymmetric RSA keys. According to the source code (`res_krypto.c`), Asterisk uses RSA keys with an SHA-1 algorithm for message digests instead of the weaker MD5. Below is a step-by-step guide for setting up two servers using RSA keys.

Configuring the server for the branch

Step 1: Generate the RSA keys in the branch server

```
astkeygen -n
```

When asked, use the key name `branch`. We have used the parameter `-n` to avoid passing a passphrase whenever Asterisk reinitializes. If you want to improve the security, don't use the `-n` and start Asterisk with `asterisk -i`

Step 2: Copy the keys to the directory `/var/lib/asterisk/keys`

```
cp branch.* /var/lib/asterisk/keys
```

Step 3: Copy the public key to the HQ server

```
scp branch.pub root@hq_ip_address:/var/lib/asterisk/keys
```

Step 4: Edit the `iax.conf` file in the Branch server.

```
[general]
bindport=4569                ; bindport and bindaddr may be specified
bindaddr=0.0.0.0            ; more than once to bind to multiple
disallow=all
allow=ulaw

;Create an entry for the HQ server
[hq]
type=user
context=default
host=192.168.2.10
trunk=yes
notransfer=yes
auth=rsa
inkeys=hq
```

```
[2200]
type=friend
auth=md5
context=default
secret=password
host=dynamic
callerid='2200'
```

```
[2201]
type=friend
auth=md5
context=default
secret=password
host=dynamic
callerid='2201'
```

Step 8: Configure the `extensions.conf` file in the Branch server

```
[default]
exten=>_20XX,1,dial(IAX2/branch:[branch]@192.168.2.10/${EXTEN},20)
exten=>_20XX,2,hangup
exten=>_22XX,1,dial(IAX2/${EXTEN},20)
exten=>_22XX,2,hangup
```

Configuring the server for the headquarters

Step 1: Generate the RSA keys in the HQ server

```
astkeygen -n
```

When asked use the key name `hq`.

Step 2: Copy the keys to the directory `/var/lib/asterisk/keys`

```
cp hq.* /var/lib/asterisk/keys
```

Step 3: Copy the public key to the BRANCH server

```
scp hq.pub root@branch_ip_address:/var/lib/asterisk/keys
```

Step 4: Configure the `iax.conf` file in the HQ server

```
[general]
bindaddr=0.0.0.0
bindport=4569
disallow=all
allow=ulaw
allow=gsm
```

```
;Configure an entry for the branch server
```

```
[branch]
type=user
```

```
context=default  
host=192.168.2.9  
trunk=yes  
nottransfer=yes  
auth=rsa  
inkeys=branch
```

```
[2000]  
type=friend  
auth=md5  
context=default  
secret=password  
callerid="2000"  
host=dynamic
```

```
[2001]  
type=friend  
auth=md5  
context=default  
secret=password  
callerid="2001"  
host=dynamic
```

Step 10: Configure the `extensions.conf` file in the HQ server.

```
[default]  
exten=>_22XX,1,Dial(IAX2/hq:[hq]@192.168.2.9/${EXTEN})  
exten=>_22XX,2,hangup  
exten=>_20XX,1,Dial(IAX2/${EXTEN})  
exten=>_20XX,2,hangup
```

Step 11: Test a call from the 2000 phone in the HQ server to the 2200 phone in the Branch server.

The iax.conf file configuration

The file `iax.conf` has several parameters; discussing each parameter one by one would be boring and counterproductive. All parameters, along with a description, can be found in the sample file. In the wiki www.voip-info.org you will find detailed information about each one. Here we will show some of the most important parameters for the configuration of the general section, peers, and users.

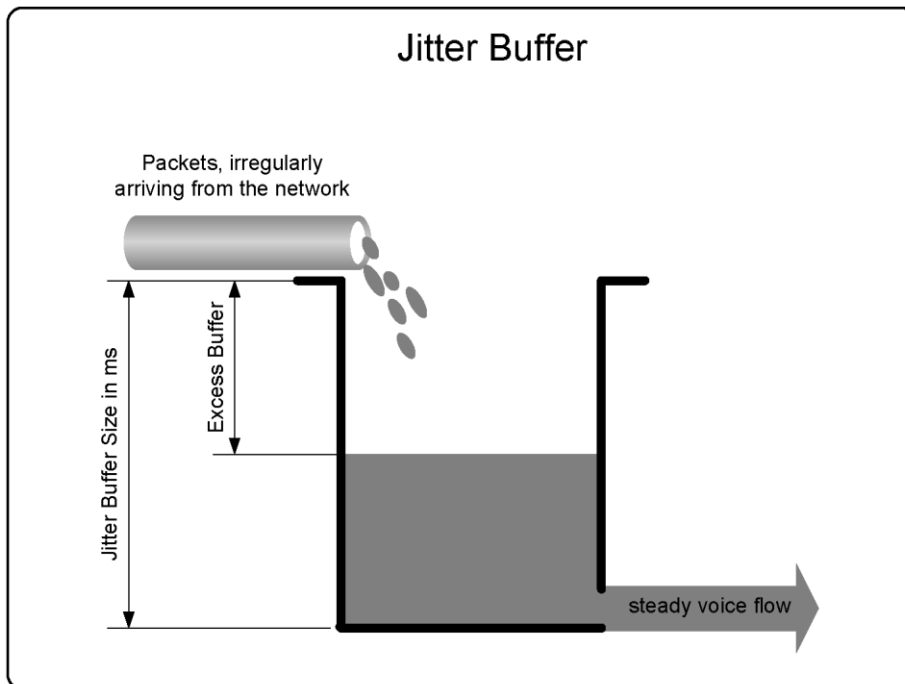
[General] Section

Server addresses	
<code>bindport = <portnum></code>	Configures the IAX UDP port. Default is 4569.
<code>bindaddr = <ipaddr></code>	Use 0.0.0.0 to bind Asterisk to all interfaces or specify the IP address of a

	specific interface.
Codec selection	
bandwidth = [low medium high]	High = all codecs Medium = all codecs except ulaw and alaw Low = low bandwidth codecs
allow/disallow = [alaw ulaw gsm g.729 etc.]	Codec selection fine tuning

Jitter buffer

Jitter is the delay variation between packets. It is the most important factor affecting voice quality. A Jitter buffer is used to compensate for the delay variation. It sacrifices latency in favor of lower jitter. You can make an analogy between the jitter buffer and a water tank. Both can receive packets or water at irregular intervals, but will ultimately deliver a regular flow.



A small jitter (i.e., below 20 ms) is usually imperceptible. However, jitter above this level is annoying. The latency or delay should be kept to below 150ms. Creating a jitter buffer will sacrifice some delay for a lower jitter—a concept known as “delay-budget”.

You can affect the jitter buffer using these parameters:

- **Jitterbuffer=<yes/no>** - Enables or disables
- **Dropcount=<number>** - Maximum amount of frames that should be delayed in the last two seconds. The recommended setting is 3 (1.5% of dropped frames)
- **Maxjitterbuffer=<ms>** - Usually below 100 ms
- **Maxexcessbuffer=<ms>** - If the network delay improves, the jitter buffer could be oversized. Consequently, Asterisk will try to reduce it.
- **Minexcessbuffer=<ms>** - Once the excess buffer drops to this value, Asterisk starts to increase the buffer size.

Frame tagging

The parameter below marks the IP packet in the type of service field. Routers can read this tag, thereby prioritizing traffic. In version 1.4, Asterisk uses DSCP codes for this field (RFC2474). Allowed values are CS0, CS1, CS2, CS3, CS4, CS5, CS6, CS7, AF11, AF12, AF13, AF21, AF22, AF23, AF31, AF32, AF33, AF41, AF42, AF43, and ef (i.e., expedited forwarding).

```
tos=ef
```

IAX2 Encryption

According to Spencer, Guy, Capouch, Muller, and Shumard (2008), IAX supports call encryption using a symmetric key, 128-bit block cipher called AES-Advanced Encryption Standard. It is very simple to activate the encryption between IAX trunks. In the file `iax.conf` use:

```
encryption=yes
```

To force the encryption:

```
forceencryption=yes
```

To guarantee compatibility with old versions, you may need to impede key rotation using:

```
keyrotate=no
```

IAX2 debug commands

Below are some of the most important troubleshooting console commands for Asterisk.

`iax2 show netstats`

```
vtsvooffice*CLI> iax2 show netstats
----- LOCAL ----- REMOTE -----
Channel      RTT  Jit  Del  Lost  %  Drop  OOO  Kpkts  Jit  Del  Lost  %  Drop  OOO
Kpkts
IAX2/8590-1  16  -1   0   -1  -1   0   -1    1   60  110   3   0   0   0
0
```

iax2 show channels

vtsvoffice*CLI> iax2 show channels

Channel Format	Peer	Username	ID (Lo/Rem)	Seq (Tx/Rx)	Lag	Jitter	JitBuf
IAX2/8590-2 unknown	8.8.30.43	8590	00002/26968	00004/00003	00000ms	-0001ms	0000ms

iax2 show peers

vtsvoffice*CLI> iax2 show peers

Name/Username	Host	Mask	Port	Status
8584	(Unspecified)	(D) 255.255.255.255	0	UNKNOWN
8564	(Unspecified)	(D) 255.255.255.255	0	UNKNOWN
8576	(Unspecified)	(D) 255.255.255.255	0	UNKNOWN
8572	(Unspecified)	(D) 255.255.255.255	0	UNKNOWN
8571	(Unspecified)	(D) 255.255.255.255	0	UNKNOWN
8585	(Unspecified)	(D) 255.255.255.255	0	UNKNOWN
8589	(Unspecified)	(D) 255.255.255.255	0	UNKNOWN
8590	8.8.30.43	(D) 255.255.255.255	4569	OK (16 ms)
3232	(Unspecified)	(D) 255.255.255.255	0	UNKNOWN

9 iax2 peers [1 online, 8 offline, 0 unmonitored]

iax2 debug

Looking at this output, identify the beginning and end of the call. Observe the delay and jitter information obtained using poke and pong packets. These packets help create the output of the “iax2 show netstats” command.

vtsvoffice*CLI> iax2 debug

IAX2 Debugging Enabled

```
Rx-Frame Retry[ No] -- OSeqno: 000 ISeqno: 000 Type: IAX      Subclass: REGREQ
Timestamp: 00003ms  SCall: 26975  DCall: 00000 [8.8.30.43:4569]
USERNAME           : 8590
REFRESH            : 60
```

```
Tx-Frame Retry[000] -- OSeqno: 000 ISeqno: 001 Type: IAX      Subclass: REGAUTH
Timestamp: 00009ms  SCall: 00003  DCall: 26975 [8.8.30.43:4569]
AUTHMETHODS        : 2
CHALLENGE           : 137472844
USERNAME            : 8590
```

```
Rx-Frame Retry[ No] -- OSeqno: 001 ISeqno: 001 Type: IAX      Subclass: REGREQ
Timestamp: 00016ms  SCall: 26975  DCall: 00003 [8.8.30.43:4569]
USERNAME           : 8590
REFRESH            : 60
MD5 RESULT          : f772b6512e77fa4a44c2f74ef709e873
```

```
Tx-Frame Retry[000] -- OSeqno: 001 ISeqno: 002 Type: IAX      Subclass: REGACK
Timestamp: 00025ms  SCall: 00003  DCall: 26975 [8.8.30.43:4569]
USERNAME           : 8590
DATE TIME          : 2006-04-17 16:03:00
REFRESH            : 60
APPARENT ADDRES    : IPV4 8.8.30.43:4569
CALLING NUMBER     : 4830258590
CALLING NAME       : Flavio
```

```
Rx-Frame Retry[ No] -- OSeqno: 002 ISeqno: 002 Type: IAX      Subclass: ACK
Timestamp: 00025ms  SCall: 26975  DCall: 00003 [8.8.30.43:4569]
```

| IAX2 debug commands |

```
Tx-Frame Retry[000] -- OSeqno: 000 ISeqno: 000 Type: IAX Subclass: POKE
Timestamp: 00003ms SCall: 00006 DCall: 00000 [8.8.30.43:4569]
Rx-Frame Retry[ No] -- OSeqno: 000 ISeqno: 001 Type: IAX Subclass: ACK
Timestamp: 00003ms SCall: 26976 DCall: 00006 [8.8.30.43:4569]
Rx-Frame Retry[ No] -- OSeqno: 000 ISeqno: 001 Type: IAX Subclass: PONG
Timestamp: 00003ms SCall: 26976 DCall: 00006 [8.8.30.43:4569]
RR_JITTER : 0
RR_LOSS : 0
RR_PKTS : 1
RR_DELAY : 40
RR_DROPPED : 0
RR_OUTOFORDER : 0
Tx-Frame Retry[-01] -- OSeqno: 001 ISeqno: 001 Type: IAX Subclass: ACK
Timestamp: 00003ms SCall: 00006 DCall: 26976 [8.8.30.43:4569]

Rx-Frame Retry[ No] -- OSeqno: 000 ISeqno: 000 Type: IAX Subclass: NEW
Timestamp: 00003ms SCall: 26977 DCall: 00000 [8.8.30.43:4569]
VERSION : 2
CALLING NUMBER : 8590
CALLING NAME : 4830258590
FORMAT : 2
CAPABILITY : 1550
USERNAME : 8590
CALLED NUMBER : 8580
DNID : 8580

Tx-Frame Retry[000] -- OSeqno: 000 ISeqno: 001 Type: IAX Subclass: AUTHREQ
Timestamp: 00007ms SCall: 00004 DCall: 26977 [8.8.30.43:4569]
AUTHMETHODS : 2
CHALLENGE : 190271661
USERNAME : 8590

Rx-Frame Retry[Yes] -- OSeqno: 000 ISeqno: 000 Type: IAX Subclass: NEW
Timestamp: 00003ms SCall: 26977 DCall: 00000 [8.8.30.43:4569]
VERSION : 2
CALLING NUMBER : 8590
CALLING NAME : 4830258590
FORMAT : 2
CAPABILITY : 1550
USERNAME : 8590
CALLED NUMBER : 8580
DNID : 8580

Tx-Frame Retry[-01] -- OSeqno: 000 ISeqno: 001 Type: IAX Subclass: ACK
Timestamp: 00003ms SCall: 00004 DCall: 26977 [8.8.30.43:4569]
Rx-Frame Retry[ No] -- OSeqno: 001 ISeqno: 001 Type: IAX Subclass: AUTHREP
Timestamp: 00063ms SCall: 26977 DCall: 00004 [8.8.30.43:4569]
MD5 RESULT : 57cc5c48affba14106c29439944413a1

Tx-Frame Retry[000] -- OSeqno: 001 ISeqno: 002 Type: IAX Subclass: ACCEPT
Timestamp: 00054ms SCall: 00004 DCall: 26977 [8.8.30.43:4569]
FORMAT : 1024

Tx-Frame Retry[000] -- OSeqno: 002 ISeqno: 002 Type: CONTROL Subclass: ANSWER
Timestamp: 00057ms SCall: 00004 DCall: 26977 [8.8.30.43:4569]
Tx-Frame Retry[000] -- OSeqno: 003 ISeqno: 002 Type: VOICE Subclass: 138
Timestamp: 00090ms SCall: 00004 DCall: 26977 [8.8.30.43:4569]
Rx-Frame Retry[ No] -- OSeqno: 002 ISeqno: 002 Type: IAX Subclass: ACK
Timestamp: 00054ms SCall: 26977 DCall: 00004 [8.8.30.43:4569]
Rx-Frame Retry[ No] -- OSeqno: 002 ISeqno: 003 Type: IAX Subclass: ACK
Timestamp: 00057ms SCall: 26977 DCall: 00004 [8.8.30.43:4569]
```

```

Rx-Frame Retry[ No] -- OSeqno: 002 ISeqno: 004 Type: IAX      Subclass: ACK
Timestamp: 00090ms SCall: 26977 DCall: 00004 [8.8.30.43:4569]
Rx-Frame Retry[ No] -- OSeqno: 002 ISeqno: 004 Type: VOICE    Subclass: 138
Timestamp: 00210ms SCall: 26977 DCall: 00004 [8.8.30.43:4569]
Tx-Frame Retry[-01] -- OSeqno: 004 ISeqno: 003 Type: IAX      Subclass: ACK
Timestamp: 00210ms SCall: 00004 DCall: 26977 [8.8.30.43:4569]
Rx-Frame Retry[ No] -- OSeqno: 003 ISeqno: 004 Type: IAX      Subclass: PING
Timestamp: 02083ms SCall: 26977 DCall: 00004 [8.8.30.43:4569]
Tx-Frame Retry[000] -- OSeqno: 004 ISeqno: 004 Type: IAX      Subclass: PONG
Timestamp: 02083ms SCall: 00004 DCall: 26977 [8.8.30.43:4569]
RR_JITTER          : 0
RR_LOSS            : 0
RR_PKTS            : 1
RR_DELAY           : 40
RR_DROPPED         : 0
RR_OUTOFORDER      : 0

Rx-Frame Retry[ No] -- OSeqno: 004 ISeqno: 005 Type: IAX      Subclass: ACK
Timestamp: 02083ms SCall: 26977 DCall: 00004 [8.8.30.43:4569]
Rx-Frame Retry[ No] -- OSeqno: 004 ISeqno: 005 Type: IAX      Subclass: HANGUP
Timestamp: 08693ms SCall: 26977 DCall: 00004 [8.8.30.43:4569]
CAUSE              : Dumped Call

```

To turn off debugging, use:

```
vtsvoffice*CLI>iax2 no debug
```

Summary

This chapter has reviewed the strengths and weaknesses of the IAX protocol. It has demonstrated how IAX works in several scenarios, such as soft-phones and a trunk between two Asterisk servers. Trunk mode allows you to save bandwidth by carrying more than one call in a single packet. Finally, you learned console commands that you can use to check the status and debug the protocol.

Quiz

- Two of the main benefits of IAX are bandwidth savings and easier NAT traversal.
 - False
 - True
- IAX protocols use different UDP ports for signaling and media.
 - False
 - True
- The bandwidth used by the IAX protocol is the voice payload plus the following headers (mark all that apply):
 - IP
 - UDP

- C. IAX
- D. RTP
- E. cRTP

4. It is important to match the codec payload (20 to 30 ms) with frame synchronization (20ms default) when using trunk mode.

- A. False
- B. True

5. When IAX is used in trunk mode, just one header is used for multiple calls.

- A. False
- B. True

6. IAX is the most used protocol to connect to service providers because it is easier for NAT traversal.

- A. False
- B. True

7. In an IAX channel as shown below, the option <secret> can be a password or a _____.

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

8. The IAX2 show registry provides information about:

- A. Registered users
- B. Providers to which Asterisk is connected

9. Jitter buffer sacrifices latency to have a steady flow of voice.

- A. False
- B. True

10. RSA keys might be used for IAX authentication. You have to keep the _____ key secret and give your customers the matching _____ key.

- A. public, private
- B. private, public
- C. shared, private
- D. public, shared

8

The SIP Protocol

Session Initiated Protocol (SIP) is a text-based protocol similar to HTTP and SMTP that was designed to initialize, keep, and terminate interactive communication sessions between users. These sessions may include voice, video, chat, interactive games, and others. SIP was defined by the IETF and is becoming the de facto standard for voice communications. It is very important to understand how SIP works and how it is configured. If you are going to work with Asterisk, the file `sip.conf` will probably be the second most changed file (just after the file `extension.conf`).

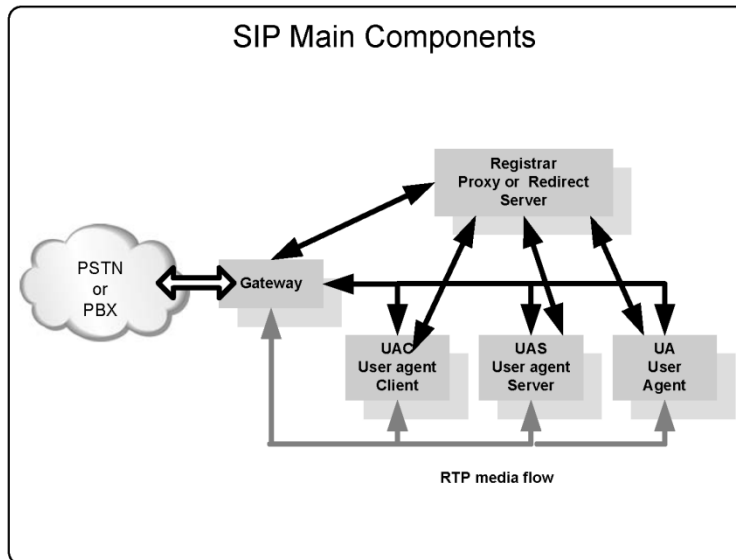
Objectives

By the end of this chapter, you will be able to:

- Understand SIP theory of operation
- Understand the strengths and weaknesses of SIP
- Configure Asterisk to connect to a SIP provider
- Integrate two Asterisk servers using SIP
- Configure Asterisk in a NAT scenario
- Configure a client behind NAT

Theory of Operation

SIP is a signaling protocol with the following components: User Agent Client, User Agent Servers, SIP Proxies, and SIP Gateways. The following figure depicts the relationships among these components.

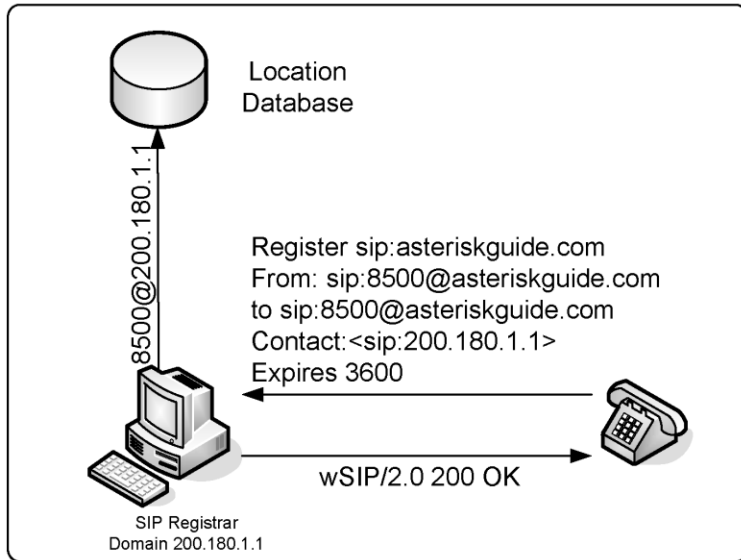


- **UAC (user agent client)** – The client or terminal that initializes SIP signaling.
- **UAS (user agent server)** – The server that responds to a SIP signaling coming from a UAC.
- **UA (user agent)** – The SIP terminal (phones or gateways that contain both UAC and UAS).
- **Proxy Server** – Receives requests from a UA and transfers to other SIP Proxies if the particular station is not under their administration.
- **Redirect Server** – Receives requests and sends them back to the UA, including destination data, instead of directly forwarding them to the destination.
- **Location Server** – Receives requests from a UA and updates the location database with this information.

Usually, the proxy, redirect, and location servers are hosted within the same hardware and use the same piece of software, which we call the SIP proxy. The SIP proxy is responsible for location database maintenance, connection establishment, and session termination.

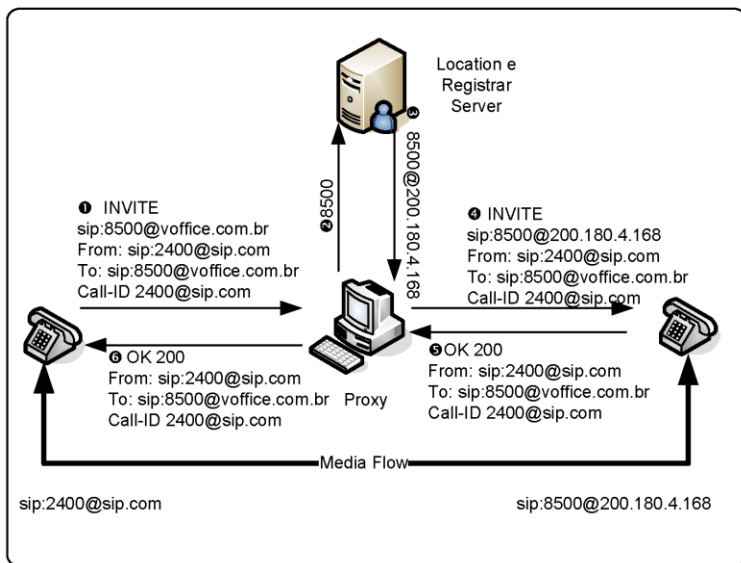
SIP Register process

Before a phone can receive calls, it needs to be registered to a location database. In the location database, the IP address will be bonded to the name. In the following example, extension 8500 will be bound to IP address 200.180.1.1. You do not necessarily need to use phone numbers. In the SIP architecture, the registered extension could be flavio@asteriskguide.com as well.



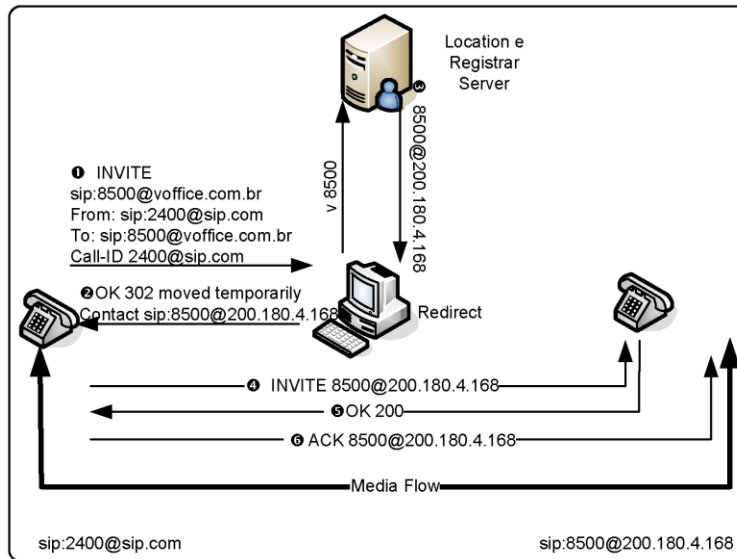
Proxy operation

When operating as a SIP proxy, the SIP server stays in the middle of the signaling and is capable of advanced routing and billing. The media flow, based on the real time protocol (RTP) still goes directly between the endpoints.



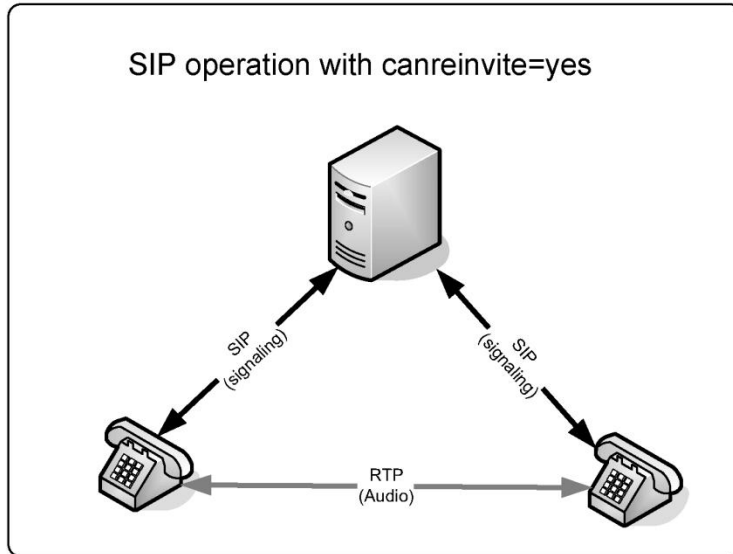
Redirect operation

When redirecting, the SIP server simply sends a message (e.g., 302 moved temporarily) to the user agent and stays out of the path of new messages. It is very light in terms of resource usage, but you have no control at all. Redirection is sometimes used in load balance designs.

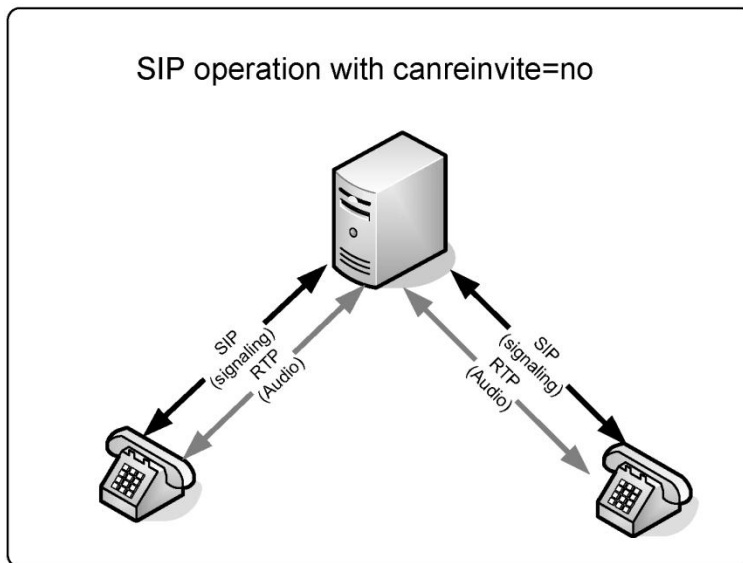


How Asterisk handles SIP

It is important to understand that Asterisk is neither a SIP proxy nor a SIP redirector. Asterisk can perform the role of the registrar and location server; however, it only connects two UACs to itself. Therefore, Asterisk is considered a back-to-back user agent (B2BUA). In other words, it connects two SIP channels, bridging them together. Asterisk has a re-invite mechanism that can make the SIP channels talk to each other directly instead of passing through Asterisk. This mechanism is controlled by the parameter `canreinvite` in the `sip.conf` file. When using `canreinvite=yes` the RTP flow goes directly from one endpoint to another, freeing server resources.



However, if you need to transfer or record the call using Asterisk, you may use the parameter `canreinvite=no` to force the RTP flow through the Asterisk server.



