



# **Asterisk Administrator Guide**

**Asterisk Development Team <[asteriskteam@digium.com](mailto:asteriskteam@digium.com)>**

1. About the Project	5
1.1 A Brief History of the Asterisk Project	6
1.2 Asterisk as a Swiss Army Knife of Telephony	7
1.3 Asterisk Versions	8
1.4 License Information	10
1.4.1 Voice Prompts and Music on Hold License	12
2. Getting Started	13
2.1 Beginning Asterisk	14
2.2 Installing Asterisk	16
2.2.1 Installing AsteriskNOW	17
2.2.2 Upgrading AsteriskNOW	23
2.2.3 Installing Asterisk From Source	26
2.2.4 Alternate Install Methods	51
2.2.5 Installing Asterisk on Non-Linux Operating Systems	60
2.3 Hello World	63
3. Operation	67
3.1 System Requirements	68
3.1.1 Compiler	69
3.1.2 System Libraries	70
3.2 Running Asterisk	71
3.2.1 Stopping and Restarting Asterisk From The CLI	74
3.3 Maintenance and Upgrades	75
3.3.1 Asterisk Backups	76
3.3.2 Updating or Upgrading Asterisk	77
3.4 Logging	79
3.4.1 Basic Logging Commands	80
3.4.2 Basic Logging Start-up Options	81
3.4.3 Call Identifier Logging	82
3.4.4 Collecting Debug Information	83
3.4.5 Queue Logs	85
3.4.6 Verbosity in Core and Remote Consoles	87
3.5 Asterisk Command Line Interface	88
3.5.1 Connecting to the Asterisk CLI	89
3.5.2 CLI Syntax and Help Commands	90
3.5.3 Creating and Manipulating Channels from the CLI	92
3.5.4 Simple CLI Tricks	94
3.6 Asterisk Audio and Video Capabilities	95
4. Fundamentals	99
4.1 Asterisk Architecture	100
4.1.1 Asterisk Architecture, The Big Picture	101
4.1.2 Types of Asterisk Modules	103
4.2 Directory and File Structure	113
4.3 Asterisk Configuration	115
4.3.1 Asterisk Configuration Files	116
4.3.2 Database Support Configuration	129
4.3.3 Sorcery	135
4.4 Asterisk Internal Database	140
4.4.1 SQLite3 astdb back-end	141
4.5 Key Concepts	143
4.5.1 Bridges	144
4.5.2 Channels	148
4.5.3 Frames	151
4.5.4 States and Presence	159
4.5.5 The Stasis Message Bus	167
5. Configuration	169
5.1 Core Configuration	170
5.1.1 Asterisk Main Configuration File	171
5.1.2 Timing Interfaces	174
5.1.3 Asterisk Builtin mini-HTTP Server	176
5.1.4 Logging Configuration	177
5.1.5 Asterisk CLI Configuration	179
5.1.6 Configuring the Asterisk Module Loader	180
5.1.7 Configuring Localized Tone Indications	181
5.1.8 Video Telephony	182
5.1.9 Video Console	183
5.1.10 Named ACLs	186
5.2 Channel Drivers	189
5.2.1 SIP	190
5.2.2 Inter-Asterisk eXchange protocol, version 2 (IAX2)	270
5.2.3 DAHDI	283
5.2.4 Local Channel	284
5.2.5 Motif	295

5.2.6 mISDN	300
5.2.7 Mobile Channel	312
5.2.8 Unistim	324
5.2.9 Skinny	339
5.2.10 RTP Packetization	345
5.3 Dialplan	347
5.3.1 Contexts, Extensions, and Priorities	348
5.3.2 Special Dialplan Extensions	350
5.3.3 Include Statements	351
5.3.4 Switch Statements	355
5.3.5 Variables	356
5.3.6 Pattern Matching	383
5.3.7 Subroutines	386
5.3.8 Expressions	401
5.3.9 Conditional Applications	414
5.4 Features	415
5.4.1 Feature Code Call Transfers	416
5.4.2 One-Touch Features	419
5.4.3 Call Pickup	421
5.4.4 Built-in Dynamic Features	424
5.4.5 Custom Dynamic Features	425
5.4.6 Call Parking	427
5.5 Applications