SIP Messages

The basic SIP messages are:

- INVITE – connection establishment

- ACK – acknowledge
- BYE – connection termination
- CANCEL – connection termination for a non-established call
- REGISTER – register a UAC to a SIP proxy
- OPTIONS – can be used to check availability
- REFER – transfer a SIP call to someone else
- SUBSCRIBE – subscribe to notification events
- NOTIFY – send out channel information
- INFO – send various messages (e.g., DTMF)
- MESSAGE – send instant messages

The SIP responses are in text format and are easily readable (similar to HTTP messages). The most important responses are:

- 1XX – Information messages (100–trying, 180–ringing, 183–progress)
- 2XX – Successful request complete (200 – OK)
- 3XX – Call redirect, request has to be directed to another place (302 – moved temporarily, 305 – use proxy)
- 4XX – Error (403 – Forbidden)
- 5XX – Server Error (500 – Internal Server Error; 501 – Not implemented)
- 6XX – Global Failure (606 – Not acceptable)

For example:

```
INVITE sip:2000@192.168.1.133 SIP/2.0
Via: SIP/2.0/UDP
192.168.1.116;rport;branch=z9hG4bKc0a8017400000063452fafbb00006967000000d2
From: "unknown"<sip:2001@192.168.1.133>;tag=1556140623845
To: <sip:2000@192.168.1.133>
Contact: <sip:2001@192.168.1.116>
Call-ID: 64B4C8EC-FCFC-49E9-98B1-90982EEEBED3@192.168.1.116
CSeq: 2 INVITE
Max-Forwards: 70
User-Agent: SJphone/1.61.312b (SJ Labs)
Content-Length: 335
Content-Type: application/sdp
Proxy-Authorization: Digest
username="2001",realm="asterisk",nonce="6c55905e",uri="sip:2000@192.168.1.133",
response="983c0099eea125d8cdf93b0ec99f3ec",algorithm=MD5
```


Session description protocol (SDP)

SDP is defined in IETF RFC2327. It is intended for describing multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session initiation. SDP includes:

- Transport protocol (RTP/UDP/IP)
- Type of media (text, audio, video)
- Media format or codec (H.261 video, g.711 audio, etc.)
- Information needed to receive these media (addresses, ports, etc.)

The following example is a transcription of a SDP describing a call between two phones.

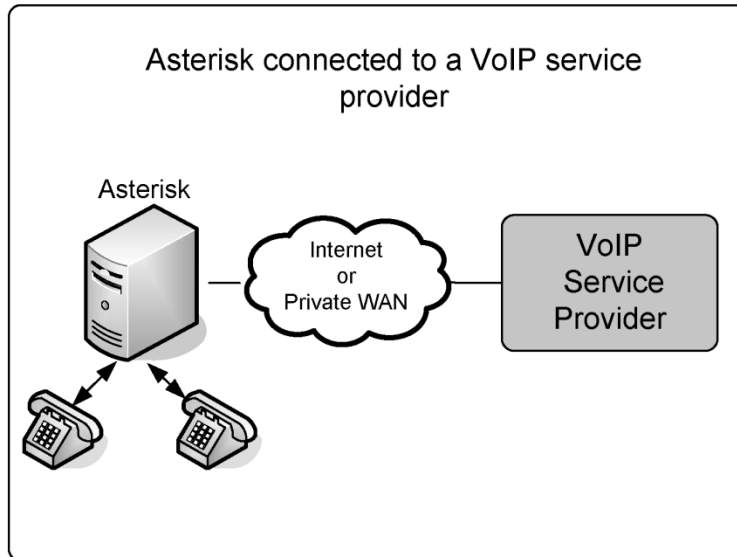
```
v=0
o=- 3369741883 3369741883 IN IP4 192.168.1.116
s=SJphone
c=IN IP4 192.168.1.116
t=0 0
a=setup:active
m=audio 49160 RTP/AVP 3 97 98 8 0 101
a=rtpmap:3 GSM/8000
a=rtpmap:97 iLBC/8000
a=rtpmap:98 iLBC/8000
a=fmtp:98 mode=20
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-11,16
```

SIP advanced scenarios

Chapter 3 reviewed the basic options to connect a SIP phone to Asterisk. Now let's move on to more advanced configurations. In the next sections, you will learn how to configure Asterisk to connect to a SIP provider, how to connect two Asterisks together using SIP, and how to place a call to a SIP provider. All SIP configurations are done in the file `/etc/asterisk/sip.conf`

Connecting Asterisk to a SIP provider

Asterisk is often used to connect to a SIP VoIP provider. VoIP providers usually have better rates for phone calls than traditional providers. Another interesting and attractive point of VoIP providers is the possibility to buy DID numbers in other cities—even in foreign countries. These are good reasons to use VoIP for telecommunications. In this section, you will learn how to connect Asterisk to a VoIP provider.



Three steps are required to connect Asterisk to a SIP provider. Tests can be conducted by establishing an account with your favorite provider.

Step 1: Registering with a SIP provider in `sip.conf`

To connect to a SIP provider, you will need the following information from the provider:

- username
- secret
- hostname
- domain
- codecs allowed

This configuration will allow your provider to locate Asterisk's IP address. In the following statement, we are telling Asterisk to register to a SIP provider defined by the *hostname* and inform the provider of Asterisk's IP address. The statement says that you want to receive calls at extension 4100.

In the `[general]` section of the `sip.conf` file, enter the following line:

```
register=>name:secret@hostname/4100
```

Step 2: Configure the `[peer]` on `sip.conf`

Create an entry of peer type to the desired provider to simplify Asterisk's dialing.

```
[provider]
context=incoming
type=friend
dtmfmode=rfc2833
canreinvite=no
username=username
```

```
secret=secret
host=hostname
fromuser=username
fromdomain=domain
insecure=invite
disallow=all
allow=ulaw ; or any other codec available from your provider
```

Step 3: Create a route to the provider in the dial plan

We will choose the digits 010 as the destination route to the provider. To dial #610000 inside the provider, simply dial 010610000.

```
exten=>_010.,1,Set(CALLERID(num)=username)
exten=>_010.,n,Set(CALLERID(Name)="Flavio Gonçalves")
exten=>_010.,n,Dial(SIP/${EXTEN:3}@provider)
exten=>_010.,n,Hangup
```

SIP options specific to the provider scenario

The following discussion examines the details of the options set in the `sip.conf` file for connection to a VoIP provider.

```
register=>username:password@hostname/4100
```

The instruction registered in the `sip.conf` file is used to register with a provider. The register transaction is authenticated with the name and secret. You can use a slash (“/”) to provide an extension for incoming calls. Technically speaking, the extension will be placed in the “Contact” header field of the SIP request.

The registering behavior can be controlled by certain parameters:

```
registertimeout=20
registerattempts=10
```

To check if registration was successful, use the following console command:

```
CLI> sip show registry
```

The parameter “username” is used in the authentication digest. The digest is computed using username, secret, and realm:

```
username=username
```

Host defines the VoIP provider address or name:

```
host=hostname
```

The parameters `Fromuser` and `Fromdomain` are sometimes required for authentication. These parameters are used in the SIP `From` header field:

```
fromuser=username
fromdomain=hostname
```

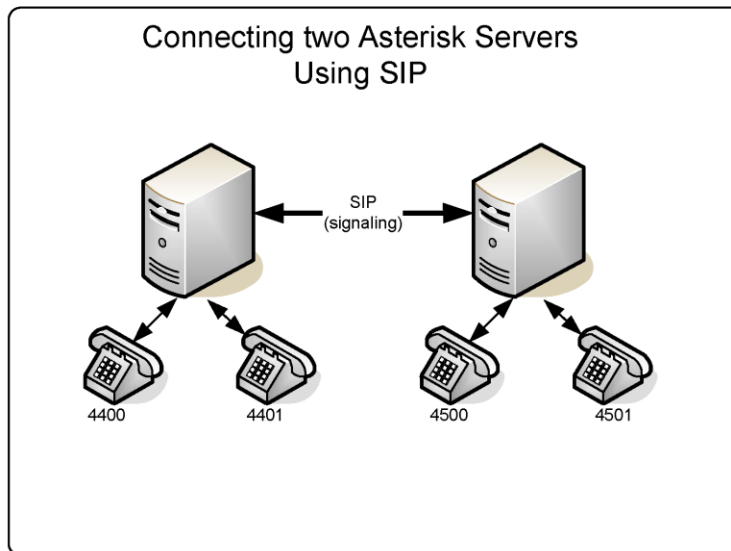
When you connect to a VoIP provider, credentials are required. After the initial invite, the provider sends you a message called “407 Proxy Authentication Required”; you provide the credentials in the subsequent INVITE message. For incoming calls, your Asterisk server will ask for credentials for the

provider. Obviously, the provider does not have a valid credential for your Asterisk server. When you use `insecure=invite`, you are telling Asterisk not to send the “407 Proxy Authentication Required” to the provider and to accept incoming calls. You can also use `insecure=port, invite` to match the peer based on the IP address without matching the port number.

```
insecure=invite, port
```

Connecting two Asterisk servers together using SIP

You can use SIP to interconnect two Asterisk boxes. It is important to pay attention to the dial plan before moving on with this configuration. Users generally want to connect other PBXs with minimal effort. The idea here is to use an extension number only to connect to the other PBX.



Step 1: Edit the `sip.conf` file in server A:

```
[B]
type=user
secret=B
host=A
disallow=all
allow=ulaw
canreinvite=no
```

```
[B-out]
type=peer
fromuser=A
username=A
secret=A
host=B
```

```
disallow=all
allow=ulaw
canreinvite=no
```

Step 2: Edit the `sip.conf` file in server B:

```
[A]
type=user
host=B
secret=A
disallow=all
allow=ulaw
canreinvite=no
```

```
[A-out]
type=peer
host=A
fromuser=B
username=B
secret=B
disallow=all
allow=ulaw
canreinvite=no
```

Step 3: Edit the `extensions.conf` file in server A:

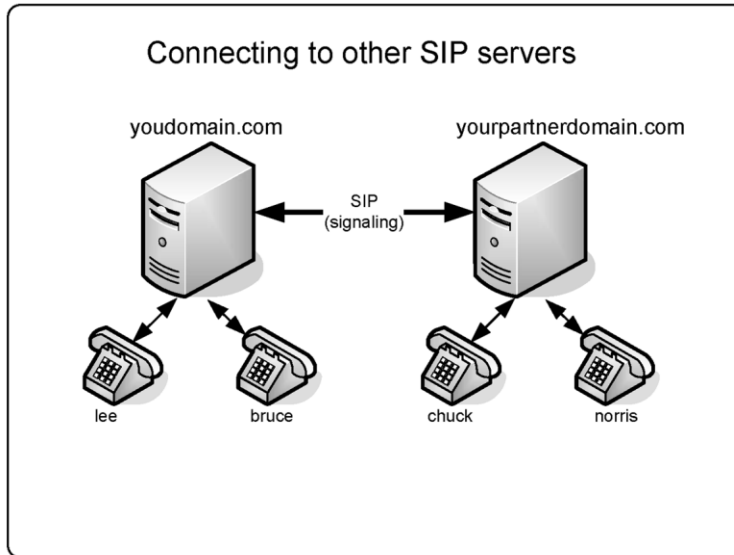
```
[default]
exten=_44XX,1,dial(SIP/${EXTEN},20)
exten=_44XX,2,hangup()
exten=_45XX,1,dial(SIP/B-out/${EXTEN})
exten=_45XX,2,hangup()
```

Step 4: Edit the `extensions.conf` file in server B:

```
[default]
exten=_44XX,1,dial(SIP/A-out/${EXTEN})
exten=_44XX,2,hangup()
exten=_45XX,1,dial(SIP/${EXTEN})
exten=_45XX,2,hangup()
```

Asterisk domain support

The SIP protocol follows the Internet architecture. The first thing to do before configuring SIP is to correctly set the DNS servers. In a SIP environment, you can call a user located in any SIP proxy, and other users can call you as well using your SIP Uniform Resource Identifier (URI). To set a DNS server for SIP, you have to add SRV records to your DNS server.



```

; SIP server/proxy and its backup server/proxy
sip1.yourdomain.com 21600 IN A 200.180.4.169
sip2.yourdomain.com 21600 IN A 200.175.61.150
;
; DNS SRV records for SIP
_sip._udp.yourdomain.com 21600 IN SRV 10 0 5060 sip1.asteriskguide.com.
_sip._udp.yourdomain.com 21600 IN SRV 20 0 5060 sip2.asteriskguide.com.

```

After configuring the DNS, you can use the URI, which points to a SIP user, SIP phone, or telephone extension. A SIP URI looks similar to an email address (e.g., `sip:chuck@yourpartnerdomain.com`). Using SIP URIs, no telephone number is needed to make a call from one SIP phone to another. To dial an external user, simply use a statement as the one shown below.

```
exten=4000,1,dial(SIP/chuck@yourpartnerdomain.com)
```

Certain parameters can control domain behavior.

```
srvlookup=yes
```

This parameter enables DNS SRV lookups on outbound calls. Using this parameter, it is not possible to dial calls using SIP names based on domain.

```
allowguest=yes
```

This parameter allows an external invite to be processed without authentication. It processes the call within the context defined in the general section or in the domain statement.

Warning: If you define a context in the general section with access to PSTN, an external user can dial the PSTN over your PBX. In this case, you will incur any charges. Allow only your own extensions in the context defined in the general section.

```
domain=acme.com,default
```

The domain command allows you to handle more than one domain within Asterisk. If a call comes from one specific domain, it is directed to a specific context.

```
;autodomain=yes
```

This parameter includes the local IP and hostname in the allowed domains.

```
;allowexternaldomains=no
```

The default is yes. Uncomment the line to disallow calls to outside domains.

Advanced configurations

This section will explain some advanced parameters of the SIP channel, such as presence, codec selection, DTMF options, and QOS packet marking.

SIP Presence

SIP presence is partially implemented in Asterisk. Asterisk supports requests such as SUBSCRIBE and NOTIFY users depending on the state of a channel. Asterisk does not support the SIP method PUBLISH. In other words, you can subscribe to the states (busy, idle, and ringing) of a channel, but cannot publish information such as “away” or “do not disturb”.

The most common scenario for presence is busy lamp field (BLF), in which you simulate the behavior of a KS system with lamps for each extension and trunk.

SIP parameters for presence:

- allowsubscribe=yes: Allow SIP subscription methods
- subscribecontext=sip_subscribers: Context where to look for hints
- notifying=yes: Send SIP NOTIFY on ring
- notifyhold=yes: Send SIP NOTIFY on hold
- counteronpeer (renamed from **limitonpeer** for Asterisk 1.4.x): Apply the counter only on the peer side
- callcounter=yes: Enable call counters in the device.
- busylevel=1: Threshold for the number of calls for considering the device as busy.

For example:

Step 1: Testing SIP presence with Asterisk is not that hard. First, let’s configure the files **sip.conf** and **extensions.conf**.

In the file **sip.conf**

```
[general]
bindaddr=0.0.0.0
```

```

bindport=5060
disallow=all
allow=ulaw
allowssubscribe=yes
notifying=yes
notifyhold=yes
limitonpeer=yes
counteronpeer=yes
subscribecontext=default

```

```

[2000]
type=friend
host=dynamic
context=default
dtmfmode=rfc2833
secret=senha
callcounter=yes
busylevel=1

```

```

[2001]
type=friend
host=dynamic
context=default
dtmfmode=rfc2833
secret=senha
callcounter=yes
busylevel=1

```

In the file **extensions.conf**

```

[default]
exten=2000, hint, SIP/2000
exten=2001, hint, SIP/2001
exten=_20XX, 1, dial(SIP/${EXTEN})
exten=_20XX, n, Hangup()

```

Step 2: Now configure the soft-phone to use presence. We will show you how to configure X-Lite.

- Sequence: right-click->SIP Account Settings->Properties->Presence
- Change the presence model from peer-to-peer to presence agent, which will make the soft-phone subscribe Asterisk for SIP events.

The screenshot shows a configuration interface with tabs for Account, Voicemail, Topology, Presence, and Advanced. The 'Advanced' tab is selected. It contains three settings: 'Mode' is set to 'Presence Agent' in a dropdown menu; 'Poll time' is set to '300' seconds in a text input field; and 'Refresh interval' is set to '3600' seconds in a text input field.

Step 3: Add the contact to other soft-phones. In this example, the Xlite is account 2000, so we will add a contact for account **2001**.

Sequence: Open the right panel (presence panel in Xlite)->Click in Contacts->Add a contact.

Fill the name **2001**. Display as **2001** and don't forget to check the box **Show this contact's availability**

The screenshot shows the 'Details' configuration tab for a contact. It has two sub-tabs: 'General' and 'Details'. The 'Details' tab is active. The 'Name' field is split into 'First' (containing '2001') and 'Last' (empty). The 'Display as' field contains '2001'. Below is a 'Group(s)' dropdown menu with a 'Select More...' button. The 'Contact Methods' section has a table with columns for 'Type', 'Phone / Address', and 'Primary'. The 'Softphone' row is selected with a radio button and contains '2001' in the 'Phone / Address' field. At the bottom, the checkbox 'Show this contact's Availability' is checked.

Type	Phone / Address	Primary
Business		<input type="radio"/>
E-mail		<input type="radio"/>
Fax		<input type="radio"/>
Home		<input type="radio"/>
Mobile		<input type="radio"/>
Softphone	2001	<input checked="" type="radio"/>

Step 4: Now call extension 2001 and check the status of the phone in the right panel of the soft-phone.

Use the console commands **core show hints** to see the presence status changing in the server and **sip show inuse** to show how many calls you have on each line.



Codec configuration

Codec configuration is simple and straightforward. You can set the words **allow** and **disallow** in the **[general]** section or **peer/user** section. The best practice is to standardize the codec to avoid transcoding, which is processor intensive. Please use the same codec for messages and prompts.

```
[general]
disallow=all
allow=g729
```

DTMF options

On certain occasions, you will pass digits to an application such as voicemail or interactive voice response (IVR). It is important to pass DTMF correctly.

The simplest method for passing DTMF is called **inband**. It is set in the **[general]** or **peer/user** section of the **sip.conf** file. When you set **dtmfmode=inband**, DTMF tones are generated as sounds in the audio channel. The main issue with this method is that, when you compress the audio channel using a codec such as **g729**, sounds are distorted and DTMF tones are not properly recognized. If you are planning to use **dtmfmode=inband**, use the **g.711** codec (**ulaw** and **alaw**).

```
dtmfmode=inband
```

Another approach is to use RFC2833, which allows you to pass DTMF tones as named events in the RTP packets. A table of events corresponding to tones is provided below.

Event Codification

0—9	0—9
*	10
#	11
A—D	15
Flash	16

```
dtmfmode=rfc2833
```

Finally, you can pass DTMF digits inside SIP packets, instead of RTP packets. This method is defined in the RFC3265 (signaling events) and RFC2976.

```
dtmfmode=info
```

Following the release of version 1.2, it is now possible to use:

```
dtmfmode=auto
```

This tries to use the RFC2833; if it is not possible, use band tones.

Quality of service (QoS) marking configuration

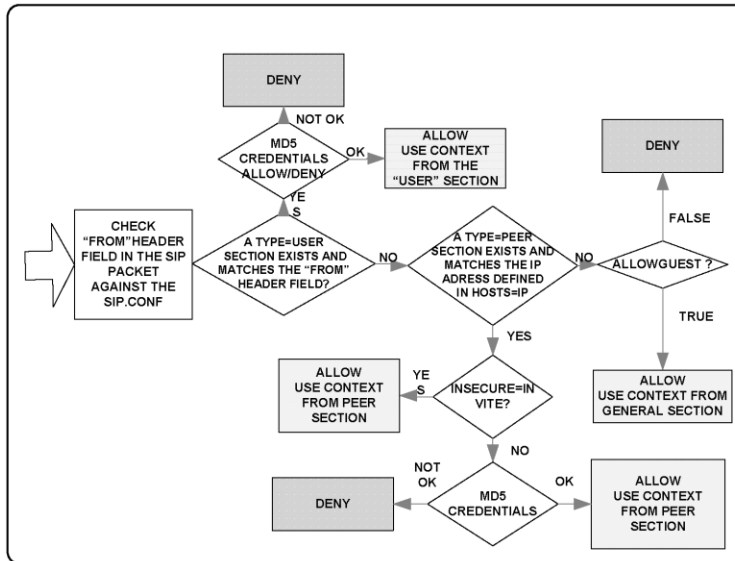
QoS is a set of techniques responsible for voice quality. QoS is implemented in such a way as to reduce bandwidth, latency, and jitter. The main QoS functions are packet scheduling, fragmentation, and header compression. **QoS is implemented in switches and routers**, not by Asterisk itself. However, Asterisk can help routers and switches by marking packets for express delivery. Marking is done using differentiated services code points (DSCP) defined in RFCs 2474 and RFC2475.

```
tos_sip=cs3  
tos_audio=ef  
tos_video=af41
```

Starting from version 1.4, you can specify different codes for signaling (SIP), audio (RTP), and video (RTP).

SIP authentication

When Asterisk receives a SIP call, it follows the rules described in the following diagram.



Three parameters play an important role in SIP authentication:

`allowguest=yes/no`

This parameter controls whether a **user** without a corresponding **peer** can authenticate without a name and secret. We discussed this parameter in the domain support section.

`insecure=invite,port`

When we use `insecure=invite`, Asterisk does not generate the message “407 Proxy Authentication Required”. Without this message, the user can make a call without authentication. This is often used to connect to VoIP service providers. The calls coming from the VoIP service provider are usually not authenticated.

`autocreatepeer=yes/no`

This command is used when Asterisk is connected to a SIP proxy. It dynamically creates a peer to each call. When this option is enabled, any UAC can connect to the Asterisk server. It is important to limit the IP connection to the SIP proxy. The SIP proxy, in turn, takes care of access control. Peer configuration is based on the general options as well as the “Contact” header field of the SIP packet.

Warning: Use this with extreme caution as it completely opens Asterisk.

`secret=senha`

This parameter configures the secret for authentication. If you do not want to present the secrets in text files, you can use `md5secret` to include a hash instead of the secret. To generate the MD5 secret, you can use:

```
echo -n "username:realm:secret" |md5sum
```

Then use the following statement:

```
md5secret=0b0e5d467890...
```

Warning: Do not forget to use the `-n` parameter; the carriage return will be used in the md5 computation.

```
deny=0.0.0.0/0.0.0.0  
permit=192.168.1.0/255.255.255.0
```

The statements above will deny all IP addresses and allow UAC only from the local network (192.168.1.0/24).

RTP options

It is possible to control some RTP parameters.

```
rtptimeout=60
```

This terminates calls without RTP activity for more the 60 seconds when not in hold.

```
rtpholdtimeout=120
```

This terminates calls without RTP activity even on hold (should be bigger than `rtptimeout`).

SIP NAT Traversal

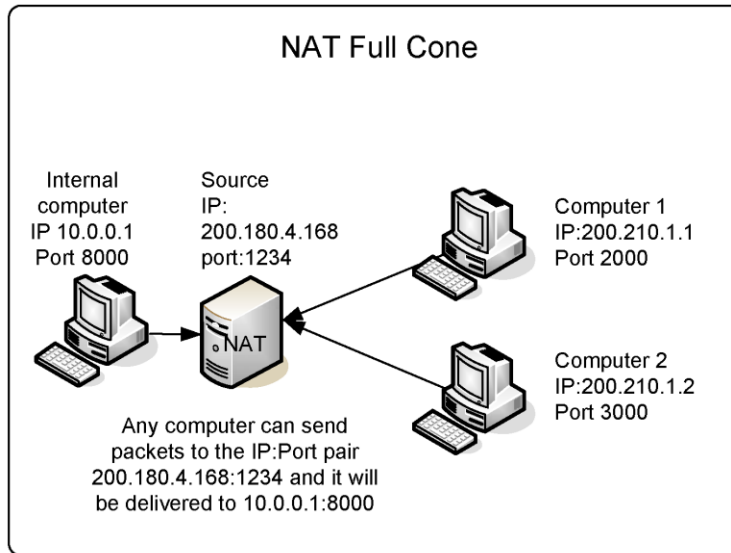
Network Address Translation (NAT) is a feature used by most networks to save Internet IP addresses. Usually, a company receives a small block of IP addresses, and end users receive one IP address dynamically when connected to the Internet. NAT solves the addressing problem by mapping internal addresses to external addresses. It stores a mapping of internal to external addresses in its memory. This mapping is valid for a specific length of time, after which the mapping is discarded. The mapping uses IP:port pairs for the internal and external addresses.

Four kinds of NAT exist:

- Full Cone
- Restricted Cone
- Port Restricted Cone
- Symmetric

Full Cone

The first NAT, full cone, represents a static mapping from an external IP:port pair to an internal IP:port pair. Any external computer can connect to it using the external IP:port pair. This is the case in non-stateful firewalls implemented with the use of filters.



Restricted Cone

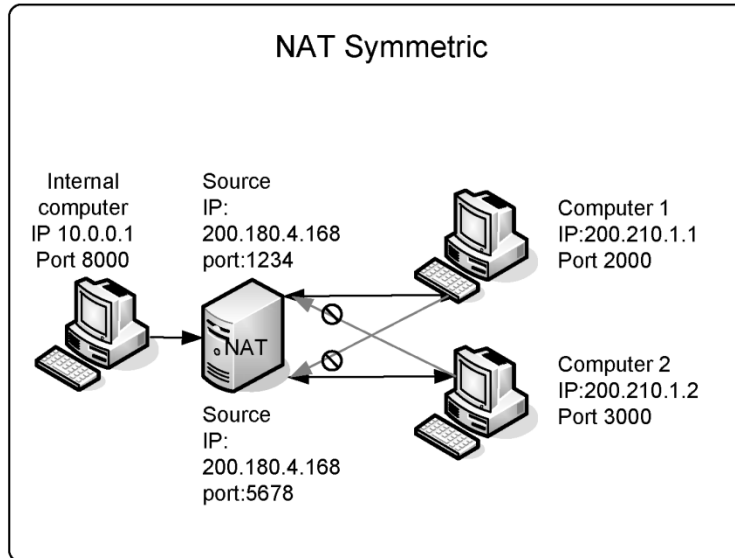
In the restricted cone scenario, the external IP:port pair is opened only when the internal computer sends data to an outside address. However, the restricted cone NAT blocks any incoming packets from a different address. In other words, the internal computer has to send data to an external computer before it can send data back.

Port Restricted Cone

The port restricted cone firewall is almost identical to the restricted cone. The only difference is that, now, the incoming packet has to come from exactly the same IP and port of the sent packet.

Symmetric

The last type of NAT is called symmetric. It is different from the first three in that a specific mapping is done to each external address. Only specific external addresses are allowed to come back by the NAT mapping. It is not possible to predict the external IP:port pair that will be used by the NAT device. The other three types of NAT allow the use of an external server to discover the external IP address for communication. With symmetric NAT, even if you can connect to an external server, the discovered address cannot be used for any other device except for this server.



NAT firewall table

The following table summarizes the three types of NAT.

	Need to send data before receiving	It is possible to determine the IP:port pair for returning packets	It restricts the incoming packets to the destination IP:port
Full Cone	No	Yes	No
Restricted Cone	Yes	Yes	Only IP
Port Restricted Cone	Yes	Yes	Yes
Symmetric	Yes	No	Yes

SIP signaling and RTP over NAT

Some of the biggest issues in NAT traversal are that you have to solve two problems: SIP signaling and audio (RTP). Most problems of one-way audio are NAT related.

An interesting thing about SIP is that, when a UAC sends a packet, it embeds the IP address in the SIP "Contact" header field. Usually this is an internal (RFC1918) address; responses to this packet cannot be routed over the Internet back to the UAC. When you put the statement "nat=yes" in the sip.conf file, you are telling Asterisk to ignore the address contained in the "Contact" header field of the SIP header and use the source IP address and port in the packet's IP header.

```
nat=yes
```

It is necessary to keep the NAT mapping open. If NAT times out, Asterisk cannot send an invite to the UAC. The UAC is able to send calls, but not receive any. The following statement can be used to keep NAT open.

```
qualify=yes
```

Qualify will send a SIP packet using the OPTIONS method regularly, which will help keep NAT open.

Even with SIP signaling resolved, we now have the challenge of passing RTP from one phone to another. If the user's NAT is of the symmetric type, it is not possible to send packets from one UAC to another directly. In this case, we have to force the RTP through Asterisk using: [something missing?]

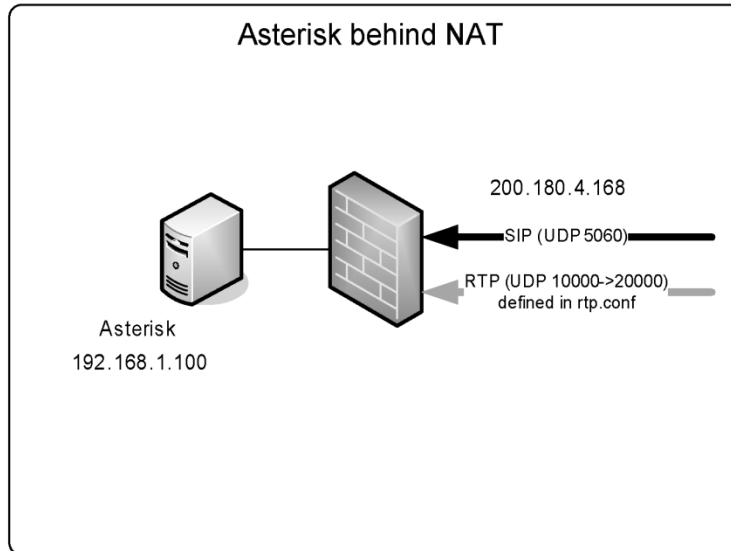
Qualify sends an OPTION each 60 seconds and every 10th second when the host is not reachable. You can use "sip show peers" to see the latency for the peers.

```
canreinvite=no
```

These configurations are appropriate for most cases. However, it is possible to optimize the traffic using advanced techniques like Simple Traversal of UDP over NAT (STUN), which is useful with full cone, restricted cone, port restricted cone, and Application Layer Gateway (ALG). Using these techniques, you do not need to do anything in Asterisk for NAT traversal. Unfortunately, most firewalls today—even home DSL/cable routers—are symmetric, making STUN unusable. ALG could solve the problem, but it is not supported, not implemented, or buggy in most cases.

Asterisk behind NAT

All the previous scenarios assume that the Asterisk server has an external (valid) Internet address. Sometimes the Asterisk server is implemented behind a firewall with NAT. In this case, it is necessary to do some extra configurations.



Step 1: Configure the firewall to redirect the UDP port 5060 statically to the Asterisk server.

Step 2: Configure the firewall to redirect the UDP ports from 10000 to 20000 statically. If you want to restrict the number of opened ports, you can edit the `rtp.conf` file to change the RTP port range. Another way is to use an intelligent firewall that supports the SIP protocol to open the RTP ports dynamically.

```
; RTP Configuration
;
[general]
;
; RTP start and RTP end configure start and end addresses
;
rtpstart=10000
rtpend=20000
```

Step 3: Configure Asterisk to include the external address in the header fields of the SIP packets including Session Description Protocol (SDP). You can accomplish this by adding the following two statements to the `sip.conf` file:

```
externip=200.180.4.168 ;External IP address
localnet=192.168.1.0/255.255.255.0 ;Internal Network Address
nat=yes
```

The first parameter `externip` tells Asterisk to include the external IP address inside the SIP headers for external destinations. The second parameter `localnet` allows Asterisk to differentiate between external and internal addresses. Optionally, you can use `externhost` if you use a Dynamic DNS with a DHCP address on the server.

SIP limitations

Asterisk uses the incoming RTP flow to synchronize the outgoing flow. If the incoming flow is interrupted (silence suppression), music-on-hold will be cut. In other words, you cannot use silence suppression in phones or providers with Asterisk.

SIP dial strings

You can call a SIP destination using different dial strings:

```
SIP/peer ; Need to have a defined peer in sip.conf
SIP/flavio@voffice.com.br ; By the URI
SIP/[exten@]peer[:portno]
SIP/[user:password@domain/extension
```

Examples include:

```
exten=>s,1,Dial(SIP/ipphone)
exten=>s,1,Dial(SIP/info@voffice.com.br)
exten=>s,1,Dial(SIP/192.168.1.8:5060,20)
exten=>s,1,Dial(SIP/8500@sip.com:9876)
```

SIP CLI commands

You can show all available SIP console commands using:

```
CLI>help sip
```

Quiz

- SIP is a protocol similar to _____ and _____ .
 - IAX
 - HTTP
 - H.323
 - SMTP
- SIP can have which types of sessions? (mark all that apply)
 - Voice
 - E-mail
 - Video
 - Chat
 - Games
- SIP components include: (mark all that apply)

- A. User Agent
 - B. Media Gateway
 - C. PSTN Server
 - D. Proxy Server
 - E. Registrar Server
4. Before a phone can receive calls, it needs to _____.
5. A SIP server can operate in the PROXY or REDIRECT mode. The difference between the two is that, in the PROXY mode, all signaling passes by the SIP proxy. In the REDIRECT mode, after discovering the location, the clients signal between themselves.
- A. True
 - B. False
6. In proxy mode, the media flow goes through the SIP proxy.
- A. True
 - B. False
7. Asterisk is a SIP proxy.
- A. True
 - B. False
8. The `canreinvite=yes/no` option is fundamental. It will define whether the medium passes within Asterisk or goes directly from one client to another. It has a major impact on Asterisk's scalability.
- A. True
 - B. False
9. Asterisk supports silence suppression in the SIP channels.
- A. True
 - B. False
10. The hardest NAT type to traverse is:
- A. Full Cone
 - B. Restricted Cone
 - C. Port Restricted Cone
 - D. Symmetric

9

Dial Plan advanced features

Chapter 3 discussed the basics of a dial plan. For didactical reasons, we didn't explain all the features, but only some of the most important ones. This chapter will delve more deeply into the dial plan, describing advanced techniques, new applications, and concepts.

Objectives

By the end of this chapter, you should be able to:

- Simplify your extension entries
- Address dial plan security and filtering extensions
- Receive calls using an IVR menu
- Use macros to avoid unnecessary rewrites
- Implement some dial plan security using “Include”
- Implement follow-me using AsteriskDB
- Implement after-hours behavior in your PBX
- Use the switch command to transfer to another PBX
- Implement the privacy manager
- Implement voicemail
- Implement a corporate directory

Simplifying your Dial Plan

This is a new feature available on Asterisk 1.6.2. You can now simplify your dial plan using the keyword “same” to define an extension. It should reduce the number of typos in the dial plan. Check the example below:

```
exten => 4000,1,NoOp()
same  =>          n,Dial(SIP/005C2B313E22)
```

Dial Plan Security

A flaw was recently discovered in the Asterisk dial plan.¹ This vulnerability allows a user to inject a new channel and dial number into your dial plan. Let's suppose that you have the following line in your server `exten=>_X.,1,dial(SIP/${EXTEN})` and some malicious user dialed the number `3000&DAHDI/1/011551123456789` in the softphone. The SIP protocol, by default, accepts any alphanumeric characters, so the extension dialed will actually trigger two calls: one for the channel SIP/3000 and the other for the channel DAHDI/011551123456789, which is an international number. Thus, any user with access to an extension can actually dial anywhere in the world.

The easiest way to avoid this behavior is to filter the numbers before calling the dial application. The function `FILTER()` is very handy for this.

Example:

```
exten=>_X.,1,DIAL(SIP/${FILTER(0-9,${EXTEN})})
```

The application filter will allow you to filter all characters from the dialed number except for the numbers 0 to 9. More information can be found in the file `README-SERIOUSLY.bestpractices.txt` available from Asterisk.

Receiving calls using an IVR menu.

In the last section, you received all calls using DID or forwarding to the operator. Now you will learn how to implement an IVR menu as well as create an auto-attendant service. Before getting into the specifics, let's examine some new applications.

Applications used in this section

- `Background()`
- `Gotoif()`
- `Record()`
- `Playback()`
- `Read()`

We put the output of the command `show application` below simply to make it easier for readers. You can obtain these descriptions using `show application applicationname`.

The `Background()` application

This application will play the given list of files while waiting for an extension to be dialed by the calling channel.

¹ <http://downloads.asterisk.org/pub/security/AST-2010-002.pdf>

Background()

Play an audio file while waiting for digits of an extension to go to.

[Description]

```
Background(filename1[&filename2...][|options[|langoverride][|context]]):
```

Options

s - skip recording if not answered
 n - Don't answer the channel before playing the files.
 m - Only break if a digit hit matches a one-digit extension

To continue waiting for digits after this application has finished playing files, the **waitExten** application should be used. The **langoverride** option explicitly specifies which language to attempt to use for the requested sound files. Any **context** that is specified will be the dial plan context that this application uses when exiting to a dialed extension. If one of the requested sound files does not exist, call processing will be terminated.

Options:

- s - Causes the playback of the message to be skipped if the channel is not in the 'up' state (i.e., it hasn't been answered yet). If this happens, the application will return immediately.
- n - Don't answer the channel before playing the files.
- m - Only break if a digit hit matches a one-digit extension in the destination context.

The Record() application

This application records from the channel into a given filename. If the file exists, it will be overwritten.

Record()

Record to a file

[Description]

```
Record(filename.format|silence[|maxduration][|options])
```

'**format**' is the format of the file type to be recorded

'**silence**' is the number of seconds of silence to allow before returning.

'**maxduration**' is the maximum recording duration in seconds.

Options

'a' : append to existing recording rather than replacing

'n' : do not answer, but record anyway if line not yet answered

'q' : quiet (do not play a beep tone)

's' : skip recording if the line is not yet answered

't' : use alternate '*' terminator key (DTMF) instead of default '#'

'x' : ignore all terminator keys (DTMF) and keep recording until hangup

- 'format' is the format of the file type to be recorded (wav, gsm, etc).
- 'silence' is the number of seconds of silence allowed before returning.
- 'maxduration' is the maximum recording duration in seconds; if it is missing or zero, there is no maximum.
- 'options' may contain any of the following letters:
 - 'a': appends to an existing recording rather than replacing it
 - 'n': do not answer, but record anyway if line is not yet answered
 - 'q': quiet (do not play a beep tone)
 - 's': skips recording if the line is not yet answered
 - 't': use alternate '*' terminator key (DTMF) instead of default '#'
 - 'x': ignore all terminator keys (DTMF) and keep recording until hang-up

If filename contains %d, these characters will be replaced with a number incremented by one each time the file is recorded. Use [core show file formats](#) to see the available formats on your system. The user can press # to terminate the recording and continue to the next priority. If the user hangs up during a recording, all data will be lost and the application will terminate.

The Playback() application

This application plays back given filenames (do not include extension). Options may also be included following a pipe symbol. The 'skip' option causes the playback of the message to be skipped if the channel is not in the 'up' state (i.e., hasn't been answered yet).

Playback()

Play a file

[Description]

Playback(filename[&filename2...][|option]):

If 'skip' is specified, the application will return immediately should the channel not be off the hook. Otherwise, unless 'noanswer' is specified, the channel will be answered before the sound is played. Not all channels support playing messages while still on the hook. If 'j' is specified, the application will jump to priority n+101 when the file does not exist, if present. This application sets the following channel variable upon completion:

- **PLAYBACKSTATUS** - The status of the playback attempt as a text string, one of:
 - SUCCESS
 - FAILED

The Read() application

This application reads a predetermined number of string digits, a certain number of times, from the user into the given variable.

Read()

Read a variable

[Description]

Read(variable[|filename][|maxdigits][|option][|attempts][|timeout])

maxdigits -- maximum acceptable number of digits.

option

- 's' to return immediately if the line is not up,
- 'i' to play filename as an indication tone from your indications.conf
- 'n' to read digits even if the line is not up

attempts -- If greater than 1, that many attempts will be made

timeout -- An integer number of seconds to wait for a digit response

- **filename** -- file to play before reading digits or tone with option **i**
- **maxdigits** -- maximum acceptable number of digits. Stops reading after **maxdigits** have been entered (without requiring the user to press the **#** key). Defaults to 0 - no limit - to

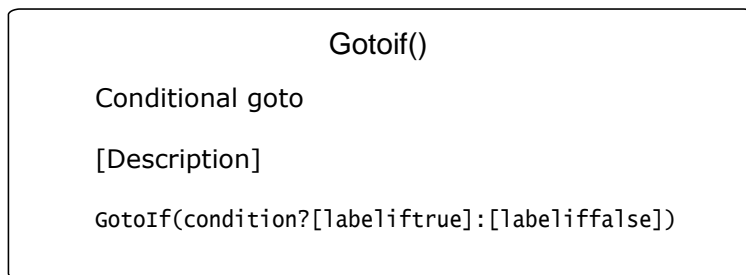
wait for the user to press the # key. Any value below 0 means the same. The maximum accepted value is 255.

- **option** -- options are **s**, **i**, **n**
 - **s** to return immediately if the line is not up
 - **i** to play filename as an indication tone from your **indications.conf**
 - **n** to read digits even if the line is not up
- **attempts** -- if greater than 1, the number of attempts that will be made in case no data are entered
- **timeout** -- An integer number of seconds to wait for a digit response. If greater than 0, that value will override the default timeout.

The read() application should disconnect if the function fails or errors out.

The Gotoif() application

This application will cause the calling channel to jump to the specified location in the dial plan based on the evaluation of the given condition. The channel will continue at **labeliftrue** if the condition is **true**, or **'labeliffalse'** if the condition is **false**. The labels are specified with the same syntax as that used within the Goto application. If the label chosen by the condition is omitted, no jump is performed; rather, the execution continues with the next priority in the dial plan.



Lab: Building an IVR menu step-by-step

Let's create an IVR menu with the following functionality:

When dialed, the IVR will play back an audio file with the message "Welcome to the XYZ Corporation; press 1 for sales, 2 for tech support, 3 for training, or wait to speak to a representative." When 1 is pressed, the call is transferred to sales (**SIP/4001**); if 2 is pressed, the call is transferred to tech support (**SIP/4002**); if 3 is pressed, the call is transferred to training (**SIP/4003**). If no digits are pressed, the call is transferred to the operator (**SIP/4000**).

Step 1 – Record the prompts

Let's create an extension to record the prompts. To record a prompt, dial from a soft phone to **9003filename**. When you hear the beep, start recording; press # to stop recording. You will hear a beep and the system will play back the recorded prompt.

Recording a prompt

```

exten=_9003.,1,answer()
exten=_9003.,n,Record(${EXTEN:4}.wav||5|t)
exten=_9003.,n,Playback(${EXTEN:4})
exten=_9003.,n,Hangup()

```

Step 2 – Creating the menu logic

When dialing the 9004 extension, processing will jump to the menu in the ‘s’ extension, priority 1.

Creating the menu logic

```

exten=>9004,1,goto(menu,s,1)
[menu]
exten=>s,1,Background(mainmenu)
exten=>s,n,waitExten(30)
exten=>1,1,goto(sales,s,1)
exten=>2,1,goto(techsupport,s,1)
exten=>3,1,goto(training,s,1)
;handling an invalid digit
exten=>i,1,Dial(${OPERATOR})
;handling timeout
exten=>t,1,Dial(${OPERATOR})
[sales]
exten=>s,1,Dial(SIP/4001,20,t)
[techsupport]
exten=s,1,Dial(SIP/4002,20,t)
[training]
exten=s,1,Dial(SIP/4002,20,t)

```

Matching as you dial

This is a company setup menu for receiving calls. The background application reads the current context and defines the maximum length for each number for any possible combination.

```

[incoming]
exten=>s,1,Background(welcome)
exten=>1,1,Dial(DAHD1/1)
exten=>2,1,Dial(DAHD1/2)
exten=>21,1,Dial(DAHD1/3)
exten=>22,1,Dial(DAHD1/4)
exten=>31,1,Dial(DAHD1/5)
exten=>32,1,Dial(DAHD1/6)

```

When you dial this company, the welcome message is played first. After that, Asterisk waits for a digit to be dialed.

Dialed number	Asterisk Action
1	Immediately calls Dial(DAHDI/1)
2	Waits for the timeout, then goes to Dial(DAHDI/2)
21	Immediately calls (DAHDI/3)
22	Immediately calls (DAHDI/4)
3	Waits for the timeout, then disconnects
31	Immediately calls Dial(DAHDI/5)
32	Immediately calls Dial(DAHDI/6)
4	Immediately disconnects

It is important to avoid ambiguity in the menus. Everybody wants to be answered quickly. For this reason, you should not use numbers 2, 21, or 22.

Lab: Using the Read() application

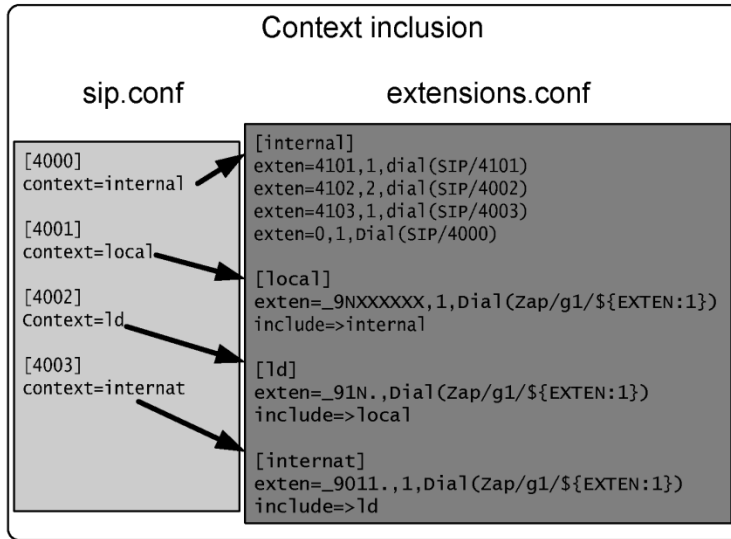
Please try the lab with the read() application. **Read** accept digits from the user and inserts them into the specified variable; you can then use the **gotoif** application to redirect the call.

Using the Read() application

```
exten=9005,1,Read(test||1)
exten=9005,n,Gotoif($[${test}=1]?one:other)
exten=9005,n,hangup()
exten=9005,n(one),playback(tt-weasels)
exten=9005,n,hangup()
exten=9005,n(other),playback(tt-monkeys)
```

Context inclusion

A context can include the contents of another context.



In the example above, any channel can dial any extension in the internal context, but only the 4003 channel can dial international extensions. You can use context inclusion to make dial plan creation easier. Using context inclusion, you can control who has access to what extensions.

Troubleshooting the message “number not found”

It is very common to receive the message “number not found”. Most people confuse the concept of included contexts because it is really not intuitive. As a rule of thumb, first go to the incoming channel configuration file, such as `sip.conf`, `chan_dahdi.conf`, and `iax.conf`, and determine the **current context**. Then, go to the dial plan in the file `extensions.conf` and check if the number dialed can be found in that context. If not, something is wrong with your dial plan. The golden rules of contexts are:

1. A channel can only dial numbers within the same context as the channel.
2. The context where the call is processed is defined in the incoming channel (`chan_dahdi.conf`, `iax.conf`, `sip.conf`).

Using the switch statement

You can send the dial plan processing to another server using the `switch` command. You will need the name and key of the other server. The context is the destination context.

Switch

The SWITCH statement permits a server to share the dial plan with another server.

```
;Switch to another server using IAX2  
[iaxprovider]  
;switch => IAX2/user:[key]@myserver/mycontext  
  
;Switch to the same server using local channel  
[subroutine]  
switch => Loopback/1234@othercontext
```

Dial plan processing order

When Asterisk receives an incoming call, it looks in the context defined by the channel. In some cases, if more than one pattern matches the dialed number, Asterisk cannot process the call in the exact way you think it should. You can see the matching order using the `dialplan show` CLI command.

Dial plan processing order

1. An exact match using dialed number and callerID
2. An exact match using only dialed number
3. A pattern that matches the dialed number
4. A context included in the switch statement
5. An included context

Example:

Let's say that you want to dial 912 to route to an analog trunk (DAHDI/1) and all other numbers starting with 9 to another analog trunk (DAHDI/2). You would write something like:

```
[example]  
exten=>_912.,1,Dial(DAHDI/1/${EXTEN})  
exten=>_9.,1,Dial(DAHDI/2/${EXTEN})
```

If two patterns match an extension, you can control what extension is processed first using the included contexts. An included context is processed later than a pattern in the same context.

The #INCLUDE statement

Should we use a big file or several files? You can use the `#include <filename>` statement to include other files in your `extensions.conf`. For example, we could create a `users.conf` for local users and

`services.conf` for special services. Be careful to not confuse `#include <filename>` with the `include=>context` statement.

Macros

A macro is an instruction set that executes sequentially. Macros are used to avoid the rewriting lines of code, thereby allowing the reutilization of pieces of code within the dial plan.

Macros

Defining

```
[macro-stdexten] ;Macro standard extension
;${ARG1}-channel to ring
exten=>s,1,Dial(${ARG1}
exten=>s,n,goto(${DIALSTATUS})
exten=>s,n,hangup()
exten=>s,n(BUSY),voicemail(b${MACRO_EXTEN}
exten=>s,n,hangup
exten=>s,n(NOANSWER),voicemail(u${MACRO_EXTEN}
exten=>s,n,hangup
exten=>s,n(CHANUNAVAILABLE),hangup
exten=>s,n(CANCEL),hangup
exten=>s,n(CONGESTION),hangup
```

Calling

```
exten =>_4XXX,2,Macro(stdexten,sip/${EXTEN})
```

Defining a macro

You can define a macro using `[macro-macroname]`. Within the macro, you may use the specific variables listed below:

- `${ARG1}`: First macro argument
- `${ARG2}`: Second macro argument (etc.)
- `${MACRO_CONTEXT}`: Context where macro was called
- `${MACRO_EXTEN}`: Extension that triggers the macro
- `${MACRO_OFFSET}`: Set by a macro to influence priority after exiting
- `${MACRO_PRIORITY}`: Priority where the macro was triggered

Calling a macro

To call a macro you need to use the application:

```
Macro(macroname, arg1, arg2...)
```

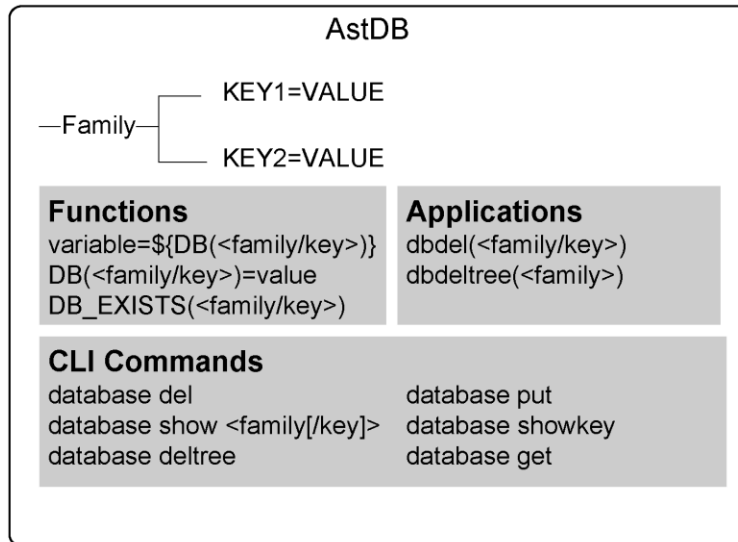
To call the macro defined above, you should use:

```
exten =>_4XXX,2,Macro(stdexten,sip/${EXTEN})
```

Please check the macros defined in `extensions.conf` (sample file) for additional examples.

Using Asterisk DB

To implement call forward and black lists, we need some way to store and restore data. Fortunately, Asterisk provides a mechanism for storing and retrieving data from a Berkley DB database (version 1) called AstDB. This is similar to the Windows registry database using the family and keys hierarchical concept. The data persist between Asterisk restarts.



Functions, applications, and CLI commands

There are some functions, applications, and CLI commands that work with AstDB:

- `variable=${DB(<family/key>)}`
- `DB(<family/key>)=value`
- `DB_EXISTS(<family/key>)`

Examples:

```
exten=_*21*XXXX,1,set(DB(CFBS/${CALLERID(num)})=${EXTEN:4})
exten=s,1,set(temp=${DB(CFBS/${EXTEN})})
```

Some applications can be used to manipulate AstDB:

- `dbdel(<family/key>)`
- `dbdeltree(<family>)`

It is possible to use CLI commands to set and delete keys as well:

- database del
- database put
- database show <family[/key]>
- database showkey
- database deltree
- database get

Implementing Call Forward, DND, and Blacklists

In this example, you will learn how to implement call forward immediate and call forward on busy. We will use ***21*** to program call forward immediate and ***61*** to program call forward on busy status. To cancel the programming, use **#21#** and **#61#**, respectively. Use the above example to populate the database.

Call Forward. BlackList, DND

```
[apps]
;call forward immediate
exten=>_ *21*XXXX,1,Set(DB(CFIM/${CALLERID(num)})=${EXTEN:4})
exten=>_ *21*XXXX,2,Hangup
exten=>*21*,1,DBdel(CFIM/${CALLERID(num)})
exten=>*21*,2,Hangup
;Do not disturb
exten=>_ *41*X.,1,Set(DB(dnd/${EXTEN:4})=${EXTEN:4})
exten=>_ *41*X.,n,Hangup
exten=>*41*,1,DBdel(dnd/${EXTEN:4})
exten=>*41*,2,Hangup
;call forward on busy status
exten=>_ *61*XXXX,1,Set(DB(CFBS/${CALLERID(num)})=${EXTEN:4})
exten=>_ *61*XXXX,2,Hangup
exten=>*61*,1,DBdel(CFBS/${CALLERID(num)})
exten=>*61*,2,Hangup
```

Families used:

- CFIM – Call Forward Immediate
- CFBS – Call Forward on Busy status
- DND – Do Not Disturb

Try populating the database dialing:

- ***21*** (Destination extension for call forwarding immediate)
- ***61*** (Destination extension for call forwarding on busy status)
- ***41*** (Extension to put on do not disturb)

Use the CLI command **database show** to see the families, keys, and values added.

Call Forward. BlackList, DND

```
[macro-stdexten]
;${ARG1}-Extension
exten=>s,1,gotoif(${DB_EXISTS(dnd/${ARG1})}?dnd)
exten=>s,n,gotoif(${DB_EXISTS(CFIM/${ARG1})}?cfim)
exten=>s,n(dial),Dial(SIP/${ARG1},20)
exten=>s,n,goto(${DIALSTATUS})
exten=>s,n,hangup()
exten=>s,n(BUSY),gotoif(${DB_EXISTS(CFBS/${ARG1})}?cfbs:end)
exten=>s,n(cfbs),Dial(SIP/${DB(CFBS/${ARG1})},20)
exten=>s,n,hangup()
exten=>s,n(cfim),Dial(SIP/${DB(CFIM/${ARG1})},20)
exten=>s,n,hangup()
exten=>s,n(dnd),Playback(donotdisturb)
exten=>s,n,hangup()
```

```
exten=_4XXX,1,Macro(stdexten,${EXTEN})
```

The above macro verifies if the database contains the key:value pairs corresponding to CFIM, CFBS, or DND, and then handles them appropriately. The follow macro calls the dialing routine:

```
exten=_4XXX,1,Macro(stdexten,${EXTEN})
```

Using a blacklist

To create a blacklist, we use the `LookupBlacklist()` application. The application checks the name/number in the caller ID. If the number is not found, the application sets the variable `$LOOKUPBLSTATUS` to `NOTFOUND`. If the number is found, the application sets the variable to `FOUND`. You can use the “j” option in the application to use the old (1.0) behavior, jumping 101 positions if the number/name is found.

Example:

```
[incoming]
exten => s,1,LookupBlacklist(j)
exten => s,2,Dial(SIP/4000,20,tTj)
exten => s,3,Hangup()
exten => s,102,Goto(blocked,s,1)
```

```
[blocked]
exten => s,1,Answer()
exten => s,2,Playback(blockedcall)
exten => s,3,Hangup()
```

To insert a number in the blacklist, we can use the same resource as before, using `*31*` followed by the extensions to be blacklisted. To remove a number from the blacklist, you should use `#31#` followed by the number to be removed.

```
[apps]
exten=>_*31*X.,1,Set(DB(blacklist/${EXTEN}=1})
exten=>_*31*X.,2,Hangup()
exten=>_#31#X.,1,dbdel(blacklist/${EXTEN}:4)
exten=>_#31#X.,2,Hangup()
```

You can also insert the numbers in the blacklist using the console CLI:

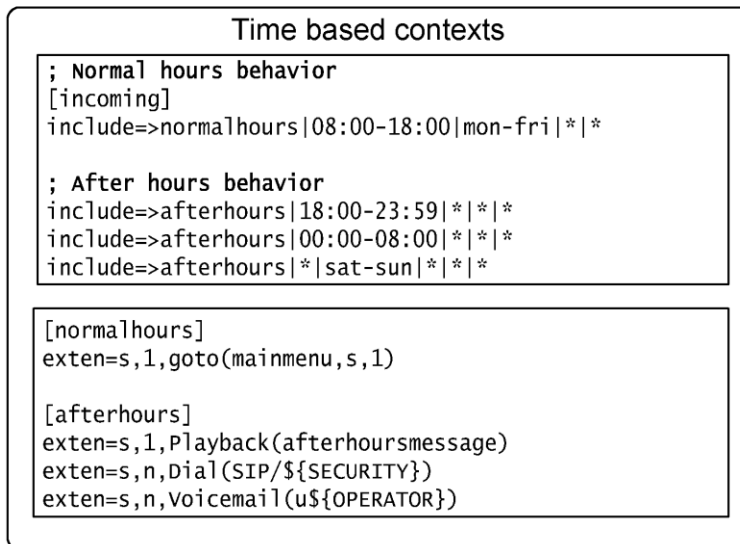
```
CLI>database put blacklist <name/number> 1
```

Note: Any value can be associated with the key. The blacklist application will search for the key, not the value. To erase the number from the blacklist, you can use:

```
CLI>database del blacklist <name/number>
```

Time-based contexts

In the following figure, we have a dial plan with three contexts. The [incoming] context is where the calls are usually received. We have included four lines that change behavior depending on the system time, as exemplified below:



```
include => context|<times>|<weekdays>|<mdays>|<months>
```

During regular working hours, processing will be redirected to the **mainmenu**, where it will probably call an IVR to handle the incoming call. If the call takes place after hours, it will call the security extension defined in the **\${SECURITY}** variable. If the security extension does not answer the call, it will be sent to the operator's voicemail.

Time-based messages using gotoiftime()

The `gotoiftime()` syntax is shown below.

```
GotoIfTime(<timerange>|<daysofweek>|<daysofmonth>|<months>?[[context|]extension  
|pri)
```

This application can replace the time-based context and seems easier to understand and read. You can specify the time as follows:

- `<timerange>=<hour>:'<minute>'-'<hour>:'<minute>|"*"`
- `<daysofweek>=<dayname>|<dayname>'-'<dayname>|"*"`
- `<dayname>="sun"|"mon"|"tue"|"wed"|"thu"|"fri"|"sat"`
- `<daysofmonth>=<daynum>|<daynum>'-'<daynum>|"*"`
- `<daynum>=number from 1 to 31`
- `<hour>=number from 0 to 23`
- `<minute>=number from 0 to 59`
- `<months>=<monthname>|<monthname>'-'<monthname>|"*"`
- `<monthname>="jan"|"feb"|"mar"|"apr"|"may"|"jun"|"jul"|"aug"|"sep"|"oct"|"nov"|"dec"`

Names for days and months are not case sensitive.

```
exten=>s,1,GotoIfTime(8:00-18:00|mon-fri|*|*?normalhours,s,1)
```

The previous statement transfers the processing to the extension `s` in the `normalhours` context if the call is between 08:00AM and 06:00PM from Monday to Friday.

Using DISA to get a new dial tone

DISA, or “direct inward system access,” is a system that allows users to receive a second dial tone. It permits users to dial again to another destination. It is often used by technicians when dialing long distance calls for technical support on weekends; instead of dialing from their homes directly to the destination, they call the office’s DISA number, receive a dial tone, and then call the destination. Long-distance charges incur at the company instead of the home phone.

```
DISA(passcode[|context])  
DISA(password file)
```

Example:

```
exten => s,1,DISA(no-password|default)
```

Using the previous statement, the user dials the PBX and—without requiring any password—receives a dial tone. Any call using DISA will be processed using the default context.

The arguments for this application include a global password or individual password within a file. If no context is specified, `DISA` will be assumed. If you use a password file, the complete path has to be specified. A caller ID can be specified for the DISA external dialing too.

Example:

```
numeric-passcode|context|"Flavio" <4830258590>
```

Limit simultaneous calls

Since version 1.2, new functions have been added. The `GROUP()` function allows you to count how many active channels you have in one group at the same time.

Example:

You have a branch in Rio de Janeiro, where phones follow the pattern “_214X”. This location is served by a leased line, with 64K reserved for voice bandwidth. In this case, the maximum number of allowed calls is 2 (G.729, 30r.2K per call). To limit calls to Rio by two:

```
exten=>_214X,1,set(GROUP())=Rio)
exten=>_214X,n,Gotoif($[${GROUP_COUNT()} > 1]?outoflimit)
exten=>_214X,n,Dial(SIP/${EXTEN})
exten=>_214X,n,hangup
exten=>_214X,n(outoflimit),playback(callsexceedcapacity)
exten=>_214X,n,hangup
```

Voicemail

Voicemail is a computerized telephone answering system that records incoming voice messages, saving them on disk or sending them via e-mail. Sometimes it has a directory where you can look up voicemail boxes by name. In the past, voicemail systems were very expensive. Now, with IP telephony, voicemail is becoming a standard feature. To configure voicemail, you should go through the following steps:

Step 1: Edit the file `voicemail.conf` and set general parameters.

format	Codec used to read the message (e.g., wav49, wav, gsm)
serveremail	Who the email notification should appear to come from
maxmsg	Maximum number of messages in the mailbox; after this threshold, messages are discarded
maxsecs	Maximum number of seconds for the voicemail message
minsecs	Minimum number of seconds for the message; below this threshold, no message is recorded
maxsilence	What to consider silence in seconds

Step 2: Edit the file `voicemail.conf` and create the users' mailboxes

Voicemail.conf

Mailbox ID=pincode, fullname,email address,pager e-mail,option=value, option=value

- MailboxID => usually the extension number
- Pincode=>password to access e-mail system
- Full name => used for the directory application
- E-mail => for voicemail notification
- Pager e-mail=>For notification via SMS gateway or pager
- Options=> The same options but applied to the MailboxID

Voicemail has several options that control voicemail behavior. For now, we will stick to the default options and concentrate on mailbox definition. After the `[general]` section in the file, you will start configuring the mailbox ID in its own context.

Example:

```
[general]
```

```
[default]
```

```
1234=>1234,SomeUser,email@address.com,pager@address.com,saycid=yes|dialout=from  
vm|callback=fromvm|review=yes|operator=yes
```

Please check for advanced options in the file `voicemail.conf`.

Step 3: Configuring the file `extensions.conf`

Below, you have the instructions to create the macro and the macro-call to implement voicemail in the file `extensions.conf`. We use the results of the channel variable `$(DIALSTATUS)` to redirect the call flow to the proper voicemail menu.

Voicemail Macro

```

;we will, usually use a macro to process the voicemail
[macro-stdexten]
exten=>s,1,Dial(${ARG1},20,t)
exten=>s,n,Goto(${DIALSTATUS})
exten=>s,n,hangup()
exten=>s,n(BUSY),voicemail(${MACRO_EXTEN},b)
exten=>s,n,hangup()
exten=>s,n(NOANSWER),voicemail(${MACRO_EXTEN},u)
exten=>s,n,hangup()
exten=>s,n(CANCEL),hangup
exten=>s,n(CHANUNAVAIL),hangup
exten=>s,n(CONGESTION),hangup

[local]
exten=>6601,1,Macro(stdexten,SIP/6601)
exten=>6602,1,Macro(stdexten,SIP/6602)

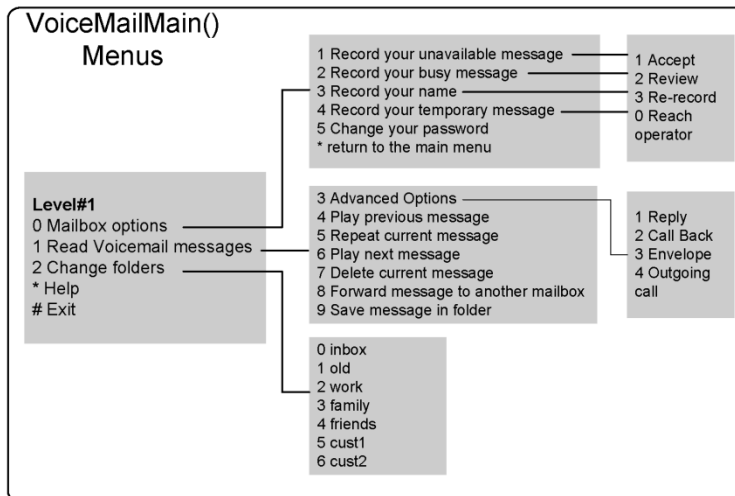
```

Using the Voicemailmain() application

The application `voicemailmain()` is used to configure the voicemail mailbox. Users can dial the application, record their greeting, and listen to their voicemail. To call the application in the dial plan, use:

```
exten=>9000,1,voicemailmain()
```

Below you will find a list of the options available for the application.



Voicemail application syntax

This application allows the calling party to leave a message for a specified list of mailboxes. When multiple mailboxes are specified, the greeting will be taken from the first specified mailbox. The dial plan execution will stop if the specified mailbox does not exist. The syntax is shown below:

[Synopsis]

Leave a voicemail message.

[Description]

This application allows the calling party to leave a message for the specified list of mailboxes. When multiple mailboxes are specified, the greeting will be taken from the first mailbox specified. Dialplan execution will stop if the specified mailbox does not exist.

The voicemail application will exit if any of the following DTMF digits are received:

0 - Jump to the 'o' extension in the current dialplan context.

* - Jump to the 'a' extension in the current dialplan context.

This application will set the following channel variable upon completion:
\${VMSTATUS}: This indicates the status of the execution of the voiceMail application.

SUCCESS

USEREXIT

FAILED

[Syntax]

VoiceMail(mailbox[@context] [&mailbox[@context] [&...]] [,options])

[Arguments]

options

b: Play the 'busy' greeting to the calling party.

d([c]): Accept digits for a new extension in context <c>, if played during the greeting. Context defaults to the current context.

g(#): Use the specified amount of gain when recording the voicemail message. The units are whole-number decibels (dB). Only works on supported technologies, which is DAHDI only.

s: Skip the playback of instructions for leaving a message to the calling party.

u: Play the 'unavailable' greeting.

U: Mark message as 'URGENT'.

P: Mark message as 'PRIORITY'.

In all cases, the beep.gsm file will be played before the recording begins. Voicemail messages will be stored in the inbox directory.

/var/spool/asterisk/voicemail/context/boxnumber/INBOX/

If a caller presses 0 (zero) during the announcement, it will be moved to the 'o' (out) extension in the voicemail current context. This can be used to exit to the operator.

If during the recording the caller presses # or the silence limit times out, recording is stopped and the call goes to the next priority. Make sure that you handle the call after the voicemail is played, as shown below.

```
exten=>somewhere,5,Playback(Goodbye)
exten=>somewhere,6,Hangup
```

New feature in version 1.6.x

You may now tag some messages as “urgent.” Two methods are available for this:

- Pass the option ‘**U**’ in the application voicemail()
- Specify **review=yes** in the file **voicemail.conf**. If using this option, the user will be able to tag the message as urgent after recording the voice instructions.

Sending voicemail to e-mail

In some cases (like mine), we simply do not use the **voicemailmain()** application to read e-mail. It is simpler and more practical to send all messages to e-mail with the audio attached. Using the parameters ‘attach’ and ‘delete’, you can send all mails to e-mail and delete them from the mailbox.

```
attach=yes
delete=yes
```

To send voicemail to e-mail, the voicemail application uses the message transfer agent (MTA), a component of your operating system. Debian uses Exim as the MTA. The application that sends the e-mail is defined in the ‘mailcmd’ parameter.

```
mailcmd =/usr/sbin/sendmail -t
```

In the Debian distribution of Linux, the MTA is Exim. To configure Exim in Debian, use:

```
dpkg-reconfigure exim4-config
```

You can choose to make your MTA send an e-mail directly through SMTP or a smarthost (usually your company’s mail server). Verify with your e-mail administrator the best way to send e-mail from the Asterisk server to your e-mail server.

Customizing the e-mail message

You can control how messages are sent by setting up the following variables:

Variables for e-mail subject and e-mail body:

- VM_NAME
- VM_DUR
- VM_MSGNUM
- VM_MAILBOX
- VM_CIDNUM

You might need to edit this script before installing. Copy the source files to the html directory.

```
cp /usr/asterisk/images/*.gif /var/html/asterisk
```

Use the following command to make the cgi executable.

```
chmod +x vmail.cgi
```

Voicemail notification

You can configure voicemail to send a notify message to your phone when you have new voicemail. Voicemail notification works with SIP phones, ZAP phones, and some IAX2 phones. To indicate an unheard voicemail, an indicator light may blink or the phone may play a shutter tone. You need to configure the mailbox in the corresponding channel configuration file.

Example: `sip.conf`

```
mailbox=8590
```

Lab: Message Notification in the Phone

This lab was tested using Xlite for windows 3.0.

1. Edit the `sip.conf` and add 'mailbox=4401' in the SIP channel named 4401.
2. Edit the `extensions.conf` and create an extension to record a voicemail to 4401 extensions.

```
exten=9008,n,voicemail(b4401)
```
3. Go to the CLI > console and reload.
4. Go to the X-Lite > Mouse Right Button > SIP Account Settings > Properties > Voicemail and check the box 'check voicemail'.
5. Dial 9008 and leave a message.
6. Observe the message icon on the phone.

Using the directory application

This application allows you to quickly find a user to dial. The list of names and corresponding extensions are retrieved from the voicemail configuration file `voicemail.conf`. Syntax for the application can be shown using `core show application directory`:

```
-- Info about application 'Directory' --
```

```
[Synopsis]
Provide directory of voicemail extensions.
```

```
[Description]
This application will present the calling channel with a directory of
extensions from which they can search by name. The list of names and
corresponding extensions is retrieved from the voicemail configuration file,
```

"voicemail.conf".

This application will immediately exit if one of the following DTMF digits are received and the extension to jump to exists:

'0' - Jump to the 'o' extension, if it exists.

'*' - Jump to the 'a' extension, if it exists.

[Syntax]

Directory([vm-context][,dial-context[,options]])

[Arguments]

vm-context

This is the context within voicemail.conf to use for the Directory.

If not specified and 'searchcontexts=no' in "voicemail.conf", then 'default' will be assumed.

dial-context

This is the dialplan context to use when looking for an extension

that the user has selected, or when jumping to the 'o' or 'a' extension.

options

e: In addition to the name, also read the extension number to the caller before presenting dialing options.

f(n): Allow the caller to enter the first name of a user in the directory instead of using the last name. If specified, the optional number argument will be used for the number of characters the user should enter.

l(n): Allow the caller to enter the last name of a user in the directory. This is the default. If specified, the optional number argument will be used for the number of characters the user should enter.

b(n): Allow the caller to enter either the first or the last name of a user in the directory. If specified, the optional number argument will be used for the number of characters the user should enter.

m: Instead of reading each name sequentially and asking for confirmation, create a menu of up to 8 names.

p(n): Pause for n milliseconds after the digits are typed. This is helpful for people with cellphones, who are not holding the receiver to their ear while entering DTMF.

NOTE: Only one of the <f>, <l>, or options may be specified.

If more than one is specified, then Directory will act as if was specified. The number of characters for the user to type defaults to '3'.

Lab: Using the directory application

1. Edit the **voicemail.conf** file to add two extensions in the dial plan

```
[default]
```

```
; Define maximum number of messages per folder for a particular context.
```

```
;maxmsg=50
```

```
4400=>4400,Clint Eastwood,ceastwood@asteriskguide.com
```

```
4401=>4401,John Wayne,jwayne@asteriskguide.com
```

2. Create these extensions in your dial plan

```

exten=9006,1,voiceMailMain()
exten=9006,n,Hangup()
exten=9007,1,Directory(default|default)
exten=9007,n,Hangup()

```

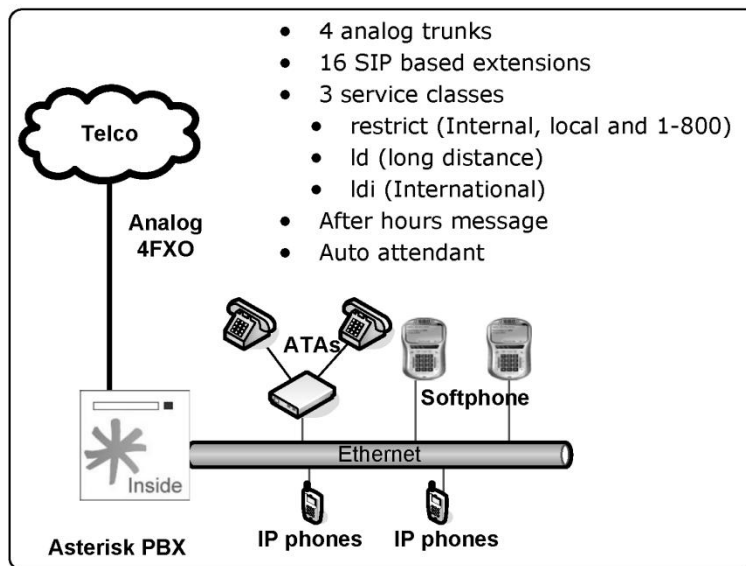
3. Go to the console and reload

4. Dial 9006 and record a name for each extension (4400, 4401)

5. Dial 9007 and select the three letters of the last name for one extension (Eas=327). If this is the correct option, press '1' to transfer to the name.

Lab: Putting it all together

Thus far, you have learned several dial plan concepts. Let's put all the applications, functions, and concepts in a dial plan example so you can understand how they are used together.



Let's guide you through the whole PBX configuration for the scenario below.

- 4 analog trunks
- 16 SIP-based extensions
- 3 service classes
 - restrict (Internal, local and 1-800)
 - ld (long distance)
 - ldi (international)

- After-hours message
- Auto attendant

Step 1 – Configuring channels

Analog trunks (chan_dahdi.conf)

First, we will configure the analog trunks in the DAHDI channel configuration file `chan_dahdi.conf`. In this case, we will use a T400P Digium card with 4 FXO interfaces. Let's assume that the driver is already loaded and the driver configuration file (`/etc/dahdi/system.conf`) is correctly configured.

```
signalling=fxs_ks
language=en
context=incoming
group=1
channel => 1-4
```

SIP channels (sip.conf)

We have chosen the dial plan numbering from 2000 to 2099. Two codecs will be used: G.729 and G.711 ulaw. The first one will be used for phones using Asterisk over the Internet or WAN while the second one will be used for phones using the local network. In the sip.conf, we will arbitrate which devices will belong to each class of service (`restrict`, `ld`, `ldi`). To reduce the vulnerability to brute force attacks, we will use the phone's MAC addresses as device names.

I strongly advise that you use strong passwords to avoid brute force attacks!

```
[general]
disallow=all
allow=gsm
allow=ulaw
bindport = 5060
bindaddr = 0.0.0.0
context = restrict
```

```
[00001A000002]
type=friend
username=20
secret=#s2cr2t#
host=dynamic
mailbox=20
context=restrict
canreinvite=yes
```

```
[00001A000003]
type=friend
username=20
```

```
secret=#s3cr3t#
host=dynamic
mailbox=20
context=ld
canreinvite=yes
dtmfmode=rfc2833
```

```
[00001A000004]
type=friend
username=20
secret=#s3cr3t#
host=dynamic
mailbox=20
context=ldi
canreinvite=yes
dtmfmode=rfc2833
```

Step 2 – Configure the dial plan

Now let's start to configure the `extensions.conf`.

Define internal extensions and local dialing

```
[restrict]
exten=>_2000,1,Dial(SIP/00001A000002,20,t)
exten=>_2030,1,Dial(SIP/00001A000003,20,t)
exten=>_2040,1,Dial(SIP/00001A000004,20,t)
exten=>_9XXXXXXX,1,Dial(DAHD1/g1/${EXTEN:1},20) ; local calls
exten=>_91800.,1,Dial(DAHD1/g1/${EXTEN:1},20); 1-800
```

Define LD (long distance)

```
[ld]
include=>restrict
exten=>_9NXXNXXXXXX,1,Dial(DAHD1/g1/${EXTEN:1},20)
```

Define international calls

```
[ldi]
include=> ld
exten=>_901X.,1,Dial(DAHD1/g1/${EXTEN:1},20)
```

Step 3 - Receiving calls using an auto-attendant

To receive calls, use two contexts. The first one is for normal-hours operation, where the call will be received by an auto-attendant. The second one is for after hours, where the caller will receive a message such as “you have called company XYZ, our normal hours are from 08:00 AM to 06:00 PM; if you know the destination extension number you can try dialing it now or hang up.”

Menus: Normal-hours, After-hours

In the menus below, the system will play a message warning the caller that the company was reached after regular working hours, allowing the caller to dial the destination extension number (someone may be working after regular working hours).

```
[incoming]
include=>normalhours|08:00-18:00|mon-fri|*|*
```

```
include=>afterhours|18:00-23:59|*|*|*
include=>afterhours|00:00-07:59|*|*|*
include=>afterhours|*|sat-sun|*|*
```

```
[normalhours]
exten=>s,1,Goto(mainmenu,s,1)
```

```
[afterhours]
exten=>s,1,Background(afterhours)
exten=>s,2,hangup()
exten=>i,1,hangup()
exten=>t,1,hangup()
include=>restrict
```

Menus: Main and Sales

During normal working hours, the call is answered by an auto-attendant menu, receiving a message such as “welcome to XYZ Company; dial 1 for sales, 2 for tech support, 3 for training, or the desired extension number”.

```
[globals]
OPERATOR=SIP/2060
SALES=SIP/2035
TECHSUPPORT=SIP/2004
TRAINING=SIP/2036
```

```
[mainmenu]
exten=> s,1,Background(welcome)
exten=>1,1,Goto(sales,s,1)
exten=>2,1,Goto(techsupport,s,1)
exten=>3,1,Goto(training,s,1)
exten=>i,1,Playback(Invalid)
exten=>i,2,hangup()
exten=>t,1,Dial(${OPERATOR},20,Tt)
include=>restrict
```

```
[sales]
exten=>s,1,Dial(${SALES},20,Tt)
```

```
[techsupport]
exten=>s,1,Dial(${TECHSUPPORT},20,Tt)
```

```
[training]
```

```
exten=>s,1,Dial(${TRAINING},20,Tt)
```

With all these statements, the functionality of your dialing plan is now ready. In the next section, we will demonstrate how to operate the PBX.

Summary

In this chapter, you have learned how to receive calls using an IVR or an auto-attendant. You have studied the concept of context inclusion and implemented a few examples. Macros were used to avoid repetitive typing, and the Asterisk database based on the Berkley DB engine was used for functions that require data storage (e.g., call forward, do not disturb, blacklists). Finally, you have learned how to implement after-hours behavior and implemented a complete dial plan using these concepts.

Quiz

1. To include a time-dependent context, you can use:

```
include=> context|<times>|<weekdays>|<mdays>|<months>
```

What does the following statement do?

```
include=>normalhours|08:00-18:00|mon-fri|*|*
```

- A. Execute extensions from Monday to Friday from 08:00 to 18:00
- B. Execute options every day in all months
- C. Nothing; its format is invalid.

2. When a user dials “0” to get an external line, Asterisk automatically cuts the audio. This can be bad because the user is used to hearing an external dialing tone before dialing the other numbers. You can simulate the old dialing behavior with the _____ statement.

3. The following statements make the user who called the 8590 extension (mark all that apply):

```
exten => 8590/482518888,1,Congestion
exten => 8590,2,Dial(DAHDI/1,20,j)
exten => 8590,3,voicemail(u8590)
exten => 8590,103,voicemail(b8590)
```

- A. Receive a busy tone if the CallerID=482518888
- B. Receive a busy tone, independent from the number dialed.
- C. Dial the DAHDI/1 channel
- D. Go to to the voicemail if DAHDI/1 is busy or is not answered, except when CallerID=482518888.

4. To concatenate several extensions, you can separate them using the ____ character.

5. A voice menu is usually created using the _____ application.

6. You can include files inside the configuration files using the _____ statement.

7. The Asterisk database is based in _____.
- A. Oracle
 - B. MySQL
 - C. Berkley DB
 - D. PostgreSQL
8. When you use `Dial(type1/identifier1&type2/identifier2)`, Asterisk dials to each one in sequence, waiting 20 seconds between them.
- A. False
 - B. True
9. Using the Background application, you need to wait until the message is played before you can choose an option to send a DTMF digit.
- A. False
 - B. True
10. The valid formats for the `goto` application are:
- A. Goto (context,extension)
 - B. Goto (context,extension,priority)
 - C. Goto (extension,priority)
 - D. Goto(priority)
11. Switches are used to direct the dial plan processing to another server.
- A. False
 - B. True
12. A macro can be used to automate the processing of an extension. The first macro argument is:
- A. `${ARG1}`
 - B. `${ENV1}`
 - C. `${V1}`
 - D. `${X}`

10

Using PBX features

In SIP systems, most of the phone features are implemented in the endpoint. A variety of SIP phones and manufacturers exist, and the interoperability is not guaranteed. The Asterisk development team has done an amazing job of implementing most of the features in the PBX itself, making Asterisk almost endpoint independent. However, sometimes you will find the same function being done by both the phone and Asterisk itself. The integration of the phone and the PBX is the next frontier on usability and where proprietary systems are focusing right now. In this chapter, you will learn how to use most of these features.

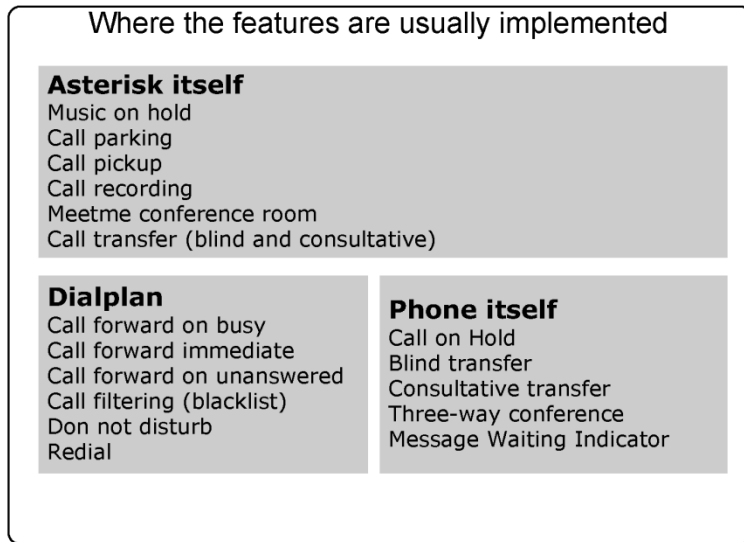
Objectives

By the end of this chapter, you will be able to understand and use:

- Call Parking
- Call Pickup
- Call Transfer
- Call Conference
- Call Recording
- Meetme
- Music on hold

Where features are implemented

First and foremost, it is important to understand when the PBX features are being executed versus when the phone is doing all the work. For example, you may transfer a call using the TRANSFER button on the phone or by dialing # (unconditional transfer executed by the PBX itself).



Features implemented by Asterisk

These features are implemented in the PBX by the Asterisk code:

- Music on hold
- Call parking
- Call pickup
- Call recording
- Meetme conference room
- Call transfer (blind and consultative)

Features usually implemented by the dial plan

These features need to be programmed in the Asterisk dial plan ([extensions.conf](#)):

- Call forward on busy
- Call forward immediate
- Call forward on unanswered
- Call filtering (blacklist)
- Do not disturb
- Redial

Features usually implemented by the phone

These features are implemented by the phone's firmware:

- Call on hold
- Blind transfer
- Consultative transfer
- Three-way conference
- Message waiting indicator

The features configuration file

Some of the features presented in this chapter are configured in `features.conf` configuration file. It is possible to change the behavior of some features by modifying this file. We have included the entire sample file below. In the next sections of this chapter, we will describe each feature.

```

PBX features support

[featuremap]
;blindxfer => #1      ;Blind transfer (default is #)
;disconnect => *0    ;Disconnect (default is *)
;automon => *1       ; One Touch Record
;atxfer => *2        ; Attended transfer
;parkcall => #72     ; Park call (one step parking)

```

Excerpt from the sample file (Asterisk 1.6.1)

```

[general]
parkext => 700          ; what extension to dial to park      (all parking lots)
parkpos => 701-720     ; what extensions to park calls on. (default parking lot)
                               ; These needs to be numeric, as Asterisk starts from the start

position                ; and increments with one for the next parked call.

context => parkedcalls  ; which context parked calls are in (default parking lot)
;parkinghints = no     ; Add hints priorities automatically for parking slots (default is
no).
;parkingtime => 45     ; Number of seconds a call can be parked for
                               ; (default is 45 seconds)
;comebacktoorigin = yes ; whether to return to the original calling extension upon parking
                               ; timeout or to send the call to context 'parkedcallsttimeout' at
                               ; extension 's', priority '1' (default is yes).

;courtesytone = beep   ; sound file to play to the parked caller
                               ; when someone dials a parked call
                               ; or the Touch Monitor is activated/deactivated.

;parkedplay = caller   ; who to play the courtesy tone to when picking up a parked call
                               ; one of: parked, caller, both (default is caller)

;parkedcalltransfers = caller ; Enables or disables DTMF based transfers when picking up a parked
call.
                               ; one of: callee, caller, both, no (default is no)

;parkedcallreparking = caller ; Enables or disables DTMF based parking when picking up a parked
call.
                               ; one of: callee, caller, both, no (default is no)

;parkedcallhangup = caller ; Enables or disables DTMF based hangups when picking up a parked
call.
                               ; one of: callee, caller, both, no (default is no)

```

```
;parkedcallrecording = caller ; Enables or disables DTMF based one-touch recording when picking up a
parked call.
; one of: callee, caller, both, no (default is no)
;adsipark = yes ; if you want ADSI parking announcements
;findslot => next ; Continue to the 'next' free parking space.
; Defaults to 'first' available
;parkedmusicclass=default ; This is the MOH class to use for the parked channel
; as long as the class is not set on the channel directly
; using Set(CHANNEL(musicclass)=whatever) in the dialplan
;transferdigittimeout => 3 ; Number of seconds to wait between digits when transferring a call
; (default is 3 seconds)
;xfersound = beep ; to indicate an attended transfer is complete
;xferfailsound = beeperr ; to indicate a failed transfer
;pickupexten = *8 ; Configure the pickup extension. (default is *8)
;featuredigittimeout = 1000 ; Max time (ms) between digits for
; feature activation (default is 1000 ms)
;atxfernoanswertimeout = 15 ; Timeout for answer on attended transfer default is 15 seconds.
;atxferdropcall = no ; If someone does an attended transfer, then hangs up before the
transferred
; caller is connected, then by default, the system will try to call back
the
will ; person that did the transfer. If this is set to "yes", the callback
; not be attempted and the transfer will just fail.
;atxferloopdelay = 10 ; Number of seconds to sleep between retries (if atxferdropcall = no)
;atxfercallbackretries = 2 ; Number of times to attempt to send the call back to the transferer.
; By default, this is 2.
; Note that the DTMF features listed below only work when two channels have answered and are bridged
together. They can not be used while the remote party is ringing or in progress. If you require this
feature you can use
; chan_local in combination with Answer to accomplish it.
[featuremap]
;blindxfer => #1 ; Blind transfer (default is #) -- Make sure to set the T and/or t
option in the Dial() or Queue() app call!
;disconnect => *0 ; Disconnect (default is *) -- Make sure to set the H and/or h option
in the Dial() or Queue() app call!
;automon => *1 ; One Touch Record a.k.a. Touch Monitor -- Make sure to set the W
and/or w option in the Dial() or Queue() app call!
;atxfer => *2 ; Attended transfer -- Make sure to set the T and/or t option in the
Dial() or Queue() app call!
;parkcall => #72 ; Park call (one step parking) -- Make sure to set the K and/or k option in
the Dial() app call!
;automixmon => *3 ; One Touch Record a.k.a. Touch MixMonitor -- Make sure to set the X
and/or x option in the Dial() or Queue() app call!
```

Call Transfer

Call transfer can be implemented by the phone, by ATA, or by Asterisk itself. Refer to your phone manual to understand how calls are transferred. If your phone does not support call transfer, you can use Asterisk to accomplish this task.

Call Transfer

Blind Transfer

- Needs to be enabled in the Dial() application using t or T option.
- Press # during the call
- Enter the extension to transfer to
- Hangup
- If the extension does not answer, the call will ring back.

Attended Transfer

- Enable in the features.conf (is disabled by default)
- Hit *2 to start the transfer
- Dial to the destination extension
- Talk to the destination extension
- Hangup
- The call is bridged between the caller and the transferee

Call transfer is implemented in two different ways. The first way is to use the blind transfer feature: dial # followed by the number to be transferred. Sometimes you will use the transfer feature of your IP phone or IP soft phone. You can change the transfer character by editing the `blindxfer` parameter in the `features.conf` file.

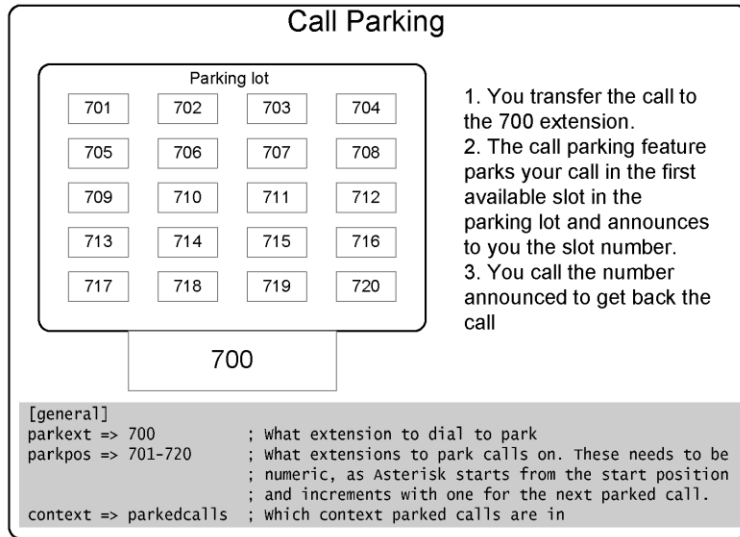
You can enable assisted transfer in Asterisk by removing the `;` before the `atxfer` parameter in the `features.conf` file. During a conversation, you would press *2. Asterisk will say “transfer” and will give you a dial tone. The caller is sent to music on hold. After you speak to the destination person and hang up the phone, the system bridges the caller to the destination.

Configuration task list

1. If the phone is SIP based, make sure that the option `canreinvite` is equal to `no` or use a `t` or `T` option in the `dial()` application

Call parking

This feature is used to park a call. This helps, for example, when you are answering a phone call outside of your room and you want to transfer the call back to your desk. You may accomplish this by parking the call in an extension. Once you reach your desk, simply dial the number of the parking extension to recover the call.



1. You transfer the call to the 700 extension.
2. The call parking feature parks your call in the first available slot in the parking lot and announces to you the slot number.
3. You call the number announced to get back the call

By default, the 700 extension is used to park a call. In the middle of a conversation, press # to transfer the call to the 700 extension. Now the Asterisk will announce your parking extension, such as 701 or 702. Hang up the phone, and the caller will be placed on hold. Go to your desk phone and dial the announced parking extension to recover the call. If the caller is parked for a long time, the timeout feature will trigger and the original dialed extension will ring again.

Configuration task list

Follow the steps below to enable call parking.

Step 1: Enable call parking: (required)

In the `extensions.conf` file, type the following statement.

```
include=>parkedcalls
```

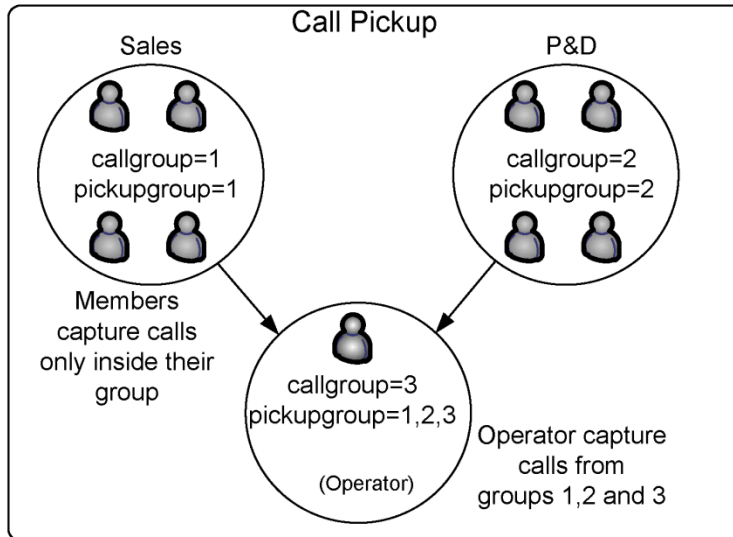
Step 2: Test the call parking feature by dialing #700.

Notes:

- The parking extension won't be shown in the `dialplan show` CLI command.
- It is necessary to restart Asterisk after changing the `features.conf` file. A simple reload won't work.
- To park a call, you need to transfer to #700. Verify the options `t` and `T` in the `dial()` application.

Call pickup

Call pickup allows you to capture a call from a colleague in the same call group. This would help avoid, for example, having to wake up to take a call that is ringing to another person in your room, but who is not present.



By dialing ***8**, you can capture a call within your call group. This number can be modified in the `features.conf` file.

Configuration task list

Follow the steps below to configure the call pickup feature.

Step 1: Configure a call group for your extensions. This is done in the channel configuration file (`sip.conf`, `iax.conf`, `chan_dahdi.conf`). This task is required.

```
[4x00]
callgroup=1
pickupgroup=1,2
```

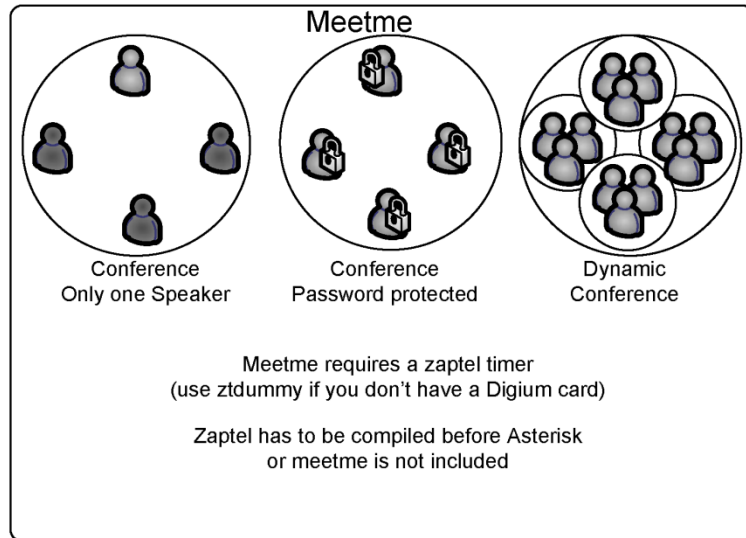
Step 2: Change the call-pickup feature number (optional).

```
pickuptexten=*8; configures the call pickup extension
```

Call Conference (Meetme)

Some SIP phones implement three-way conferencing in the phone itself. If this is the case, you may choose to use the phone feature. Refer to the phone manual for instructions on conference calling.

Alternatively, you may choose to use the `meetme()` application. Meetme is a conference bridge that is very simple to use. It works with any type of channel and is the standard method for conference in the Asterisk platform. Let's look at this feature in more depth.



The `meetme()` application

Using the `meetme show` CLI command, you can obtain the description above. To use `meetme`, you need to compile the DAHDI drivers and have at least one DAHDI kernel module loaded. If you don't have at least one DAHDI card installed, load the `dahdi_dummy` kernel module to provide a timing source.

```

Meetme()
MeetMe conference bridge
[Description]
MeetMe([confno][,[options][,pin]])

Main Options
'a' -- set admin mode
'c' -- announce user(s) count on joining a conference
'd' -- dynamically add conference
'D' -- dynamically add conference, prompting for a PIN
'e' -- select an empty conference
'E' -- select an empty pinless conference
'i' -- announce user join/leave with review
'I' -- announce user join/leave without review
'l' -- set listen only mode (Listen only, no talking)
'm' -- set initially muted
'M' -- enable music on hold when the conference has a single caller
'q' -- quiet mode (don't play enter/leave sounds)
't' -- set talk only mode. (Talk only, no listening)
'1' -- do not play message when first person enters

```

Description:

The `meetme()` application gets the user into a specified `meetme` conference. If the conference number is omitted, the user will be prompted to enter one. The user can leave the conference by hanging up or—if the `p` option is specified—by pressing `#`.

Please note: The DAHDI kernel modules and at least one hardware driver (or `dahdi_dummy`) must be present for conferencing to operate properly. In addition, the `chan_dahdi` channel driver must be loaded for the `i` and `r` options to operate at all.

The option string may contain zero or one or more of the following characters:

- 'a' -- sets admin mode
- 'A' -- sets marked mode
- 'b' -- runs the AGI script specified in `/${MEETME_AGI_BACKGROUND}`Default: `conf-background.agi` (Note: This does not work with non-Zap channels in the same conference)
- 'c' -- announces user(s) count upon joining a conference
- 'd' -- dynamically adds conference
- 'D' -- dynamically adds conference, prompting for a PIN
- 'e' -- selects an empty conference
- 'E' -- selects an empty pinless conference
- 'i' -- announces a user joining/leaving with review
- 'I' -- announces a user joining/leaving without review
- 'l' -- sets listen only mode (Listen only, no talking)
- 'm' -- sets initially muted
- 'M' -- enables music on hold when the conference has a single caller

- 'o' -- sets talker optimization, which treats talkers who aren't speaking as being muted, meaning (a) no encode is done on transmission and (b) received audio that is not registered as talking is omitted, causing no buildup in background noise
- 'p' -- allows users to exit the conference by pressing '#'
- 'P' -- always prompts for the pin even if it is specified
- 'q' -- quiet mode (don't play enter/leave sounds)
- 'r' -- Records conference (records as \${MEETME_RECORDINGFILE} using format \${MEETME_RECORDINGFORMAT}). Default filename is meetme-conf-rec-\${CONFNO}-\${UNIQUEID} and the default format is wav.
- 's' -- Presents menu (user or admin) when '*' is received ('send' to menu)
- 't' -- sets talk only mode. (Talk only, no listening)
- 'T' -- sets talker detection (sent to manager interface and meetme list)
- 'w[(<secs>)]' -- waits until the marked user enters the conference
- 'x' -- closes the conference when the last marked user exits
- 'X' -- allows the user to exit the conference by entering a valid single digit extension \${MEETME_EXIT_CONTEXT} or the current context if that variable is not defined.
- 'l' -- does not play message when the first person enters

Meetme configuration file

This file is used to configure the application meetme. For example:

```
;
; Configuration file for MeetMe simple conference rooms for Asterisk of course.
;
; This configuration file is read every time you call app meetme()

[general]
;audiobuffers=32          ; The number of 20ms audio buffers to be used
                          ; when feeding audio frames from non-Zap channels
                          ; into the conference; larger numbers will allow
                          ; for the conference to 'de-jitter' audio that arrives
                          ; at different timing than the conference's timing
                          ; source, but can also allow for latency in hearing
                          ; the audio from the speaker. Minimum value is 2,
                          ; maximum value is 32.

;
[rooms]
;
; Usage is conf => confno[,pin][,adminpin]
;
conf=>9000
conf=>9001,123456
```

It is not necessary to use either reload or restart to make Asterisk see the changes in the `meetme.conf` file.

Meetme-related applications

The `meetme()` application has two other support applications.

`MeetMeCount(confno[|var])`

This plays the number of users in the conference. If a variable is specified, it does not play the message but sets the number of users to it.

`MeetMeAdmin(confno,command,[user]):`

Run the admin command for a conference:

- 'e' -- Ejects the last user that joined
- 'k' -- Kicks one user out of the conference
- 'K' -- Kicks all users out of the conference
- 'l' -- Unlocks the conference
- 'L' -- Locks the conference
- 'm' -- Unmutes one user
- 'M' -- Mutes one user
- 'n' -- Unmutes all users in the conference
- 'N' -- Mutes all non-admin users in the conference
- 'r' -- Resets one user's volume settings
- 'R' -- Resets all users' volume settings
- 's' -- Lowers entire conference speaking volume
- 'S' -- Increases entire conference speaking volume
- 't' -- Lowers one user's talk volume
- 'T' -- Lowers all users' talk volume
- 'u' -- Lowers one user's listen volume
- 'U' -- Lowers all users' listen volume
- 'v' -- Lowers entire conference listening volume
- 'V' -- Increases entire conference listening volume

Meetme configuration task list

Follow the steps below to configure the `meetme` conferencing application.

Step 1: Choose the extension for the Meetme room (required)

Step 2: Edit the `meetme.conf` file to configure the passwords (optional)

Examples

Example #1: Simple meetme room

1. In the `extensions.conf` file, create the conference room 101

```
exten=>500,1,MeetMe(101,,123456)
```

2. In the `meetme.conf` file, establish the password for room 101.

Important Note: The `meetme()` application needs a timer to work. If you don't have digium hardware installed and configured, use `dahdi_dummy` as a timing source.

Call Recording

There are several ways to record a call in Asterisk. You can use the `mixmonitor()` application to easily record calls.

Using the mixmonitor application

The `mixmonitor` application records the audio in the current channel to the specified file. If the filename is an absolute path, it uses that path. Otherwise, it creates the file in the configured monitoring directory from `asterisk.conf`.

Mixmonitor()

Record a call and mix the audio during the recording

[Description]

MixMonitor(<file>.<ext>[|<options>[|<command>]])

Records the audio on the current channel to the specified file.

options:

- a- Append to the file instead of overwriting it.
- b-Only save audio to the file while the channel is bridged.
- Note: does not include conferences.
- v(<x>) - Adjust the heard volume by a factor of <x>
- V(<x>) - Adjust the spoken volume by a factor of <x>
- W(<x>) - Adjust both heard and spoken volumes

Valid options:

- a - Appends to the file instead of overwriting it.
- b - Only saves audio to the file while the channel is bridged.
- Note: does not include conferences.
- v(<x>) - Adjusts the audible volume by a factor of <x> (ranging from -4 to 4)
- V(<x>) - Adjusts the spoken volume by a factor of <x> (ranging from -4 to 4)

- `W(<x>)` - Adjusts both audible and spoken volumes by a factor of `<x>` (ranging from -4 to 4)
- `<command>` will be executed when the recording is over. Any strings matching `^{X}` will be unescaped to `#{X}` and all variables will be evaluated at that time. The variable `MIXMONITOR_FILENAME` will contain the filename used to record.

An interesting resource is `automon`, which allows you to simply dial `*1` to immediately start recording.

Example:

```
exten=>_4XXX,1,Set(DYNAMIC_FEATURES=automon)
exten=>_4XXX,2,Dial(SIP/${EXTEN},20,jtTww);ww enables the recording.
```

The audio channels are incoming (IN) and outgoing (OUT) and are separated into two distinct files in the directory `/var/spool/asterisk/monitor`. Both files can be mixed using the `sox` application.

```
debian#soxmix *in.wav *out.wav output.wav
```

If you don't want to use `Set()` before the `Dial()` application, you can set this in the `globals` section:

```
[globals]
DYNAMIC_FEATURES=>automon
```

Music on hold

Music on hold (MOH) has changed several times among versions 1.0, 1.2, and 1.4. In the latest version, MOH defaults to "FILE-BASED". In other words, Asterisk will supply the MOH files in formats such as `g729`, `alaw`, `ulaw`, and `gsm`. Thus, it is not necessary to transcode the music before sending it to the channel. This saves processor time, which is a welcomed modification for those working with production systems. In older versions, MOH was usually provided by MP3 (it still can be configured that way). Providing MOH using MP3 obligates Asterisk to transcode, spending valuable CPU power in the process.

MOH – Music on Hold

```
;
; Music on Hold -- Sample Configuration
;
;[class]
;mode=<mode>
;directory=<directory>
;format=<asteriskformat>
;application=<application to be executed and arguments>
;random=yes/no
;
; valid mode options:
; quietmp3    -- default
; mp3         -- loud
; mp3nb      -- unbuffered
; quietmp3nb  -- quiet unbuffered
; custom      -- run a custom application (See examples below)
; files       -- read files from a directory in any Asterisk supported
;              media format. (See examples below)
;
```

The new configuration file is shown below. Note that the default class now uses the native file format **mode=files**. All other modes are commented.

Each section is a class. The only uncommented class at this point is default. If you want to have different classes for different files, you will need to create new sections (classes).

```
; Music on Hold -- Sample Configuration

;[samplemp3]
;mode=quietmp3
;directory=/var/lib/asterisk/mohmp3
;
; valid mode options:
; quietmp3    -- default
; mp3         -- loud
; mp3nb      -- unbuffered
; quietmp3nb  -- quiet unbuffered
; custom      -- run a custom application (See examples below)
; files       -- read files from a directory in any Asterisk supported
;              media format. (See examples below)
;

;[manual]
;mode=custom
; Note that with mode=custom, a directory is not required, such as when reading
; from a stream.
;directory=/var/lib/asterisk/mohmp3
;application=/usr/bin/mpg123 -q -r 8000 -f 8192 -b 2048 --mono -s

;[ulawstream]
;mode=custom
```

```

;application=/usr/bin/streamplayer 192.168.100.52 888
;format=ulaw

; mpg123 on Solaris does not always exit properly; madplay may be a better
; choice
;[solaris]
;mode=custom
;directory=/var/lib/asterisk/mohmp3
;application=/site/sw/bin/madplay -Q -o raw:- --mono -R 8000 -a -12
;

;
; File-based (native) music on hold
;
; This plays files directly from the specified directory, no external
; processes are required. Files are played in normal sorting order
; (same as a sorted directory listing), and no volume or other
; sound adjustments are available. If the file is available in
; the same format as the channel's codec, then it will be played
; without transcoding (same as Playback would do in the dialplan).
; Files can be present in as many formats as you wish, and the
; 'best' format will be chosen at playback time.
;
; NOTE:
; If you are not using "autoload" in modules.conf, then you
; must ensure that the format modules for any formats you wish
; to use are loaded _before_ res_musiconhold. If you do not do
; this, res_musiconhold will skip the files it is not able to
; understand when it loads.
;

[default]
mode=files
directory=/var/lib/asterisk/moh
;
;[native-random]
;mode=files
;directory=/var/lib/asterisk/moh
;random=yes      ; Play the files in a random order

```

MOH configuration tasks

Now, to use music on hold, set the MOH class in the channel configuration files ([chan_dahdi.conf](#), [sip.conf](#), [iax.conf](#), and so on). The freeplay tunes installed are now in wav format. At the time of installation, you can select (using [make menuselect](#)) the MOH file formats available. If you want to add new MOH files, you will have to supply them in the required formats. For example:

In [/etc/asterisk/chan_dahdi.conf](#), add the line:


```
[channels]
musiconhold=default
```

Edit the file `/etc/asterisk/musiconhold.conf`

```
[default]
mode=files
directory=/var/lib/asterisk/moh
```

In the dial plan, you can hear the MOH using the following example:

```
Exten=>100,1,SetMusicOnHold(default)
Exten=>100,2,Dial(Zap/2)
```

To configure the file `extensions.conf` to test the MOH:

```
[local]
exten => 6601,1,waitMusicOnHold(30)
```

Application Maps

In version 1.2, it became possible to add new features by using the application maps section of the `features.conf` file. Suppose you need to identify the type of customer you are answering to a call center. You could create an application map for each customer type, which could count the number of answered customers per type.

Application MAP

Using this application an attendant can press, during the call, #8 to identify the caller as a costumer or #9 to identify the costumer as a partner

```
[applicationmap]
; Note that the DYNAMIC_FEATURES channel variable must be set to use the features
; defined here. The value of DYNAMIC_FEATURES should be the names of the features
; to allow the channel to use separated by '#'. For example:
;
; Set(DYNAMIC_FEATURES=myfeature1#myfeature2#myfeature3)
; Example Usage:
; costumer=>#8, self, Set,DB(costumer/counter)=${DB(costumer/counter)}+1]
; partner=>#9, self, Set,DB(partner/counter)=${DB(partner/counter)}+1]
```

Quiz

1. Which of the following statement(s) is true about Call Parking?

- A. By default, extension 800 is used for Call Parking
- B. When you are not at your desk and receive a call, you can park a call. The system will announce to you the parking extension. When you return to your desk, dial the announced extension to retrieve the call.
- C. By default, extension 700 is used for Call Parking. Calls are parked in extensions 701 to 720.

- D. You need to dial 700 to retrieve a parked call.
2. To use the Call Pickup feature, all extensions are required to be in the same _____. For ZAP channels, this is configured in the _____ file.
3. When transferring a call, you may choose between _____, where the destination extension is not consulted before the transfer, and _____, where you first talk to the destination extension before the transfer.
4. To make a consultative transfer, use the ___ character; for a blind transfer, use ____.
- A. #1, *2
 - B. *2, #1
 - C. #2, #1
 - D. #1, #2
5. To enable conference calls in the Asterisk server, it is necessary to use the _____ application.
6. If you have to supervise a conference, you can use the _____ application.
- A. MeetMe()
 - B. MeetMeConsole()
 - C. MeetMeAdministrator()
 - D. MeetmeAdmin()
7. The best format for music on hold is MP3 because it uses very little processing power from the Asterisk server.
- A. True
 - B. False
8. To capture a call from a specific call group, you need to be in their respective _____ group.
9. You can record a call using the `mixmonitor()` utility or the `automon` feature. By default, the `automon` feature uses the ___ character sequence.
- A. *1
 - B. *2
 - C. #3
 - D. #1
10. In the `meetme` application, if you want to have users in the listening only mode, you should:
- A. Merge different conference rooms with different options.
 - B. This is not possible using Asterisk.
 - C. Enable an extension that calls the `meetme` application with the `l` option and instruct the listening users to call that extension.
 - D. Enable an extension that calls the `meetme` application with the `t` option and instruct the listening users to call that extension.

11

Call Queues

Call queues are becoming increasingly important for answering customer calls efficiently. An automatic call distributor can help reduce costs, increase service, and improve sales as call distributors affect how your business works—not for a few days, but for many years. In a call center environment, the number one factor is people; they are the most expensive resource. It takes time, money, and patience to hire, train, and motivate agents. With an ACD, you can maximize agents' productivity by precisely dimensioning the number of agents required, controlling good and bad attendants, and analyzing the call flow.

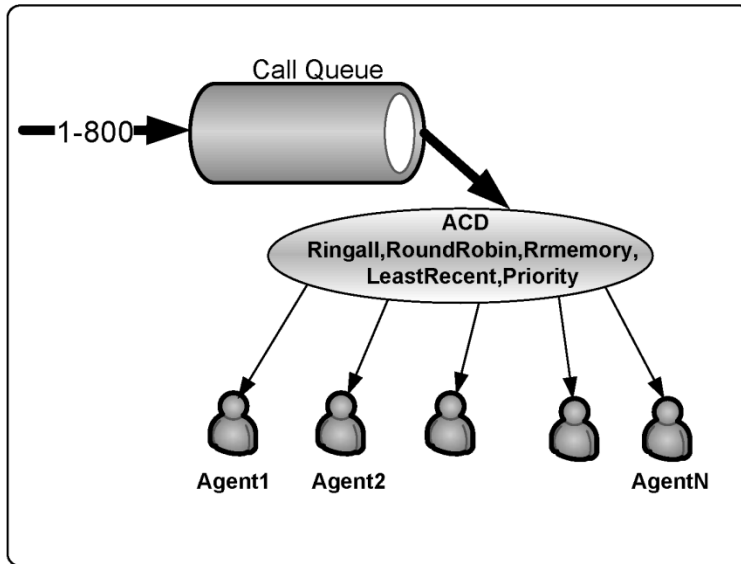
Objectives

By the end of this chapter, you should be able to:

- Understand why and how to use call queues
- Understand the basic theory of call queues
- Install and configure the queue system

How queues work?

Call queues are not exactly a novelty. When you have a high inbound call flow, it is hard to distribute calls appropriately. Using a group strategy where the phone simultaneously rings on all agents does not seem to work, unless you have only a few agents. However, a call queue will only deliver calls to a single available agent each time and put the customer on hold with music when there are no agents available. The queue works by retaining the call while finding an unoccupied agent to answer the call. One of the biggest benefits of the queue is to avoid losing calls while providing the possibility to generate statistics.



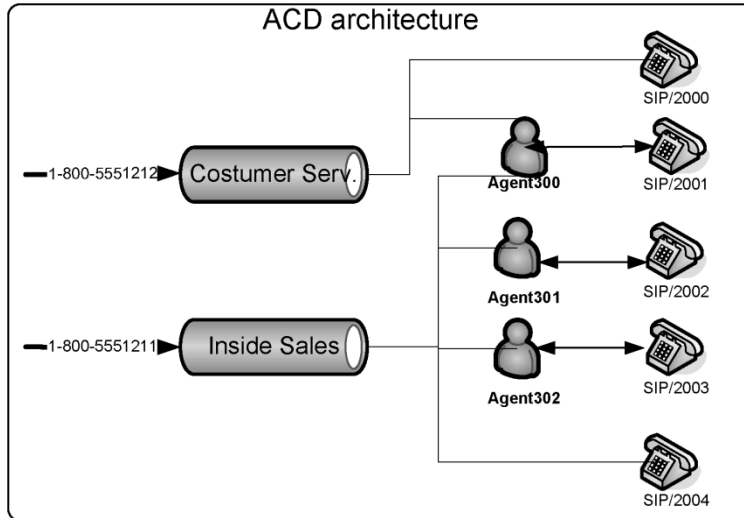
Usually, a call queue works like this:

- Calls are queued.
- Logged-in agents answer the queue.
- A queuing strategy to distribute the calls is used.
- Music on hold is played while the caller waits.
- Announcements can be made to callers, notifying them of waiting time.

The main application for queues is customer service. When using queues, you avoid losing calls when your agents are busy. You can add new agents to the queue if you find that the number of callers in the queue is growing. Another advantage is that, with queues you can now have statistics like call abandon rate, average call duration, and call answering target. These statistics will help you determine how many agents to use to provide better service to your customer.

ACD architecture

The ACD architecture is formed by queues and agents. One agent can be in two queues at the same time. A queue can have agents, channels, and agent groups.



Queues

Queues are defined in the `queues.conf` configuration file. Agents are attendants who log in and are members of queues. Agents are defined in the `agents.conf` file. In version 1.4, the queue system was largely evolved, making the configuration file huge. We will explain some of the major parameters.

General parameters

```
autofill=yes
```

The old behavior for the queue was serial type. The queue waited for a call to be dispatched before sending the succeeding call to the next agent. If an agent takes 15 seconds to answer a call, the other calls in the queue had to wait until that call was answered. For high-volume queues, this behavior was inefficient. The new behavior `autofill=yes` does not wait until a call is answered, but rather works in parallel.

Queue configuration file

Queues are configured in the `queues.conf` file. In the figure, you will find a working example of a queue.

queues.conf

```
[general]
persistentmembers = yes
autofill = yes
monitor-type = MixMonitor
[costumerservice]
musicclass = default
announce = queue-costumerservice
strategy = rrmemory
servicelevel = 60
context = costumerservice
timeout = 15
retry = 5
wrapuptime=15
announce-frequency = 90
periodic-announce-frequency=60
announce-holdtime = yes
monitor-format = wav
member => Agent/301,301
Member => Agent/300,300
```

Members

Members are active channels responding to the queue. Members can be direct channels (sip, dahdi, mgcp) or agents who log in before receiving calls.

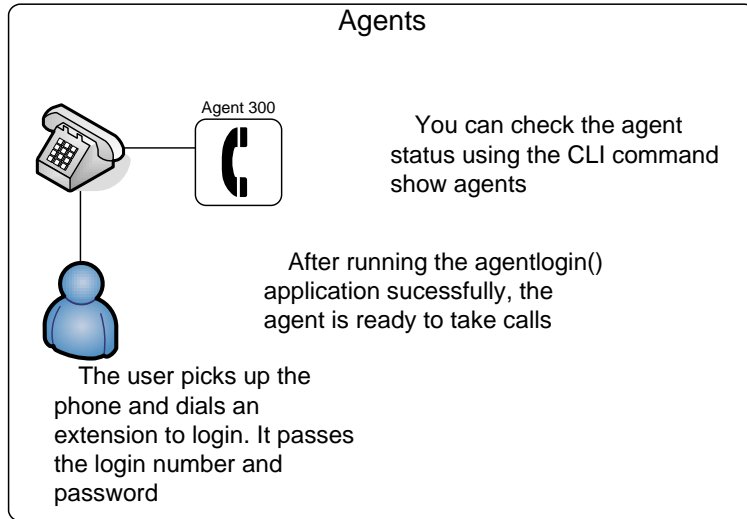
Strategies

Calls are distributed among members according to one of these strategies:

- **ringall:** Plays all channels available until someone answers.
- **roundrobin:** Distributes equally between members.
- **leastrecent:** Distributes to the least recent member.
- **fewestcalls:** Distributes to the member with fewest calls.
- **random:** Ring random interface.
- **wrandom:** Ring random interface, but use the member's penalty as a weight when calculating their metric.
- **rrmemory:** Uses round robin with memory; it remembers where it let off with the call in the last pass.

Agents

Agents are implemented as proxy channels. They can be used inside the queues. Another use for the agent channels is extension mobility. The user can log in using any phone and receive its calls. This allows a user to go to any room to make it an office. You can dial an agent in the dial plan using `dial(agent/<name>)`. You define agents in the `agents.conf` file.



Agent Groups

You may choose to use agent groups. This function does not take ACD strategies into consideration. You will probably prefer to list all agents individually. If you want to transfer to an agent group, you can use `queues.conf`:

```
member=>agent/@1 ;any agent in group 1
member=>agent/:1,1 ;any agent in group 1, wait for first available, ;do not
use agent groups.
```

The configuration file for agents

Agents are defined in the file `agents.conf`. Below is a working example of the file.

```
agents.conf
```

```
; Agent configuration

[general]
persistentagents=yes

[agents]
autologoff=15
autologoffunavail=yes
ackcall=no
endcall=yes
wrapuptime=5000
musiconhold => default
;
;This section contains the agent definitions, in the form:
;
; agent => agentid,agentpassword,name
;
agent => 300,300
agent => 301,301
```

ACD-related applications

The Asterisk queue system makes several applications available to implement the queues in the dial plan. Below, we show some of them.

The application queue()

This applications queues incoming calls into a particular call queue as defined in `queues.conf`. The option string may contain zero or more of the following characters:

The queue() application

Queue a call for a call queue

[Description]

Queue(queueename[|options[|URL][|announceoverride][|timeout][|AGI]):

Options:

- 'd' -- data-quality (modem) call (minimum delay).
- 'h' -- allow callee to hang up by hitting *.
- 'H' -- allow caller to hang up by hitting *.
- 'n' -- no retries on the timeout; will exit this application.
- 'i' -- ignore call forward requests from queue members and do nothing
- 'r' -- ring instead of playing MOH
- 't' -- allow the called user transfer the calling user
- 'T' -- allow the calling user to transfer the call.
- 'w' -- allow the called user to write the conversation to disk via Monitor
- 'W' -- allow the calling user to write the conversation to disk via Monitor

In addition to transferring the call, a call may be parked and then picked up by another user. The optional URL will be sent to the called party if the channel supports it. The optional AGI parameter will set up an AGI script to be executed on the calling party's channel once they are connected to a queue member. The timeout will cause the queue to fail out after a specified number of seconds, checked between each timeout and retry cycle. This application sets the `QUEUE` status variable upon completion:

- TIMEOUT
- FULL
- JOINEMPTY
- LEAVEEMPTY
- JOINUNAVAIL
- LEAVEUNAVAIL

The application agentlogin()

This application asks the agent to log in to the system. It always returns -1. While logged in, the agent receiving calls will hear a beep when a new call comes in. The agent can dump the call by pressing the * key.

The agentlogin() application

Call agent login

[Description]
AgentLogin([AgentNo][|options]):

Options:
's' silent login - do not announce the login ok

The application addQueueMember()

This application dynamically adds a device (e.g., SIP/3000) to a queue. If the device already exists, it will return an error.

AddQueueMember(queueName [|interface] [|penalty]):

The application removeQueueMember()

This application dynamically removes a device from the queue. If the device does not belong to the queue, it will return an error.

RemoveQueueMember(queueName [|interface])

Support applications and CLI commands

Some applications and console commands are capable of helping the work with queues. The following outlines what each application does:

Support applications and CLI commands

Support Applications

- **AddQueueMember**: Dynamically add a queue member
- **RemoveQueueMember**: Dynamically remove a queue member

CLI commands

- **show agents**: Show all agents.
- **show queues**: List all queues
- **show queue <name>**: Show na specific queue data

Configuration tasks

The figure below summarizes the major tasks to create a working queue system.

- ACD configuration tasks**

 1. Create the call queue (required)
 2. Define agent parameters (optional)
 3. Create agents (optional).
 4. Put the queue in the dial plan (required)
 5. Configure agent recording (optional)
 6. Verify using show agents e show queues(optional)

Step 1: Create the call queue

In the file `queues.conf`:

```
[telemarketing]
music = default
;announce = queue-telemarketing
;context = qoutcon
timeout = 2
retry = 2
maxlen = 0
member => Agent/300
member => Agent/301
[auditing]
music = default
;announce = queue-auditing
;context = qoutcon
timeout = 15
retry = 5
maxlen = 0
member => Agent/600
member => Agent/601
```

Step 2: Define agent parameters

In the file `agents.conf`:

```
debian:/etc/asterisk# cat agents.conf
;
; Agent configuration
;
[agents]
; Define maxlogintries to allow agent to try max logins before
; failed.
; default to 3
maxlogintries=5
; Define autologoff times if appropriate. This is how long
; the phone has to ring with no answer before the agent is
```

```
; automatically logged off (in seconds)
autologoff=15
; Define autologoffunavail to have agents automatically logged
; out when the extension that they are at returns a CHANUNAVAIL
; status when a call is attempted to be sent there.
; Default is "no".
;autologoffunavail=yes
; Define ackcall to require an acknowledgement by '#' when
; an agent logs in using agentcallbacklogin. Default is "no".
;ackcall=no
; Define endcall to allow an agent to hangup a call by '*'.
; Default is "yes". Set this to "no" to ignore '*'.
;endcall=yes
; Define wrapuptime. This is the minimum amount of time when
; after disconnecting before the caller can receive a new call
; note this is in milliseconds.
;wrapuptime=5000
; Define the default musiconhold for agents
; musiconhold => music_class
;musiconhold => default
;
; Define the default good bye sound file for agents
; default to vm-goodbye
;agentgoodbye => goodbye_file
; Define updatecdr. This is whether or not to change the source
; channel in the CDR record for this call to agent/agent_id so
; that we know which agent generates the call
;updatecdr=no
;
; Group memberships for agents (may change in mid-file)
;
;group=3
;group=1,2
;group=
```

Step 3: Create the agents

In the file **agents.conf**:

```
;agent => agentid,agentpassword,name
[agents]
agent => 300,300,Test Rep - 300
agent => 301,301,Test Rep . 301
agent => 600,600,Test Ver - 600
agent => 601,601,Test Ver . 601
```

Step 4: Insert the queue in the dial plan

In the file **extensions.conf**:

```
; Telemarketing queue.
```

```

exten=>_0800XXXXXXX,1,Answer
exten=>_0800XXXXXXX,2,SetMusicOnHold(default)
exten=>_0800XXXXXXX,3,Set(TIMEOUT(digit)=5)
exten=>_0800XXXXXXX,4,Set(TIMEOUT(response)=10)
exten=>_0800XXXXXXX,5,Background(welcome)
exten=>_0800XXXXXXX,6,Queue(telemarketing)

; Transfer to the queue auditing
exten => 8000,1,Queue,(auditing)
exten => 8000,2,Playback(demo-echotest); No auditor available
exten => 8000,3,Goto(8000,1) ; verify auditor again

; Agent login for the telemarketing and auditing queues

exten => 9000,1,wait(1)
exten => 9000,2,AgentLogin()

```

Configure queue recording

Calls may be recorded using Asterisk's `monitor` or `mixmonitor` application. Recording can be enabled from within the `queue` application, beginning when the call is actually picked up. Only successful calls are recorded, and no recordings are performed while people are listening to MOH. To enable monitoring, simply specify `monitor-format`. This feature is otherwise disabled.

You can set the filename for the recording using `Set (MONITOR_FILENAME=<filename>)`; otherwise it will use `MONITOR_FILENAME=${UNIQUEID}`.

In the file `queues.conf`:

```

monitor-format = wav
monitor-type = MixMonitor
monitor-join = yes

```

Queue operation

The following examples explains how to use the queue.

Step 1: Agent login

Example: An agent in the telemarketing queue picks up the phone and dials `#9000`. The agent hears an invalid login message and is asked for his/her name and password. The auditing queue follows the same procedure.

Step 2: Queue

Once in the queue, the agent will hear MOH, if defined. When a call comes in to the telemarketing queue, the agent will hear a beep and will be connected to that call.

Step 3: Call ending

When the agent finishes the call, he/she can:

- Press '*' to disconnect and stay in the queue.
- Disconnect the phone, thereby disconnecting from the queue.
- Press #8000 to transfer the call for auditing.

Advanced resources

The Asterisk queue system has some advanced features to prioritize certain customers and agents as well as enable a user menu.

User menu

You can define a menu for a user while waiting in the queue using one-digit extensions. To enable this option, define a context in the queue configuration `queues.conf`.

Penalty

Agents can be configured with a penalty. A queue will send the calls first to users with lower penalty values. For example, since we know that our customers love Susan and her soft voice, we may choose to assign priority 0 to her. Alternatively, the agent named Uber, who has less experience, is less preferred for customer service; therefore, we assign a priority 10 to this agent.

In the file `queues.conf`:

```
[customerservice]
member=300,0,Susan the excellent agent
member=300,10,Uber the new guy
```

Priority

Queues operate in the FIFO (first in first out) mode. If you want to give priority for special customers (platinum, gold) you can set up differentiated priorities.

For platinum or gold customers:

```
exten=>111,1,Playback(welcome)
exten=>111,2,Set(Queue_Prio=10)
exten=>111,3,Queue(customerservice)
```

Blue customers:

```
exten=>112,1,Playback(welcome)
exten=>112,2,Set(Queue_Prio=5)
exten=>112,3,Queue(customerservice)
```

The application `agentcallbacklogin()` is deprecated

The application `agentcallbacklogin()` has been deprecated in version 1.2 and is not available in versions 1.4 and 1.6. A document called `queues-with-callback-members.txt` is included in the

`/doc` directory of the Asterisk distribution. In this document, you will find detailed instructions on how to recreate the features of this application.

Queue statistics

All events from queues are logged to `/var/log/asterisk/queue_log`. The format of the queue log is published in the document `queuelog.txt` in the `/doc` directory of the Asterisk documentation. Below are some of the most important events logged.

- `ABANDON(position|origposition|waittime)`
- `AGENTDUMP`
- `AGENTLOGIN(channel)`
- `AGENTCALLBACKLOGIN(exten@context)`
- `AGENTLOGOFF(channel|logintime)`
- `AGENTCALLBACKLOGOFF(exten@context|logintime|reason)`
- `COMPLETEAGENT(holdtime|calltime|origposition)`
- `COMPLETECALLER(holdtime|calltime|origposition)`
- `CONFIGRELOAD`
- `CONNECT(holdtime|bridgedchanneluniqueid)`
- `ENTERQUEUE(url|callerid)`
- `EXITEMPTY(position|origposition|waittime)`
- `EXITWITHKEY(key|position)`
- `EXITWITHTIMEOUT(position)`
- `QUEUESTART`
- `RINGNOANSWER(ringtime)`
- `SYSCOMPAT`
- `TRANSFER(extension|context|holdtime|calltime)`

You can build your own utility to process these events or use a ready-to-run statistics package. We have tested two utilities on the asteriskguide:

- Qlog analyzer (<http://www.micpc.com/qlogalyzer/>) – Excellent open source package
- Queue metrics (<http://queuemetrics.com/>) – One of the most complete packages for queue stats

New in 1.6.2 for advanced users

In version 1.6.2, it is now possible to integrate the SIP presence with the queues. I see a lot of potential to double or even triple the number of users supported in incoming queues. These advanced features are beyond the scope of introductory material like this text. More information can be found at the Asterisk source at `/doc/building_queues.txt`. This excellent text, written by Leif Madsen, will teach you how to use hints to avoid sending calls to devices in use.

Summary

In this chapter you have learned how to use an ACD, its architecture, and how to configure it. Some advanced features such as priorities and penalties were also presented.

Quiz

1. Cite four strategies for routing a call in a queue.
2. It is possible to record a conversation between an agent and a customer using the statement _____ in the `queues.conf` file.
3. To log in an agent, you will use the application `agentlogin([agentnumber])`. When the agent finishes the call, he/she can:
 - A. disconnect and stay in the queue
 - B. hang up the phone and disconnect from the queue
 - C. press #8000 to transfer to call audit
 - D. press # to hang up
4. The required tasks to configure a call queue are:
 - A. Create the queue
 - B. Create the agents
 - C. Configure the agents
 - D. Configure the recording
 - E. Put the queue in the dial plan
5. What's the difference between the applications `AgentLogin()` and `AgentCallBackLogin()`.

6. When in a call queue, you can define a certain number of options that the user can dial. This is done by including a(n) _____ in the file `queues.conf`.

- A. Agent
- B. Menu
- C. Context
Application

7. The support applications `AddQueueMember()`, `AgentLogin()`, `AgentCallbackLogin`, and `RemoveQueueMember()` should be included in the _____.

- A. Dial plan
- B. Command line interface
- C. `queues.conf`
- D. `agents.conf`

8. It is possible to record the agents, but doing so requires an external recorder.

- A. True
- B. False

9. The parameter `wrapuptime` is the time the user needs after ending the call to complete business process related to that call.

- A. True
- B. False

10. A call can be prioritized depending on the caller ID inside the same queue.

- A. True
- B. False

12

Asterisk Call Detail Records

Asterisk, like other telephony platforms, allows the billing of phone calls. Several programs on the market can import the records generated by PBXs. Those records are used to verify the correct amount of the bill and statistics, among other things.

Objectives

By the end of this chapter, the reader should be able to:

- Describe where and in what format the records are generated
- Generate records in MySQL database
- Implement an authentication scheme integrated with billing

Asterisk CDR Format

Asterisk generates a call detail record (CDR) for each call. These records are stored, by default, in a text file in a comma separated value (CSV) in the `/var/log/asterisk/cdr-csv`. The file is organized in the following fields:

CDR	Description	Type	Size
Accountcode	Account Number to use	String	20
Src	Caller ID Number	String	80
Dst	Destination Extension	String	80
Dcontext	Destination Context	String	80
Clid	Caller ID with Text	String	80
Channel	Channel Used	String	80
Dstchannel	Destination channel	String	80
Lastapp	Last application	String	80
Lastdata	Last application data	String	80
Start	Start of call	Date/Time	

Answer	Answer of call	Date/Time	
End	End of Call	Date/Time	
Duration	Time, from dial to hang up (seconds)	Integer	
Billsec	Time, from answer to hang up (seconds)	Integer	
Disposition	What Happened to the call (ANSWERED, NO ANSWER, BUSY, FAILED)	String	20
Amaflags	Flags (DOCUMENTATION, BILLING, IGNORE)	String	20
User field	User defined field	String	255

Sample of csv file imported into a table.

AccountCode	CallerID No.	Extension	Context	CallerID text	Src	Dst
1234	4830258576	*72*1234*8584	admin	"Joana D'Arc" <4830258576>	SIP/8576-5f30	SIP/8584-9153
1234	4830258576	*72*1234*8584	admin	"Joana D'Arc" <4830258576>	SIP/8576-96f5	SIP/8584-3312
1234	4830258576	*72*1234*8584	admin	"Joana D'Arc" <4830258576>	SIP/8576-74ac	SIP/8584-297b
1234	4830258576	2012348584	admin	"Joana D'Arc" <4830258576>	SIP/8576-2c5d	SIP/8584-9870
1234	4830258584	2012348576	default	"Luis Sample" <4830258584>	SIP/8584-03fd	SIP/8576-645c

Application	Appdata	Start	Answer	End	Dur	Bil	Disposition	Amaflags
Dial	SIP/8584 30 tT	27/3/2006 16:05	27/3/2006 16:05	27/3/2006 16:05	5	3	ANSWERED	DOCUMENTATION
Dial	SIP/8584 30 tT	27/3/2006 16:16	27/3/2006 16:16	27/3/2006 16:16	6	4	ANSWERED	BILLING
Dial	SIP/8584 30 tT	27/3/2006 16:22	27/3/2006 16:22	27/3/2006 16:22	9	5	ANSWERED	BILLING
Dial	SIP/8584 30 tT	27/3/2006 16:37	27/3/2006 16:37	27/3/2006 16:37	5	2	ANSWERED	BILLING
Dial	SIP/8576 30 tT	27/3/2006 16:37	27/3/2006 16:37	27/3/2006 16:37	9	5	ANSWERED	BILLING

Account codes and automated message accounting

You can specify account codes and **ama** flags on each channel. Usually this is done in the channel configuration file (e.g., `chan_dahdi.conf`, `sip.conf`, `iax.conf`). The parameter `amaflags` defines what to do with the CDR record. The possible `amafld` values are:

- Default
- Omit
- Billing
- Documentation

Similar to the way in which a record can be flagged for billing or documentation, an account code can be set to each record. The account is a 20-character string usually used to assign a record to a department or business unit.

Example: `sip.conf` file

```
[8576]
amaflags=default
```

```
accountcode=Support
type=friend
username=8576
```

Changing the CSV and/or CDR format

You can change the CSV format by changing the `cdr_custom.conf` file.

```
;
; Mappings for custom config file
;
[mappings]
Master.csv =>
"${CDR(clid)}", "${CDR(src)}", "${CDR(dst)}", "${CDR(dcontext)}", "${CDR(channel)}"
, "${CDR(dstchannel)}", "${CDR(lastapp)}", "${CDR(lastdata)}", "${CDR(start)}", "${C
DR(answer)}", "${CDR(end)}", "${CDR(duration)}", "${CDR(billsec)}", "${CDR(disposit
ion)}", "${CDR(amaflags)}", "${CDR(accountcode)}", "${CDR(uniqueid)}", "${CDR(userf
ield)}"
```

You can change the CDR format in the `cdr_custom.conf` file.

CDR Storage

CDR storage can be achieved in several ways. The most important way is CSV text files that can be easily imported into spreadsheets. For small businesses, this is usually okay. Some billing software accepts, by default, CSV files. However, storing CDR in a database is a lot better and safer, and Asterisk supports several database flavors. There are some graphical interfaces for billing in the market. At our company, we have tested the open source versions of A2billing and AsteriskStats from Areski (www.areski.net). In my opinion, they are very good.

Storage drivers available

- `cdr_csv` – Comma Separated Value text files
- `cdr_sqlite` – SQLite databases
- `cdr_pgsq1` – Postgres databases
- `cdr_odbc` – unixODBC supported databases
- `cdr_mysql` – MySQL databases
- `cdr_freetds` – Sybase and MSSQL databases
- `cdr_yada` – yada databases
- `cdr_manager` – CDR to Manager Interface
- `cdr_radius` – CDR radius interface

CDR recording is done to all active modules loaded in the file `/etc/asterisk/modules.conf`. If the parameter `autoload=yes` is set, all modules are loaded.

CSV Storage

As we said before, by default, Asterisk sends all CDR to a CSV text file using the `cdr_csv.so` module. If you can't see the files in the `/var/log/asterisk/cdr-csv`, check to see if the module is being loaded using the CLI command `module show`. If it's not loaded, check `modules.conf`.

Storing in MySQL database

Due to licensing restrictions of MySQL, Digium cannot bundle the database with Asterisk. Consequently, MySQL support for CDR is in the package `asterisk-addons`. You will have to download, uncompress, and compile the module separately. Follow the instructions below to install MySQL Support.

Step 1: Install MySQL and MYSQL-devel packages.

```
apt-get install mysql-server-4.1
apt-get install libmysqlclient12-dev
cd /usr/src
wget http://ftp.digium.com/pub/asterisk/releases/asterisk-addons-1.4.1.tar.gz
tar -xzvf asterisk-addons-1.4.1
cd asterisk-addons-1.4.1
make clean
make
make install
```

Step 2: Make the necessary adjustments on `cdr_mysql.conf` file. This configuration has to point to **where the database will be located**.

In the file `cdr_mysql.conf`:

```
[global]
hostname=localhost
dbname=asteriskdb
password=asterisk
user=asterisk
port=3306
sock=/var/run/mysqld/mysqld.sock
;userfield=1
```

Use `vi` (or your preferred editor) to edit `modules.conf` to include the module `cdr_addon_mysql.so` for loading. In most cases, you don't need to do this as the option `autoload=yes` is default.

Step 3: Create a database for `cdr_addon_mysql`

```
mysql -p
```

or

```
mysql -u root -p (if you have a password for the root user)
```

Type the following commands to create the database.

```
CREATE DATABASE asteriskdb;

GRANT INSERT
ON asterisk.*
```

```
TO asterisk@localhost
IDENTIFIED BY 'asterisk';

USE asteriskdb;

CREATE TABLE `cdr` (
  `calldate` datetime NOT NULL default '0000-00-00 00:00:00',
  `clid` varchar(80) NOT NULL default '',
  `src` varchar(80) NOT NULL default '',
  `dst` varchar(80) NOT NULL default '',
  `dcontext` varchar(80) NOT NULL default '',
  `channel` varchar(80) NOT NULL default '',
  `dstchannel` varchar(80) NOT NULL default '',
  `lastapp` varchar(80) NOT NULL default '',
  `lastdata` varchar(80) NOT NULL default '',
  `duration` int(11) NOT NULL default '0',
  `billsec` int(11) NOT NULL default '0',
  `disposition` varchar(45) NOT NULL default '',
  `amaflags` int(11) NOT NULL default '0',
  `accountcode` varchar(20) NOT NULL default '',
  `userfield` varchar(255) NOT NULL default ''
);

ALTER TABLE `cdr` ADD INDEX ( `calldate` );
ALTER TABLE `cdr` ADD INDEX ( `dst` );
ALTER TABLE `cdr` ADD INDEX ( `accountcode` );
```

One tip is to copy and paste these commands into a text file named “cdr.sql” and execute the following command:

```
mysql --user=username --password=password asteriskdb <cdr.sql
```

Applications and functions

Several applications are related to billing.

CDR(accountcode)

Sets an account code before calling another application `dial()`; for example:

Format:

```
Set(CDR(accountcode)=account)
```

The account code can be verified using the channel variable ``${CDR(accountcode)}``

CDR(amaflags)

Set a flag for billing purposes. Options are default, omit, documentation, and billing.

```
Set(CDR(amaflags)=amaflags)
```

NoCDR()

No CDRs recorded to the file or database.

ResetCDR()

Resets the CDR to zero. If the **w** option is set, it saves the original CDR record.

Set(CDR(userfield)=Value)

This command sets a user field in the CDR. If you were using `cdr_addon_mysql`, check to see if you have the option `userfield=1` in the `cdr_mysql.conf`. For CSV text files, you have to edit the source code (`cdr_csv.c`) and recompile Asterisk if you want to use user fields. This command is useless if user fields are not enabled in the source code or in the MySQL configuration file, `cdr_mysql.conf`.

AppendCDRUserField(Value)

Append data to the user field on the CDR.

User authentication

Some companies bill the calls to their employees. In Asterisk you can set an authentication scheme that enables you to bill the authenticated user on the CDR. This authentication can be done using a password passed as a parameter to the Authenticate application—a password file, indicated by a **/** (slash) before the parameter or a Asterisk database (`dbput/dbget`).

Format:

```
Authenticate(password[|options])
Authenticate(/passwdfile|[|options])
Authenticate(</db-keyfamily|d>options)
```

Options:

- **a** – Sets the account code as the password.
- **d** – Interprets the parameter as a Asterisk DB key
- **r** – Removes the key after successful authentication (only with `'d'` option)
- **j** – Jumps to priority `n+101` for invalid authentication

Example: (International Calls)

```
exten=_9011.,1,Authenticate(/password|daj)
exten=_9011.,2,Dial(Zap/g1/${EXTEN:1},20,tt)
exten=_9011.,3,Hangup()
exten=_9011.,102,Playback(unauthorized)
exten=_9011.,103,Hangup()
```

To insert the password in a DB key from the console:

```
CLI> database put senha 123456 1
```

Using passwords from voicemail

This application does the same as `authenticate`, but uses the voicemail configuration file for the password.

```
VMAuthenticate([mailbox] [@context] [|options])
```

If a mailbox is specified, only the mailbox password will be considered valid. If the mailbox is not specified, a channel variable `AUTH_MAILBOX` will be set with the authenticated mailbox. If the option `'s'` (silent) is set, no prompt will be executed.

Example: (International Calls)

```
exten=_9011.,1,VMAuthenticate(${CALLERID(num)}@local|ajs)
exten=_9011.,2,Dial(Zap/g1/${EXTEN:1},20,tT)
exten=_9011.,3,Hangup()
exten=_9011.,102,Playback(unauthorized)
exten=_9011.,103,Hangup()
```

Summary

In this chapter we have learned how to implement CDR recording in text files and in a MySQL database as well as how to set `amaflags` and account codes. At the end of the chapter, we learned how to use an authentication scheme integrated with CDR and billing.

Quiz

1. By default, Asterisk records the CDR in `/var/log/asterisk/cdr-csv` directory.
 - A. False
 - B. True
2. Asterisk allows only these databases:
 - A. MySQL
 - B. Native Oracle
 - C. Microsoft SQL
 - D. CSV Text files
 - E. unix_ODBC supported databases
3. Asterisk generates a CDR only to a single kind of storage.
 - A. False
 - B. True
4. Which Asterisk `amaflags` are available?
 - A. Default
 - B. Omit

- C. Tax
 - D. Rate
 - E. Billing
 - F. Documentation
5. If you intend to associate a department to a CDR, you should use the command _____.
The account code can be verified using the channel variable _____.
6. The difference between the applications `NOCDR()` and `ResetCDR()` is that `NOCDR()` does not generate any record and `ResetCDR()` zeroes the current record.
- A. False
 - B. True
7. To use a user-defined field with the `cdr_csv.so` module, it is necessary to edit the source code and recompile the Asterisk.
- A. False
 - B. True
8. The three authentication methods available to the `Authenticate()` application are:
- A. Password
 - B. Password file
 - C. Asterisk DB (`dbput` and `dbget`)
 - D. Voicemail
9. Voicemail passwords are specified in a different section of the `voicemail.conf` file and are not the same as the voicemail users.
- A. False
 - B. True
10. This option authenticates the command using the password used to authenticate the CDR.
- A. a
 - B. j
 - C. d
 - D. r

13

Extending Asterisk with AMI and AGI

In several situations, it may be necessary to extend Asterisk features using external applications. With conventional PBXs, this was normally done using computer telephony integration (CTI) interface. Asterisk is built in a computer and not based on a circuit switch. Therefore, there are many different ways by which it may be extended. In this chapter, we will cover Asterisk's CTI interface, called Asterisk manager interface (AMI).

Since there are other ways to integrate Asterisk with other programs, we will also look at the command `asterisk -rx` and the `system()` application. At the end of this chapter, we will look at the powerful Asterisk gateway interface (AGI), which enables Asterisk to call external applications made with any languages that support Linux standard I/O, a much-used resource to build interactive voice responses (IVRs). The only drawback in Asterisk integration is that it does not have a standard CSTA interface, which could make it easier for other programs to port applications like dialers, report generators and others.

Objectives

By the end of this chapter, the reader should be able to:

- Describe access options to external programs
- Use `asterisk -rx` command to execute a console command
- Use `the system()` app to call external programs in the dial plan
- Explain what AMI is and how it works
- Configure the `manager.conf` file and enable AMI
- Execute an AMI command from a PHP program
- Explain what Asterisk manager proxy is and how it works
- Describe different AGI Flavors (DeadAGI, AGI, EAGI, FastAGI)
- Execute a simple AGI program created with PHP

Major ways to extend Asterisk

Asterisk has different ways to interface with external programs. In this chapter, we will cover:

- Linux command line and Asterisk Console
- System() Application
- AMI
- AGI

Extending Asterisk with console CLI

An application can easily call Asterisk from the Linux shell using the following command.

```
asterisk -rx <command>
```

Example:

```
asterisk -rx "stop now"
```

Even a command with an output can be called:

```
asterisk:~# asterisk -rx "sip show peers"
```

Name/username	Host	Dyn	Nat	ACL	Port	Status
4000/4000	10.1.1.6	D			5060	Unmonitored
1 sip peers [1 online , 0 offline]						

Extending Asterisk using the System() application

The `system()` application enables Asterisk to call an external application.

```
asterisk*CLI> show application system
asterisk*CLI>
  -= Info about application 'System' =-
```

[Synopsis]
Execute a system command

[Description]
System(command): Executes a command by using system(). If the command fails, the console should report a fallthrough. Result of execution is returned in the SYSTEMSTATUS channel variable:

FAILURE	Could not execute the specified command
SUCCESS	Specified command successfully executed

Example:

This application does a screen-pop using netbios WindowsPopup.

```
exten => 9000,1,system(/bin/echo -e "Incoming Call From -> ${CALLERID(num)}
\\r Received: ${DATETIME}'" | /usr/bin/smbclient -M target_netbiosname)
exten => 9000,2,Dial(SIP/9000,15,t)
exten => 9000,3,Hangup
```

What is AMI?

AMI enables a client program to connect to an Asterisk instance and issue commands or read events over a TCP connection. System integrators will find these resources useful for tracking channel states. AMI relies on a simple concept of a line protocol using `key:value` pairs over TCP. Asterisk by itself is not ready to handle too many connections over this interface. If you have lots of connections to AMI, consider using Asterisk manager proxy.

What language to use for AMI

Selecting a programming language can be hard these days. There are simply too many options—Java, PHP, Perl, C, C#, Python, and several others. It's possible to use AMI with any language that supports a socket or telnet interface. We have chosen PHP for this book because of its popularity.

AMI protocol behavior

- Before sending any commands to Asterisk, you need to establish an AMI session
- The first line of a packet will have the key “Action” when sent from a client
- The first line of a packet will have the key “Response” or “Event” when coming from Asterisk
- Packages can be transmitted in any direction after the authentication

Packet types

The type of the packet is determined by the existence of the following keys:

- Action: A packet sent from a client connected to AMI asking for a specific action. There is a finite set of actions available to clients. The loaded modules determine these actions. A packet contains the action name and its parameters.
- Response: The response sent from Asterisk to the last action sent from the client.
- Event: Data belonging to an event generated in the Asterisk core or by a module.

When a client sends packets of the Action type, a parameter named `ActionID` is included. Since the order in which the responses sent from Asterisk cannot be predicted, `ActionID` is used to correlate actions and responses.

Event packets are used in two different contexts. First, events inform the client about changes in Asterisk (e.g., newly created channels, channels disconnected, or agents logging in and out of a queue). Second, events are used to transport responses to a client action.

Configuring users and permissions

To access AMI, it is necessary to establish a TCP connection listening to a TCP port (usually 5038). You will need to configure the `/etc/asterisk/manager.conf` file to create a user account and permissions.

There is a finite set of permissions: “read,” “write,” or both. These permissions are defined in the `manager.conf` file.

```
[general]
enabled=yes
port=5038
bindaddr=127.0.0.1

[admin]
secret=senha
read=system,call,log,verbose,command,agent,user
write=system,call,log,verbose,command,agent,user
deny=0.0.0.0/0.0.0.0
permit=127.0.0.1/255.255.255.255
```

Logging in to the AMI

To log in to and authenticate AMI, you will need to send an action packet of the login type with a username and account created in the `manager.conf`.

```
Action:login
Username:admin
Secret:password
```

Example: Logging in to AMI using php

```
<?php

$socket = fsockopen("127.0.0.1","5038", $errno, $errstr, $timeout);
fputs($socket, "Action: Login\r\n");
fputs($socket, "UserName: admin\r\n");
fputs($socket, "Secret: senha\r\n\r\n");

?>
```

If you don't need to receive the events, you can use “Events Off”.

```
<?php

$socket = fsockopen("127.0.0.1","5038", $errno, $errstr, $timeout);
fputs($socket, "Action: Login\r\n");
fputs($socket, "UserName: admin\r\n");
fputs($socket, "Secret: senha\r\n\r\n");
fputs($socket, "Events: off\r\n\r\n");

?>
```

Action packets

When you send an action packet to Asterisk, you can provide some extra keys (e.g., called number) by passing **key:value** pairs after the action. It is also possible to pass channel and global variables to the dial plan.

```
Action: <action type><CRLF>
<Key 1>: <Value 1><CRLF>
<Key 2>: <Value 2><CRLF>

Variable: <Variable 1>=<Value 1><CRLF>
Variable: <Variable 2>=<Value 2><CRLF>
...
<CRLF>
```

Action commands

You can use the CLI instruction **manager show commands** to list the available actions. In version 1.6.1 the commands were:

Action	Privilege	Synopsis	
WaitEvent	<none>		Wait for an event to occur
ModuleCheck	system,all		Check if module is loaded
ModuleLoad	system,all		Module management
CoreShowChannel	system,reportin		List currently active channels
Reload	system,config,a		Send a reload event
CoreStatus	system,reportin		Show PBX core status variables
CoreSettings	system,reportin		Show PBX core settings (version etc)
VoicemailUsersL	call,reporting,		List All Voicemail User Information
UserEvent	user,all		Send an arbitrary event
SendText	call,all		Send text message to channel
ListCommands	<none>		List available manager commands
MailboxCount	call,reporting,		Check Mailbox Message Count
MailboxStatus	call,reporting,		Check Mailbox
AbsoluteTimeout	system,call,all		Set Absolute Timeout
ExtensionState	call,reporting,		Check Extension Status
Command	command,all		Execute Asterisk CLI Command
Originate	originate,all		Originate Call
Atxfer	call,all		Attended transfer
Redirect	call,all		Redirect (transfer) a call
ListCategories	config,all		List categories in configuration file
CreateConfig	config,all		Creates an empty file in the configuration directory
UpdateConfig	config,all		Update basic configuration
GetConfigJSON	system,config,a		Retrieve configuration (JSON format)
GetConfig	system,config,a		Retrieve configuration
Getvar	call,reporting,		Gets a Channel Variable
Setvar	call,all		Set Channel Variable
Status	system,call,rep		Lists channel status
Hangup	system,call,all		Hangup Channel
Challenge	<none>		Generate Challenge for MD5 Auth
Login	<none>		Login Manager
Logoff	<none>		Logoff Manager

Events	<none>	Control Event Flow
Ping	<none>	Keepalive command
DAHDIRestart	<none>	Fully Restart DAHDI channels (terminates calls)
DAHDIShowChanne	<none>	Show status DAHDI channels
DAHDIIDNDooff	<none>	Toggle DAHDI channel Do Not Disturb status OFF
DAHDIIDNDon	<none>	Toggle DAHDI channel Do Not Disturb status ON
DAHDI dialOffhook	<none>	Dial over DAHDI channel while offhook
DAHDIHangup	<none>	Hangup DAHDI Channel
DAHDITransfer	<none>	Transfer DAHDI Channel
IAXnetstats	system,reportin	Show IAX Netstats
IAXpeerlist	system,reportin	List IAX Peers
IAXpeers	system,reportin	List IAX Peers
QueueRule	<none>	Queue Rules
QueuePenalty	agent,all	Set the penalty for a queue member
QueueLog	agent,all	Adds custom entry in queue_log
QueuePause	agent,all	Makes a queue member temporarily unavailable
QueueRemove	agent,all	Remove interface from queue.
QueueAdd	agent,all	Add interface to queue.
QueueSummary	<none>	Queue Summary
QueueStatus	<none>	Queue Status
Queues	<none>	Queues
AgentLogoff	agent,all	Sets an agent as no longer logged in
Agents	agent,all	Lists agents and their status
UnpauseMonitor	call,all	Unpause monitoring of a channel
MeetmeList	reporting,all	List participants in a conference
MeetmeUnmute	call,all	Unmute a Meetme user
MeetmeMute	call,all	Mute a Meetme user
PlayDTMF	call,all	Play DTMF signal on a specific channel.
SIPnotify	system,all	Send a SIP notify
SIPshowregistry	system,reportin	Show SIP registrations (text format)
SIPqualifypeer	system,reportin	Show SIP peer (text format)
SIPshowpeer	system,reportin	Show SIP peer (text format)
SIPpeers	system,reportin	List SIP peers (text format)
AGI	agi,all	Add an AGI command to execute by Async AGI
StopMonitor	call,all	Stop monitoring a channel
PauseMonitor	call,all	Pause monitoring of a channel
ChangeMonitor	call,all	Change monitoring filename of a channel
ShowDialPlan	config,reportin	List dialplan
Monitor	call,all	Monitor a channel
DBDelTree	system,all	Delete DB Tree
DBDel	system,all	Delete DB Entry
DBPut	system,all	Put DB Entry
DBGet	system,reportin	Get DB Entry
Bridge	call,all	Bridge two channels already in the PBX
Park	call,all	Park a channel
ParkedCalls	<none>	List parked calls

If you need to know specific command parameters, use the **manager show command <command>**.

Example:

```
asterisk*CLI> manager show command originate
Action: Originate
Synopsis: Originate Call
```

Privilege: call,all

Description: Generates an outgoing call to a Extension/Context/Priority or Application/Data

Variables: (Names marked with * are required)

*Channel: Channel name to call

Exten: Extension to use (requires 'Context' and 'Priority')

Context: Context to use (requires 'Exten' and 'Priority')

Priority: Priority to use (requires 'Exten' and 'Context')

Application: Application to use

Data: Data to use (requires 'Application')

Timeout: How long to wait for call to be answered (in ms)

CallerID: Caller ID to be set on the outgoing channel

Variable: Channel variable to set, multiple Variable: headers are allowed

Account: Account code

Async: Set to 'true' for fast origination

Event packets

Events are generated in the manager interface based on some Asterisk activities. There are link and unlink events.

Link Events:

Link events are triggered when two channels are connected and voice transmission starts. More than one event can be triggered for a single call. Any call that needs transcoding will generate two events: the first one is a fail to establish a native bridge between the channels while the second is the call itself.

Example:

```
Event: Link
Channel1: SIP/4001-AAAA
Channel2: SIP/4000-BBBB
Uniqueid1: 1234567890.12
Uniqueid2: 1234567890.12
```

Unlink events:

Unlink events are triggered when a link between two channels is disconnected just before the call is completed.

Example:

```
Event: Link
Channel1: SIP/4001-AAAA
Channel2: SIP/4000-BBBB
Uniqueid1: 1234567890.12
Uniqueid2: 1234567890.12
```

Events available

Below are some of the events available for Asterisk.

AbstractAgentEvent

HoldEvent

PeerStatusEvent

AbstractParkedCallEvent	JoinEvent	QueueEntryEvent
AbstractQueueMemberEvent	LeaveEvent	QueueEvent
AgentCallbackLoginEvent	LinkageEvent	QueueMemberAddedEvent
AgentCallbackLogoutEvent	LinkEvent	QueueMemberEvent
AgentCalledEvent	LogChannelEvent	QueueMemberPausedEvent
AgentCompleteEvent	ManagerEvent	QueueMemberRemovedEvent
AgentConnectEvent	MeetMeEvent	QueueMemberStatusEvent
AgentDumpEvent	MeetMeJoinEvent	QueueParamsEvent
AgentLoginEvent	MeetMeLeaveEvent	QueueStatusCompleteEvent
AgentLogoutEvent	MeetMeStopTalkingEvent	RegistryEvent
AgentsCompleteEvent	MeetMeTalkingEvent	ReloadEvent
AgentsEvent	MessageWaitingEvent	RenameEvent
AlarmClearEvent	NewCallerIdEvent	ResponseEvent
AlarmEvent	NewChannelEvent	ShutdownEvent
CdrEvent	NewExtenEvent	StatusCompleteEvent
ChannelEvent	NewStateEvent	StatusEvent
ConnectEvent	OriginateEvent	UnholdEvent
DBGetResponseEvent	OriginateFailureEvent	UnlinkEvent
DialEvent	OriginateSuccessEvent	UnparkedCallEvent
DisconnectEvent	ParkedCallEvent	UserEvent
DNDStateEvent	ParkedCallGiveUpEvent	ZapShowChannelsCompleteEvent
ExtensionStatusEvent	ParkedCallsCompleteEvent	ZapShowChannelsEvent
FaxReceivedEvent	ParkedCallTimeoutEvent	
HangupEvent	PeerEntryEvent	
HeldCallEvent	PeerListCompleteEvent	

Asterisk Gateway Interface

AGI is a gateway interface to Asterisk similar to CGI used by web servers. It allows the use of high-level languages like Perl, PHP, and Python to extend Asterisk's functionality. The main application for CGIs is IVR building. There are four types of AGI:

- Normal AGI, which calls a program inside Asterisk's box.
- Fast AGI, which calls an AGI in another server using TCP sockets.
- EAGI, which enables sound channel access and control from the AGI.
- DEADAGI, which gives access to the channel even after `hangup()`. Usually called in the 'h' extension.

Application format:

```
asterisk*CLI> core show application agi
```


| Asterisk Gateway Interface |

```
asterisk*CLI>
  -= Info about application 'AGI' ==
```

[Synopsis]
Executes an AGI compliant application

[Description]
[E|Dead]AGI(command|args): Executes an Asterisk Gateway Interface compliant program on a channel. AGI allows Asterisk to launch external programs written in any language to control a telephony channel, play audio, read DTMF digits, etc. by communicating with the AGI protocol on stdin and stdout.
Returns -1 on hangup (except for DeadAGI) or if application requested hangup, or 0 on non-hangup exit.
Using 'EAGI' provides enhanced AGI, with incoming audio available out of band on file descriptor 3

Use the CLI command 'agi show' to list available agi commands

You can show the available AGI commands using the command **agi show** (1.6.1):

Dead	Command	Description
No	answer	Answer channel
No	channel status	Returns status of the connected channel
Yes	database del	Removes database key/value
Yes	database deltree	Removes database keytree/value
Yes	database get	Gets database value
Yes	database put	Adds/updates database value
Yes	exec	Executes a given Application
No	get data	Prompts for DTMF on a channel
Yes	get full variable	Evaluates a channel expression
No	get option	Stream file, prompt for DTMF, with timeout
Yes	get variable	Gets a channel variable
No	hangup	Hangup the current channel
Yes	noop	Does nothing
No	receive char	Receives one character from channels supporting it
No	receive text	Receives text from channels supporting it
No	record file	Records to a given file
No	say alpha	Says a given character string
No	say digits	Says a given digit string
No	say number	Says a given number
No	say phonetic	Says a given character string with phonetics
No	say date	Says a given date
No	say time	Says a given time
No	say datetime	Says a given time as specified by the format given
No	send image	Sends images to channels supporting it
No	send text	Sends text to channels supporting it
No	set autohangup	Autohangup channel in some time
No	set callerid	Sets callerid for the current channel
No	set context	Sets channel context
No	set extension	Changes channel extension
No	set music	Enable/Disable Music on hold generator
No	set priority	Set channel dialplan priority
Yes	set variable	Sets a channel variable
No	stream file	Sends audio file on channel
No	control stream file	Sends audio file and allows the listener cont.the
stream		
No	tdd mode	Toggles TDD mode (for the deaf)
Yes	verbose	Logs a message to the asterisk verbose log
No	wait for digit	Waits for a digit to be pressed
No	speech create	Creates a speech object

No	speech set	Sets a speech engine setting
Yes	speech destroy	Destroys a speech object
No	speech load grammar	Loads a grammar
Yes	speech unload grammar	Unloads a grammar
No	speech activate grammar	Activates a grammar
No	speech deactivate grammar	Deactivates a grammar
No	speech recognize	Recognizes speech
No	gosub	Execute a dialplan subroutine

To debug, use `agi debug`.

Using AGI

In this example, we will use `php-cli`, the `php` command line version. Install `php-cli` if it's not already installed. Follow these steps to use `php` AGI scripts.

Step 1: All AGI scripts are located in `/var/lib/asterisk/agi-bin`

Step 2: Change the permissions to allow execution.

```
chmod 755 *.php
```

Step 3: Shell interface (`php` specific). The script's first lines have to be:

```
#!/usr/bin/php -q
<?php
```

Step 4: Open I/O channels:

```
$stdin = fopen('php://stdin', 'r');
$stdout = fopen('php://stdout', 'w');
$stdlog = fopen('agi.log', 'w');
```

Step 5: Manage the Asterisk output.

Asterisk sends the information set each time AGI is called.

```
agi_request:teststeph
agi_channel: Zap/1-1
agi_language: en
agi_type: Zap
agi_callerid:
agi_dnid:
agi_context: default
agi_extension: 4000
agi_priority: 1
```

Save the information sent:

```
while (!feof($stdin)) {
    $temp = fgets($stdin);
    $temp = str_replace("\n", "", $temp);
    $s = explode(":", $temp);
    $agivar[$s[0]] = trim($s[1]);
    if (($temp == "") || ($temp == "\n")) {
        break;
    }
}
```

```
}  
}
```

The previous script will create an array named `$agivar`. Available options are:

- `agi_request` – AGI file name
- `agi_channel` – AGI originating channel
- `agi_language` – Language set
- `agi_type` – Channel type (e.g., SIP, ZAP)
- `agi_uniqueid` – Unique identifier
- `agi_callerid` – CallerID (Ex. Flavio <8590>)
- `agi_context` – Originating context
- `agi_extension` – Called extensions
- `agi_priority` – Priority
- `agi_accountcode` – Originating account code

To call a variable named `agi_extensions`, use `$agivar[agi_extensions]`.

Step 6: Use channel AGI

At this point, you can start talking to Asterisk. Use the `fputs` command to send commands to AGI. You can also use the `echo` command.

```
fputs($stdout,"SAY NUMBER 4000 '79#' \n");  
fflush($stdout);
```

Notes about using quotes:

- AGI command options are not optional
- Some options need to be enclosed in quotes <escape digits>
- Some options should not be enclosed in quotes <digit string>
- Some options can use both formats
- You can use single quotes

Step 7 – Pass variables

Channel variables can be set in the AGI, but cannot be used inside the AGI. The following example does not work inside an AGI.

```
SET VARIABLE MY_DIALCOMMAND "SIP/${EXTEN}"
```

The following example does work:

```
SET VARIABLE MY_DIALCOMMAND "SIP/4000"
```

Step 8: Asterisk responses

The following is necessary to verify responses from Asterisk:

```
$msg = fgets($stdin,1024);  
fputs($stderr,$msg . "\n");
```

Step 9: Kill the locked (zombie) processes

If your script fails for some reason, the process will hang. Use the killproc command to clean it before testing again.

```
#!/usr/bin/php4 -q
<?php
ob_implicit_flush(true);
set_time_limit(6);
$in = fopen("php://stdin","r");
$stdlog = fopen("/var/log/asterisk/agi.log", "w");

// Enable debug (more verbose)
$debug = false;

// Functions definition
function read() {
    global $in, $debug, $stdlog;
    $input = str_replace("\n", "", fgets($in, 4096));
    if ($debug) fputs($stdlog, "read: $input\n");
    return $input;
}

function errlog($line) {
    global $err;
    echo "VERBOSE \"\$line\"\n";
}

function write($line) {
    global $debug, $stdlog;
    if ($debug) fputs($stdlog, "write: $line\n");
    echo $line.\n";
}

// Put agi headers in the array
while ($env=read()) {
    $s = split(":", $env);
    $agi[str_replace("agi_", "", $s[[0])] = trim($s[[1]]);
    if (($env == "") || ($env == "\n")) {
        break;
    }
}

// main program
echo "VERBOSE \"start here!\" 2\n";
read();
errlog("Call from ".$agi['channel']." - Phone ringing ");
read();
write("SAY DIGITS 22 X"); // X is the escape digit. since X is not DTMF, no ex
it is possible
read();
write("SAY NUMBER 2233 X"); // X is the escape digit. since X is not DTMF, no
exit is possible
read();

// clean up file handlers etc.
fclose($in);
fclose($stdlog);

exit;
?>
```

DeadAGI

DeadAGI is used when you do not have a live channel. Usually you execute the DeadAGI in the `h` extension.

FASTAGI

Fast AGI implements AGI using a TCP port (4573 by default) as the Input/Output channel. FastAGI format is `(agi://)`. For example:

```
exten => 0800400001, 1, Agi(agi://192.168.0.1)
```

When the TCP connection is lost or disconnected, the AGI ends and the TCP connection is closed, followed by a call disconnection. This resource is useful to ease the CPU load from your Asterisk server running scripts in an external server. You may obtain more details about FastAGI in the source code directory (please see the file “`agi/fastagi-test`”).

OrderlyCalls has a Java AGI server that implements Fast AGI for Java. For more information, see <http://www.orderlycalls.com>

Changing the source code

Asterisk is developed in C language (not C++). Teaching C programming is beyond the scope of this document. If you are interested, you will find related documentation at www.asterisk.org/developers, which offers good tips on how to apply and create patches to Asterisk as well API documentation, mostly generated by Doxygen software, <http://www.asterisk.org/doxygen/>

For those familiar with C programming, changing the applications source code can be the most powerful (and dangerous) way to extend Asterisk.

Summary

In this chapter, you have learned how to interface external programs to the Asterisk PBX. We have started with `asterisk -rx` passing commands from the Linux shell to the Asterisk console. Next, we learned about the `System()` application, which allows calling an external program from the dial plan. AMI is the closest interface to a CTI interface common in traditional PBXs. To call an application from the dial plan, we used the AGI, with a taste for its different flavors: DeadAGI for dead channels, EAGI for handling the audio streaming, Fast AGI for using TCP sockets as the input/output interface, and normal AGI for calling and processing the scripts inside the same Asterisk box.

Quiz

1. Which of the following is not an interfacing method for Asterisk?

- A. AMI

- B. AGI
 - C. Asterisk -rx
 - D. System()
 - E. External()
2. AMI allows for passing Asterisk commands via TCP sockets. This resource is enabled by default.
- A. True
 - B. False
3. AMI is very safe, because its authentication is done using MD5 challenge/response.
- A. True
 - B. False
4. FastAGI allows the calling of external scripts from the dial plan to an external machine using TCP sockets (usually 4573).
- A. True
 - B. False
5. DeadAGI is used in active channels. It can be used in ZAP channels, but not in SIP or IAX channels.
- A. True
 - B. False
6. AGI supports only PHP as a scripting language
- A. True
 - B. False
7. The command _____ shows all available AGI commands.
8. The command _____ shows all available AMI commands.
9. To debug an AGI, you should use the command _____.

14

Asterisk Real-Time

As you know, the Asterisk configuration is achieved through the use of several text files in the `/etc/asterisk` directory. Despite the ease of using text files, there are some known drawbacks:

- The need to reload Asterisk each time the files are changed
- Increased memory usage for a large volume of users
- Difficulty coding a provisioning interface using text files
- No possibility of integration to existing databases

ARA or Asterisk Realtime, as it is known, was created by Anthony Minessale II, Mark Spencer, and Constantine Filin and was designed to allow transparent integration with SQL databases. An LDAP interface is available too. This system is also known as Asterisk External Configuration and is configured in `/etc/asterisk/extconfig.conf`. You can map configuration files to tables in a database (static configuration) and real-time entries for the dynamic creation of objects without the need to reload Asterisk.

Objectives

By the end of this chapter, the reader should be able to:

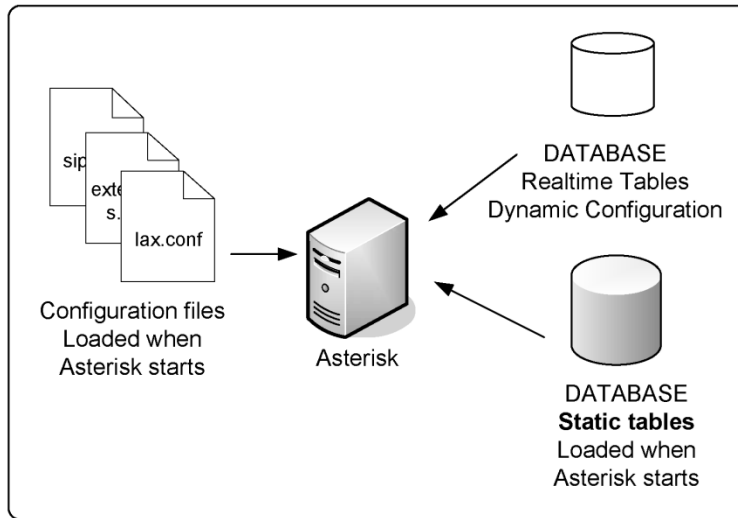
- Understand advantages and limitations of Asterisk Real Time.
- Install MySQL for use with ARA
- Compile and install ARA using MySQL
- Test the system in a lab environment

How does Asterisk Real Time work?

In the new Real Time architecture, all database-specific code was moved to channel drivers. The channel only calls a generic routine that searches the database. The result is a much simpler and cleaner process from the source code point of view. The database is accessed by three functions:

- **STATIC**: Used to set up a static configuration when a module is loaded.
- **REALTIME**: Used to search objects during a call or another event.

- UPDATE: Used to update objects.



The channel database support was not changed. There are peers and users called static (normal) and peers and users called real time (database). For the static, it doesn't matter if it is loaded from a configuration file or from the database kept in the memory. However, the real-time peers/users are loaded only when a call is made. After the call, the peer or user is deleted. Consequently, there is no support for NAT or message waiting indicator (MWI). You can enable real-time caching using the command `rtcachefriends=yes` in the `sip.conf` file or from the static database. By doing so, you will have NAT traversal and MWI; however, if you do any updates to this peer/user, you will have to reload.

Lab: Installing Asterisk Real/Time

For this lab, we will assume that you still have the MySQL libraries that were installed in Chapter 13.

Step 1: Download the add-ons package.

Please check the current version. At the time of writing, I was downloading 1.6.1.1; it certainly won't be the same when you download it.

```
wget http://downloads.asterisk.org/pub/telephony/asterisk/releases/asterisk-addons-1.6.2.x.tar.gz
```

Step 2: Uncompress the file

```
tar -xzvf asterisk-addons-1.6.2.x.tar.gz
cd asterisk-addons-1.6.2.x
make
```



```
make install  
make samples
```

Confirm the module installation using `module show`.

Configuring Asterisk Real Time

ARA is configured in the `extconfig.conf` text file, where two sections can be easily seen. The first one is the static configuration files section, where you can substitute the text configuration files for database tables. The second section is the `realtime` configuration engine, where you configure database tables for dynamic objects (peers/users). It is not unusual to use text files for the static configuration and the database for dynamic entries. In this case, the first section is untouched.

```
res_mysql.conf  
  
[settings]  
; Static configuration files:  
;  
; file.conf => driver,database[,table]  
; queues.conf => odbc,asterisk,ast_config  
  
; Realtime configuration engine  
;  
iaxusers => mysql,asteriskdb,iax_buddies  
iaxpeers => mysql,asteriskdb,iax_buddies  
sipusers => mysql,asteriskdb,sip_buddies  
sippeers => mysql,asteriskdb,sip_buddies  
voicemail => mysql,asteriskdb,voicemail  
extensions => mysql,asteriskdb,extensions_table
```

`extconfig.conf` file format:

```
;  
; Static and realtime external configuration  
; engine configuration  
;  
; Please read doc/README.extconfig for basic table  
; formatting information.  
;  
[settings]  
;  
; Static configuration files:  
;  
; file.conf => driver,database[,table]  
;  
; maps a particular configuration file to the given  
; database driver, database and table (or uses the  
; name of the file as the table if not specified)
```

```

;
;uncomment to load queues.conf via the odbc engine.
;
;queues.conf => odbc,asterisk,ast_config
;
; The following files CANNOT be loaded from Realtime storage:
;     asterisk.conf
;     extconfig.conf (this file)
;     logger.conf
;
; Additionally, the following files cannot be loaded from
; Realtime storage unless the storage driver is loaded
; early using 'preload' statements in modules.conf:
;     manager.conf
;     cdr.conf
;     rtp.conf
;
; Realtime configuration engine
;
; maps a particular family of realtime
; configuration to a given database driver,
; database and table (or uses the name of
; the family if the table is not specified
;
;example => odbc,asterisk,alhtable
;iaxusers => odbc,asterisk
;iaxpeers => odbc,asterisk
;sipusers => odbc,asterisk
;sippeers => odbc,asterisk
;voicemail => odbc,asterisk
;extensions => odbc,asterisk
;queues => odbc,asterisk
;queue_members => odbc,asterisk

```

Static configuration section

The static configuration section is where you store the equivalent to configuration files in the database. These configurations are read during the Asterisk load. Some modules reread the database when you reload. Examples of the static configuration are:

```

<conf filename> => <driver>,<databasename>[,table_name]
queues.conf => mysql,asteriskdb,queues_conf
sip.conf => odbc,asteriskdb,sip_conf
iax.conf => ldap,MyBaseDN,iax

```

Three examples are described above. In the first one, you bind **queues.conf** to a table **queues** in the **asteriskdb** database. In the second example, you bind **sip.conf** to the table **sip_conf** in the database **asteriskdb** defined in the odbc configuration. In the last example, you bind **iax.conf** to an LDAP directory. **MyBaseDN** is the base DN to be searched.

In the previous example, the application `app_queue.so` is loaded while MySQL driver queries the database and gets the required information.

Real Time configuration section

The real-time configuration (second part of the `extconfig.conf` file) is where the configuration piece to be loaded is configured, updated, and unloaded in real time. With real time, it is not necessary to reload the configurations. The real-time syntax follows:

```
<family name> => <driver>,<database name>[,table_name]
```

Example:

```
sippeers => mysql,asteriskdb,sip_peers
sipusers => mysql,asteriskdb,sip_users
queues => mysql,asteriskdb,queue_table
queue_members => mysql,asteriskdb,queue_member_table
voicemail => mysql,asteriskdb,test
```

Here we have five configuration lines. In the first line, you bind the family `sippeers` to a table `sip_peers` in the `asteriskdb` MySQL database. In the last, you bind the voicemail family to the test table in the `asteriskdb` database. Note that `sip_peers` and `sip_users` could point to the same table.

Database configuration

Now that we have configured the `extconfig.conf` file, let's create the tables. Generally speaking, the database columns need to have the same fields as the configuration files. For example, for a SIP or an IAX object, such as the one described below,

```
[4000]
host=dynamic
secret=senha
context=default
context=ramais
```

The database table should look like this:

name	Host	secret	context	ipaddr	port	regseconds
4000	dynamic	senha	default;ramais	10.1.1.1	4569	1765432

To use this with IAX, the tables need to have at least the following fields: `name`, `port`, and `regseconds`. You may configure other columns to the desired fields. For example, if you want the parameter `callerid`, create a column named `callerid` (the same parameter as the text file). A SIP table may look like the one below:

name	host	secret	context	ipaddr	port	regseconds	username
4000	dynamic	senha	default	10.1.1.1	5060	1765432	4000

A voicemail table should look like this:

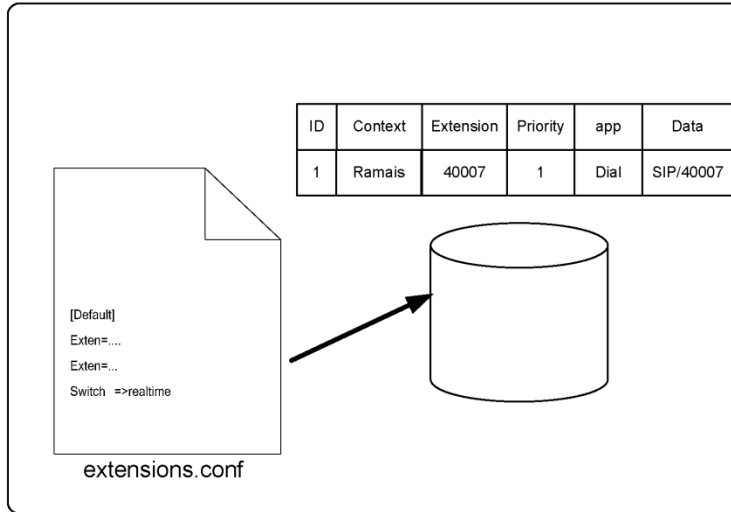
Uniqueid	mailbox	Context	password	email	fullname
----------	---------	---------	----------	-------	----------

1	4000	Default	4000	joao@silva.com	Joao Silva
---	------	---------	------	----------------	------------

The **uniqueid** should be unique to each voicemail user and can be **autoincrement**. It need not have any relationship to the mailbox or context.

Building a dial plan using Asterisk Real Time

You can also use the real-time system to create the dial plan. ARA uses the statement **switch** to include the real-time extensions into the normal dial plan contained in the **extensions.conf** file.



The extension table should look like the one below:

context	Exten	priority	app	appdata
Ramais	4000	1	dial	SIP/4000&IAX2/4000

In the dial plan, you have to use the switch command to use the real time.

```
[local]
switch => realtime
```

or

```
[local]
switch =>realtime/ramais@extensions
```

Lab: Installing and creating the database tables

In this lab, we will prepare the database to receive Asterisk parameters. We will prepare just the REALTIME databases. The static configuration will be left to the configuration text files (Cool isn't it?).

Table creation in MySQL:

Step 1: Get into to the MySQL database using root

```
mysql -u root -p
```

Step 2: Create a database for Asterisk Real Time

```
mysql>create database asteriskdb;
```

Step 3: Create a user with access to the asteriskdb database

```
mysql>grant all privileges on 'asteriskdb'.* to 'asterisk'@'localhost'  
identified by 'asterisk';
```

Step 4: Exit MySQL and log in again using the user created in step 3

```
mysql -u asterisk -p asteriskdb
```

When asked for the password, type **asterisk**.

Step 5: Create the necessary tables

Download the following file from www.asteriskguide.com:

```
wget www.asteriskguide.com/pdf/realtime.sql
```

Execute the following commands:

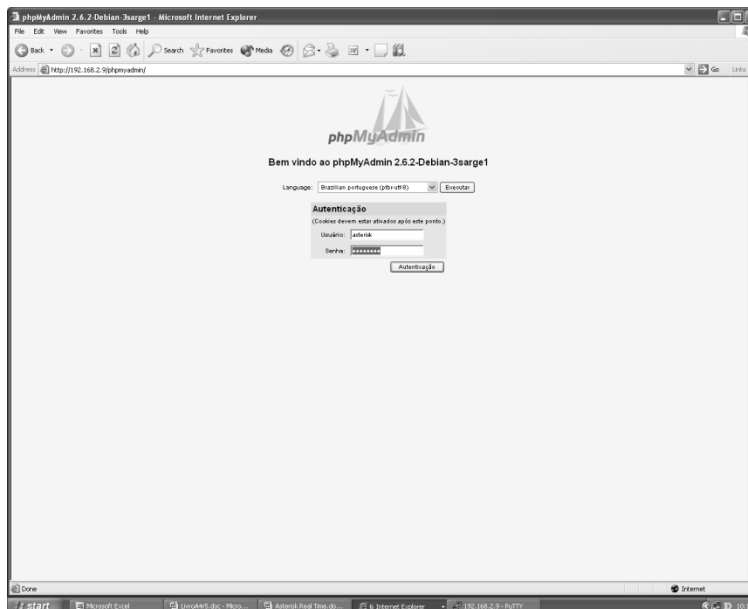
```
mysql asteriskdb -u asterisk -p <realtime.sql
```

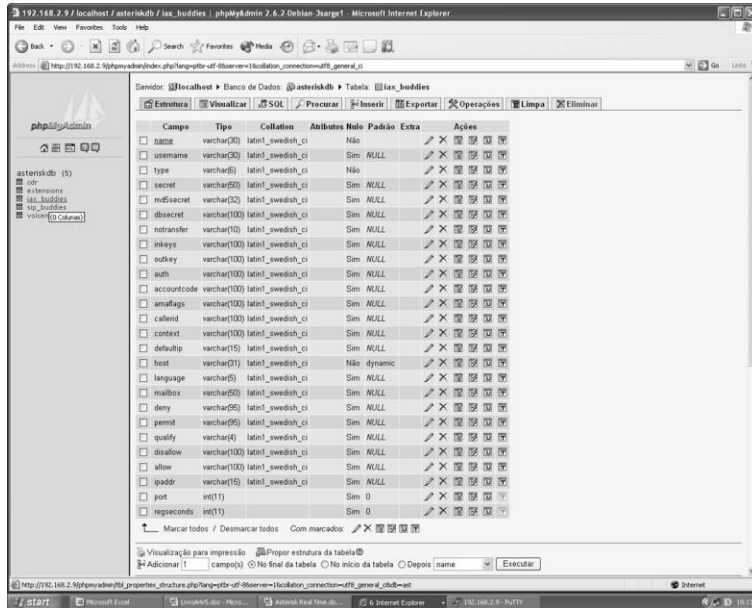
Use **asterisk** as the password.

Step 6: Install phpmyadmin to handle database tasks

```
#apt-get install phpmyadmin
```

Below are two screenshots of the utility log in screen and the table screen.





Step 7: Configure Asterisk to access the database. In the `res_mysql.conf`

```
[general]
dbhost = 127.0.0.1
dbname = asteriskdb
dbuser = asterisk
dbpass = asterisk
dbport = 3306
```

Lab: Configuring and testing ARA

In this lab we will change the `extconfig.conf` configuration to reflect our database configuration and tables.

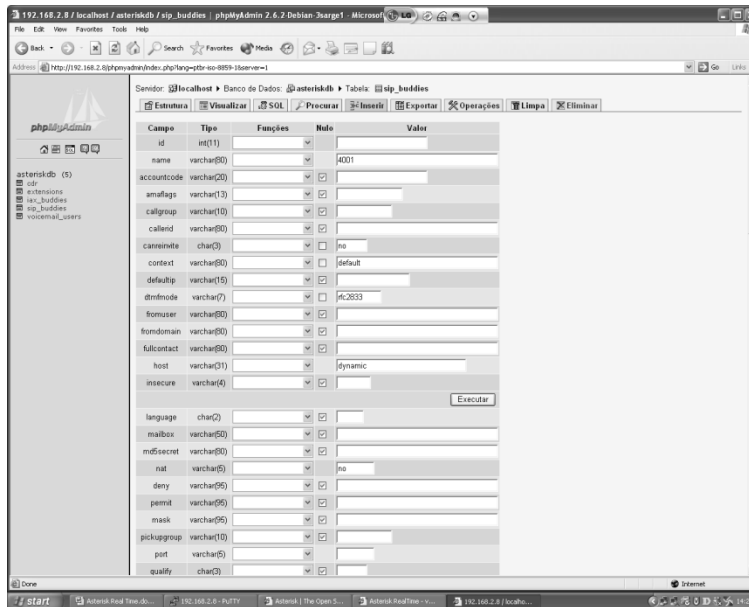
Step 1: Configure `extconfig.conf` and reload Asterisk

```
; Realtime configuration engine
;
; maps a particular family of realtime
; configuration to a given database driver,
; database and table (or uses the name of
; the family if the table is not specified
;
;example => odbc,asterisk,alltable
iaxusers => mysql,asteriskdb,iax_buddies
iaxpeers => mysql,asteriskdb,iax_buddies
sipusers => mysql,asteriskdb,sip_buddies
sippeers => mysql,asteriskdb,sip_buddies
```

```
voicemail => mysql,asteriskdb,voicemail_users  
extensions => mysql,asteriskdb,extensions_table
```

Step 2: Real Time extension test

Create a new 4001 SIP friend inserting a record on the sip_buddies table and try to authenticate this peer/user with a softphone.



Step 3: Try a call from the 4000 extension created before (static) and the new 4001

Verify that using SIP shows peers in the SIP objects. You will note that only the static peer is shown. This behavior is normal since the peer is only created when you call. If you need to have NAT traversal support or MWI, use `rtcachefriends=yes` in the `sip.conf` file.

Step 4: Put the command `rtcachefriends=yes` in the `[general]` section of the `sip.conf` file

Step 5: Once again, try a call from 4000 to 4001

Verify using `show peers`. Why does it appear now?

Step 6: Create a new SIP peer in the database with the name 4007

Change the phone registration to 4007 without reloading Asterisk

Step 7: Include the extensions in the database

```
mysql -u asterisk -p
```

Enter password:

--> Use `asterisk` when asked.

Use `phpadmin` to include an extension in the database. If you prefer, use the following commands instead in the MySQL client interface.

```
use asteriskdb;
insert into extensions_table(id, context, exten, priority, app, appdata) VALUES
('1', 'teste', '40007', '1', 'Dial', 'SIP/40007');
```

Step 8: Include Asterisk Real Time in the dial plan. In the context default:

```
switch => realtime/teste@extensions
```

Reload the extensions to activate the change.

```
asterisk-server*CLI>extensions reload
```

Step 9: Reconfigure one of the phones to 40007, if you have not already done so.

Step 10: Dial 40007 from an existent phone

Summary

In this chapter, you have learned that Asterisk Real Time allows you to put your configurations into a database. Databases supported are MySQL and any other unix ODBC-supported databases. The configuration is divided into static and real time. Static configuration replaces the configuration files, while the real-time configuration creates dynamic objects that are loaded only when a call or other related event happens. We concluded with a practical lab on how to install and configure ARA.

Quiz

1. Asterisk real-time is part of the standard Asterisk distribution.
 - A. True
 - B. False
2. To compile ARA and use it with MySQL databases, the following libraries have to be installed.
 - A. Libmysqlclient12-dev
 - B. Mysql-server-4.1
 - C. Perl
 - D. PHP
3. Configuration of a database server's IP addresses and ports is done in the following file:
 - A. extensions.conf
 - B. sip.conf
 - C. res_mysql.conf
 - D. extconfig.conf

4. The file `extconfig.conf` is used to configure the tables used by real time. This file has two distinct sections (check two):
- A. Static configuration
 - B. Real-time configuration
 - C. Outbound routes
 - D. IP addresses and database ports
5. In the static configuration, once you load the objects from the database, they are loaded dynamically into Asterisk's memory whenever necessary.
- A. True
 - B. False
6. When a SIP channel is configured in real time, it's not possible to use resources such as "qualify" or MWI (message waiting indicator) because the channel does not exist until a call is made. This causes the following problems:
- A. This channel can call, but not receive calls
 - B. The SIP channel could not be used behind NAT because qualify is used to keep NAT translation open.
 - C. It's not possible to make MWI work in the phones that support it.
 - D. It's not possible to use the channel since SIP is always static.
7. If you want to use real-time configuration with SIP channels, but need to support NAT and MWI, you should use:
- A. Real Time was not created for use with NAT
 - B. `rtcachefriends=yes` in `sip.conf`
 - C. Only MWI is possible
 - D. To use NAT, the configuration needs to be static
8. You can still use text configuration files even after installing ARA.
- A. True
 - B. False
9. Phpadmin is mandatory when you use Real Time.
- A. True
 - B. False
10. The database has to be created with all the existing fields of the configuration file.
- A. True
 - B. False

15

Building a simple PBX using AsteriskNOW

It is becoming quite popular to configure Asterisk using a ready-to-run distribution. In this chapter, we will show you how to install AsteriskNow™ and build a simple PBX using its graphical user interface called freePBX™. We will execute the same tasks from chapter 3, but with a simpler interface. Although it is much easier, it is not as flexible. As a general rule, I use ready-to-run distributions when I want to install a PBX with limited features. To implement complex telephony applications, Asterisk classic is still the best choice.

Objectives

By the end of this chapter, you should be able to:

- Install Asterisk and freePBX from the AsteriskNOW CD
- Describe the main components of the freePBX
- Build a simple IP PBX through the following tasks
 - Install phones and softphones based on the SIP and IAX protocols
 - Install and configure a SIP trunk with a VoIP operator
 - Install and configure an analog card to connect to the PSTN
 - Configure a simple dial plan
 - Dial between extensions and external destinations
 - Configure voicemail
 - Configure the reception of the calls
 - Configure an auto-attendant
 - Configure an audio conference room

AsteriskNOW

AsteriskNOW is an Asterisk distribution that is easy to install. It is provided by Digium and is based on the CentOS distribution of Linux. The latest version is 1.5. Below are some technical data about it.

- Web Site: <http://www.asterisknow.org>
- OS: CentOS 5.3

- Asterisk version 1.4
- GUI: FreePBX or Asterisk GUI
- Language: English
- Strong points: The new AsteriskNOW assumes the freePBX is the standard user interface, which seems positive to me. FreePBX is becoming a de facto standard for Asterisk GUIs.

We've chosen AsteriskNOW for the lab in this chapter because it is supported and developed by Digium. Actually, the most important feature here is the freePBX, present on most of the Asterisk distributions. Thus, if you prefer, you can change the Asterisk distribution without changing the user interface.

Introduction to freePBX

FreePBX (www.freepbx.org) is an easy-to-use interface to help administrators configure, manage, and monitor the Asterisk server. The last time I checked, freePBX had more than three million downloads and is used with the majority of distributions, such as Elastix and Trixbox.

It is important to understand that freePBX significantly changes the way in which you configure Asterisk. Once you have started using freePBX, you will not be able to freely edit the Asterisk files. Fortunately, there are custom files that allow edits without changing the freePBX settings.

AsteriskNOW installation

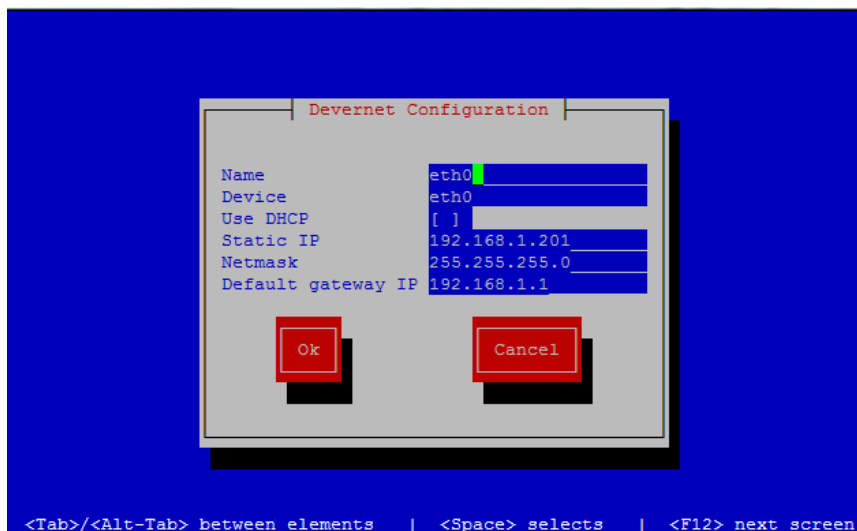
To install AsteriskNOW and freePBX, let's use the installation CD, which will take fewer than 20 minutes. Download the ISO and burn a CD to start the installation. The ISO can be downloaded from: http://dl.digium.com/load_balance.php?q=AsteriskNOW-1.5.0-i386-1of1.iso

Post-installation procedures

Before proceeding, we need to set up the IP address of your server. The next step is to detect and install telephony cards.

IP Address configuration

The first time you log in to the system, you will see a configuration menu. From this menu, you can select "Network Configuration". If you skipped the menu screen, use `system-config-network`. Enter the IP address, mask, and default gateway. To enable DNS, edit the file `/etc/resolv.conf` to include a DNS server to resolve names to IP.



Telephony card configuration

Telephony drivers such as DAHDI come pre-compiled, but you will need to configure the devices to avoid conflicts with other cards. Follow these steps to install the telephony cards:

Step 1: Create the file `genconf_parameters` and specify an echo cancellation algorithm

```
echocan=mg2
```

Step 2: Edit the file `/etc/dahdi/modules` and select only the modules that match your cards. Disable all others by inserting a “#” character in front of the line.

Step 3: Include DAHDI in the server initialization.

```
#chkconfig --add dahdi
#chkconfig dahdi on
#reboot
```

Step 4: Generate the configuration for the telephony card.

```
#dahdi_genconf
```

After generating the file, include the file `/etc/asterisk/dahdi-channels.conf` in the file `/etc/asterisk/chan_dahdi.conf`.

```
#vi chan_dahdi.conf
```

In the last line of the file, include the following command:

```
#include dahdi-channels.conf
```

Step 5: Stop and restart Asterisk and freePBX

```
#amportal stop
#amportal start
```

Step 6: Get into the Asterisk console and check that the DAHDI channels were configured. To get into the console use:

#asterisk -r

On the Asterisk console, use the following command:

```
cli>dahdi show channels
```

Installing extensions on freePBX

Before starting to create extensions, it is important to edit some general configurations of the files `iax_general_custom.conf` and `sip_general_conf_custom`. The files `iax.conf` and `sip.conf` should not be changed directly if you are using freePBX.

Edit the file `sip_general_custom.conf`

```
bindaddr=ip_do_seu_servidor
binport = 5060
allowguest=no
allowauthreject=yes
disallow=all
allow=ulaw
allow=gsm
context=dummy
```

Edit the file `iax_general_custom.conf`

```
context=dummy
bindaddr=ip_do_seu_servidor
binport = 5060
```

Restart the Asterisk server using `amportal restart`.

It is very simple to create freePBX extensions. Use an internet browser to access freePBX at:

http://ip_of_your_server/admin

The user and password are `admin:admin`.

Step 1: Create a SIP extension in the Extensions menu.

Add an Extension

Please select your Device below then click Submit

Device

Device

Choose the extension type, in this case Generic SIP Device. Some other options are Generic IAX Device and Generic Dahdi Device and select the Submit button. Fill the form with extension, name and secret.

Add SIP Extension

Add Extension

User Extension	<input type="text" value="6000"/>
Display Name	<input type="text" value="6000"/>
CID Num Alias	<input type="text"/>
SIP Alias	<input type="text"/>

Extension Options

Outbound CID	<input type="text"/>
Ring Time	Default ▾
Call Waiting	Enable ▾
Call Screening	Disable ▾
Emergency CID	<input type="text"/>

Assigned DID/CID

DID Description	<input type="text"/>
Add Inbound DID	<input type="text"/>
Add Inbound CID	<input type="text"/>

Device Options

This device uses sip technology.

secret	<input type="text" value="supersecret"/>
dtmfmode	<input type="text" value="rfc2833"/>

The screen above seems very complex at first. Let's complete only enough fields to set up an extension.

User Extension: 6000

Display Name: 6000

Leave the fields below empty and fill in only the Extension Options.

Secret: supersecret

Fill the fields related to the voicemail.

Voicemail & Directory

Status	Enabled ▾
Voicemail Password	1234
Email Address	flavio@asteriskguide.com
Pager Email Address	
Email Attachment	<input type="radio"/> yes <input type="radio"/> no
Play CID	<input type="radio"/> yes <input type="radio"/> no
Play Envelope	<input type="radio"/> yes <input type="radio"/> no
Delete Voicemail	<input type="radio"/> yes <input type="radio"/> no
VM Options	
VM Context	default

Status: Enabled

Voicemail Password: supersecret

Email Address:email@email.com

Attach: yes

Click submit to save the record.

Step 2: To create an IAX extension, access the extensions menu.

Add an Extension

Please select your Device below then click Submit

Device

Device

Choose the type of extension—in our case, Generic IAX Device. Click the Submit button. Fill in the fields for the extension, device name, and secret.

Add Extension

User Extension	<input type="text" value="6003"/>
Display Name	<input type="text" value="6003"/>
CID Num Alias	<input type="text"/>
SIP Alias	<input type="text"/>

Extension Options

Outbound CID	<input type="text"/>
Ring Time	Default <input type="button" value="v"/>
Call Waiting	Enable <input type="button" value="v"/>
Call Screening	Disable <input type="button" value="v"/>
Emergency CID	<input type="text"/>

Assigned DID/CID

DID Description	<input type="text"/>
Add Inbound DID	<input type="text"/>
Add Inbound CID	<input type="text"/>

Device Options

This device uses iax2 technology.

secret	<input type="text" value="supersecret"/>
--------	--

Fill in the parameters related to the IAX extension.

Extension: 6003

Display Name: 6003

Secret: supersecret

Fill in the voicemail parameters as well.

Voicemail & Directory

Status	Enabled ▾
Voicemail Password	1234
Email Address	flavio@asteriskguide.com
Pager Email Address	
Email Attachment	<input type="radio"/> yes <input type="radio"/> no
Play CID	<input type="radio"/> yes <input type="radio"/> no
Play Envelope	<input type="radio"/> yes <input type="radio"/> no
Delete Voicemail	<input type="radio"/> yes <input type="radio"/> no
VM Options	
VM Context	default

Status: Enabled

Voicemail Password: supersecret

Email Address: email@email.com

Attach: yes

Click submit

Step 3: Log in with the extension 6000 using the X-Lite and 6003 using Zoiper. Test the calls between the extensions. After testing the phones, turn off 6003 to test the voicemail.

FreePBX codes

FreePBX comes with a series of predefined codes to access the server.

Feature Code Admin

		Use Default?	Feature Status
Core			
Asterisk General Call Pickup	*8	<input checked="" type="checkbox"/>	Enabled
ChanSpy	555	<input checked="" type="checkbox"/>	Enabled
Dial System FAX	666	<input checked="" type="checkbox"/>	Enabled
Directed Call Pickup	**	<input checked="" type="checkbox"/>	Enabled
In-Call Asterisk Attended Transfer	*2	<input checked="" type="checkbox"/>	Enabled
In-Call Asterisk Blind Transfer	##	<input checked="" type="checkbox"/>	Enabled
In-Call Asterisk Disconnect Code	**	<input checked="" type="checkbox"/>	Enabled
In-Call Asterisk Toggle Call Recording	*1	<input checked="" type="checkbox"/>	Enabled
Simulate Incoming Call	7777	<input checked="" type="checkbox"/>	Enabled
User Logoff	*12	<input checked="" type="checkbox"/>	Enabled
User Logon	*11	<input checked="" type="checkbox"/>	Enabled
ZapBarge	888	<input checked="" type="checkbox"/>	Enabled
Info Services			
Call Trace	*69	<input checked="" type="checkbox"/>	Enabled
Directory	#	<input checked="" type="checkbox"/>	Enabled
Echo Test	*43	<input checked="" type="checkbox"/>	Enabled
Speak Your Exten Number	*65	<input checked="" type="checkbox"/>	Enabled
Speaking Clock	*60	<input checked="" type="checkbox"/>	Enabled

On the Feature Codes, page you will see codes to several PBX operations. Use *98 and *97 to access the voicemail.

Dialing external numbers

To dial external numbers, you need to create trunks and outbound routes. We are going to show you how to create these items. Below we will give you two examples; the first is how to create an analog

trunk using an X100P clone card while the second is how to create a SIP trunk, if you are using a VoIP provider.

Creating a trunk using an FXO interface

In the chapter on analog lines, you learned how to configure analog cards. After configuring the card correctly, you can use the channel DAHDI/g1 (or your specific group). Access the trunks option and select:

- ➕ Add Zap Trunk (DAHDI compatibility mode).

Add ZAP Trunk (DAHDI compatibility mode)

General Settings

Outbound Caller ID:

Never Override CallerID:

Maximum Channels:

Disable Trunk: Disable

Monitor Trunk Failures: Enable

Outgoing Dial Rules

Dial Rules:

Dial Rules Wizards:

Outbound Dial Prefix:

Outgoing Settings


Zap Identifier (trunk name):

Add the fields:

Outbound Caller ID: “your phone number”

Zap Identifier: In our case, DAHDI/g1. You should change the DAHDI group (e.g., g0, g1) accordingly.

Creating a SIP trunk to a VoIP Provider

To create a SIP trunk, you need to open an account in a SIP provider. This material does not have the intention to be biased toward a specific provider. Check with your VoIP operator for the correct parameters before starting. Choose the option Trunks and select  Add a SIP trunk.

Add SIP Trunk

General Settings

Outbound Caller ID:

Never Override CallerID:

Maximum Channels:

Disable Trunk: Disable

Monitor Trunk Failures: Enable

Outgoing Dial Rules

Dial Rules:

Dial Rules Wizards:

Outbound Dial Prefix:

Outgoing Settings

Trunk Name:

PEER Details:

```

host=provider_ip
username=user
secret=password
fromuser=user
fromdomain=provider
insecure=invite
dtmfmode=rfc2833
context=from-trunk
disallow=all
allow=ulaw

```

To configure the SIP trunk, fill in the following data:

```

host=provider
username=user

```

```
secret=secret
fromuser=user
fromdomain=provider
insecure=invite
dtmfmode=rfc2833
context=from-trunk
disallow=all
allow=ulaw
type=peer
```

Creating an outbound route

After creating the trunks, you will need to create an outbound route to these trunks. In other words, you will identify which numbers will be forwarded to these destinations. Choose the Outbound routes option and select Add Route.

Add Route

Route Name:	<input type="text" value="ZAP"/>
Route Password:	<input type="text"/>
Emergency Dialing:	<input type="checkbox"/>
Intra Company Route:	<input type="checkbox"/>
Music On Hold?	<input type="text" value="default"/>
Dial Patterns	<div style="border: 1px solid #ccc; padding: 5px; min-height: 100px;">9 .</div> <div style="text-align: right; font-size: small; margin-top: 5px;">Clean & Remove duplicates</div>
Dial patterns wizards:	<input type="text" value="(pick one)"/>
Trunk Sequence	<input type="text" value="ZAP/g0"/>
<input type="button" value="Submit Changes"/>	

On this screen, you will fill in the name of the route, the dialing rules (see onscreen help), and the sequence of trunks to be used. In our case, we used 9|.—in other words, any number prefixed by 9 with any number of digits will be sent to the trunk specified, with the first digit stripped.

Receiving calls

Our PBX is almost ready. Now it is time to receive calls from the trunk to an extension. The creation of the inbound route is not difficult. You need to change the SIP trunk and add a few lines in the details of the peer, and then you will be able to create the inbound route.

General Settings

Outbound Caller ID:

Never Override CallerID:

Maximum Channels:

Disable Trunk: Disable

Monitor Trunk Failures: Enable

Outgoing Dial Rules

Dial Rules:

Dial Rules Wizards:

Outbound Dial Prefix:

Outgoing Settings

Trunk Name:

PEER Details:

```
host=***provider ip address***
username=***userid***
secret=***password***
type=peer
fromuser=user
fromdomain=domain|
context=from-pstn
insecure=invite
```

For a provider, you shouldn't insert anything in the incoming settings, just below the outgoing settings because the SIP authentication sequence searches for an entry with the name in the SIP FROM header field. Checking this parameter, you will see that this field is filled in with the call identifier from the public network. In other words, you cannot create a user with each number coming from the public network. If the Asterisk server does not find a user with a name matching the content of the SIP FROM header, it searches for a peer who has the `host` matching the incoming IP address. If it finds a peer with the matching address, it will use this peer as the entry point, so you should add `context=from-pstn` to indicate that all calls coming from the provider will be handled in the `from-pstn` context in the file `extensions.conf`. Furthermore, you should insert the `insecure=invite` to receive the calls. This command will instruct Asterisk to not send the md5 challenge to the provider—in other words, Asterisk will not ask for a username and password because you have a name and

password provided by your provider. However, your provider does not have a user and password provided by you configured on its servers. If you are behind a NAT device, please include `nat=yes` and `qualify=yes`.

Creating an inbound route

Before receiving calls, your trunk needs to be registered. The registration line is filled with `username:password@voipproviderIP/DID`.

Registration

Register String:

After preparing the SIP trunk to receive incoming calls, you should add a parameter that was passed during the registration of the trunk (`user:password@provider/did`) in the DID number, which will enable the system to forward this call to the final destination—in our case, extension 6000.

Add Incoming Route

Add Incoming Route

Description:

DID Number:

Caller ID Number:

CID Priority Route:

Options

Alert Info:

CID name prefix:

Music On Hold:

Signal RINGING:

Pause Before Answer:

Privacy

Privacy Manager:

Fax Handling

Fax Extension:

Fax Email:

Fax Detection Type:

Pause After Answer:

Set Destination

Terminate Call:

Extensions:

Voicemail:

Using a digital receptionist

You can enhance your productivity by replacing manual repetitive work required to transfer calls to the right destinations with an interactive voice response (IVR) system. Before we can configure the auto-attendant, some planning is required. Let's suppose that you want to create the following application:

Initial message: Welcome to XYZ Telecom; press 1 for sales, 2 for support, or wait for the next available agent.

First, you will need to record these phrases on files:

- **menu.wav:** Welcome to XYZ Telecom; press 1 for sales, 2 for support, or wait for the next available agent.
- Option 1: Transfer to extension 6001
- Option 2: Transfer to extension 6003
- No option selected (timeout): Transfer to the operator.
- Invalid option: Return to the beginning of the recording

Recording an announcement

FreePBX is very ingenious for recording announcements. Use the System Recordings option. In the menu below, you should indicate your phone extension. After doing so, it will release the code *77 to start a recording. Follow the recording instructions; in the end, the system will allow you to name the recording and save it. Thus, you can generate all the announcements required by your auto-attendant system.

System Recordings

Add Recording

Step 1: Record or upload

If you wish to make and verify recordings from your phone, please enter your extension number here:

Alternatively, upload a recording in any supported asterisk format. Note that if you're using .wav, (eg, recorded with Micr

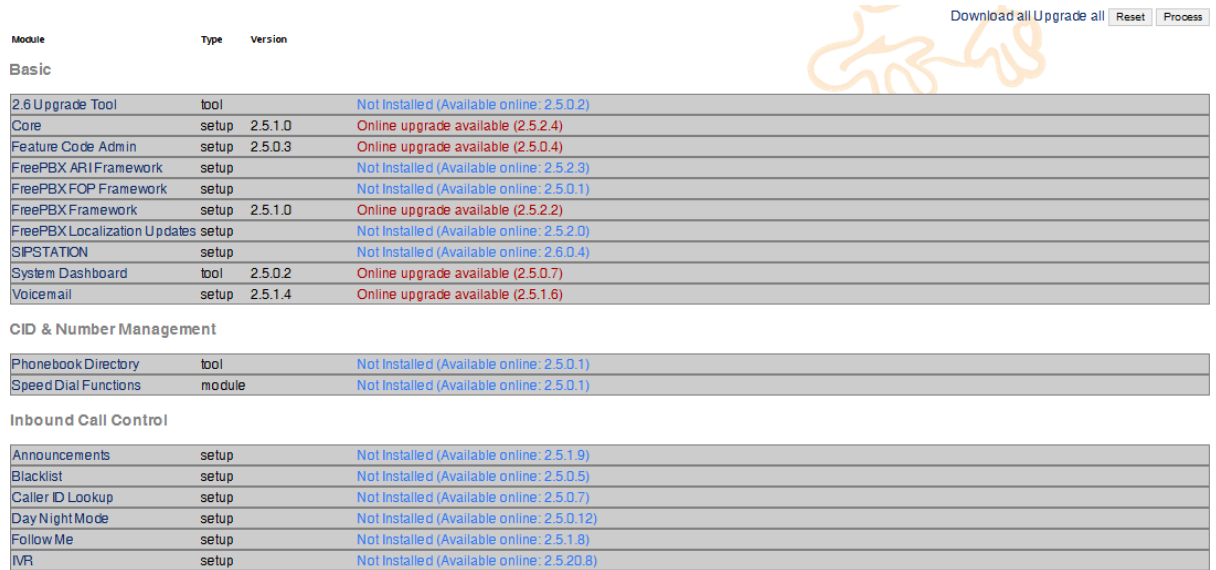
Step 2: Name

Name this Recording:

Click "SAVE" when you are satisfied with your recording

Creating the auto-attendant

To create the auto-attendant, you should install a new module. This operation is very simple: Access the Module Admin menu and select Search Online Updates. FreePBX will show you a series of modules available. Double click on the IVR module and then select Download and Install.



Module	Type	Version	
Basic			
2.6 Upgrade Tool	tool		Not Installed (Available online: 2.5.0.2)
Core	setup	2.5.1.0	Online upgrade available (2.5.2.4)
Feature Code Admin	setup	2.5.0.3	Online upgrade available (2.5.0.4)
FreePBX ARI Framework	setup		Not Installed (Available online: 2.5.2.3)
FreePBX FOP Framework	setup		Not Installed (Available online: 2.5.0.1)
FreePBX Framework	setup	2.5.1.0	Online upgrade available (2.5.2.2)
FreePBX Localization Updates	setup		Not Installed (Available online: 2.5.2.0)
SIPSTATION	setup		Not Installed (Available online: 2.6.0.4)
System Dashboard	tool	2.5.0.2	Online upgrade available (2.5.0.7)
Voicemail	setup	2.5.1.4	Online upgrade available (2.5.1.6)
CID & Number Management			
Phonebook Directory	tool		Not Installed (Available online: 2.5.0.1)
Speed Dial Functions	module		Not Installed (Available online: 2.5.0.1)
Inbound Call Control			
Announcements	setup		Not Installed (Available online: 2.5.1.9)
Blacklist	setup		Not Installed (Available online: 2.5.0.5)
Caller ID Lookup	setup		Not Installed (Available online: 2.5.0.7)
DayNightMode	setup		Not Installed (Available online: 2.5.0.12)
Follow Me	setup		Not Installed (Available online: 2.5.1.8)
IVR	setup		Not Installed (Available online: 2.5.20.8)

At the end of the page, choose the process button. This menu allows for a series of operations, such as the upgrades of the modules.

Creating the IVR itself

Below is the screen to create the auto-attendant IVR. Let's explain the configuration item per item.

Digital Receptionist

Edit Menu autoattendant

Change Name
Announcement
Timeout
Enable Directory
VM Return to IVR
Directory Context
Enable Direct Dial
Loop Before t-dest
Timeout Message
Loop Before i-dest
Invalid Message
Repeat Loops:

Return to IVR
 Terminate Call: Hangup
 Extensions: <8000> 8000
 Voicemail: <8000> 8000 (busy)
 IVR: autoattendant

Leave blank to remove

Return to IVR
 Terminate Call: Hangup
 Extensions: <8000> 8000
 Voicemail: <8000> 8000 (busy)
 IVR: autoattendant

Leave blank to remove

Return to IVR
 Terminate Call: Hangup
 Extensions: <8001> 8001
 Voicemail: <8000> 8000 (busy)
 IVR: autoattendant

Leave blank to remove

Return to IVR
 Terminate Call: Hangup
 Extensions: <8003> 8003
 Voicemail: <8000> 8000 (busy)
 IVR: autoattendant

Leave blank to remove

The screen above includes a series of options that should be configured for the IVR to work. Below are some of the options selected.

Name: auto-attendant

Announcement: menu.wav

Timeout: 10

Direct dial enable

Option 1

Option 2

Option i (invalid)

Option t (timeout)

Name of the recording to be played.

Time in seconds to wait for an option

Enable the user to dial directly an extensions

When 1 is dialed, transfer to extension 6000

When 2 is dialed, transfer to extension 6001

If some digit is invalid (3-9), return to the IVR

If no digit is dialed, transfer to extension 6003

Creating a conference room

To create a conference room, we will need to install a new module in the freePBX called conference. To create a conference, simply add the number and the XXX [missing word?] of the conference room. Dial the number of the room to test the conference.

Add Conference

Add Conference

Conference Number:	<input type="text" value="8999"/>
Conference Name:	<input type="text" value="room #1"/>
User PIN:	<input type="text"/>
Admin PIN:	<input type="text"/>

Conference Options

Join Message:	<input type="text" value="None"/>
Leader Wait:	<input type="text" value="No"/>
Quiet Mode:	<input type="text" value="No"/>
User Count:	<input type="text" value="No"/>
User join/leave:	<input type="text" value="No"/>
Music on Hold:	<input type="text" value="No"/>
Allow Menu:	<input type="text" value="No"/>
Record Conference:	<input type="text" value="No"/>

Important: The conference room won't work if you don't have a timing source. Check if you have at least one DAHDI card loaded. If your system does not use any telephony card, please load the module dahdi_dummy in the Linux command line before trying.

Summary

This chapter has provided significant information that will enable you to build a complete PBX solution with voicemail, IVR, and conference room in less than one hour. Initially, I was against the adoption of graphical user interfaces because troubleshooting is harder. However, the level of

productivity obtained by the use of a GUI surpasses the possible disadvantages. I'm not afraid to say that, for low to medium complexity IP PBX setups, freePBX is my choice.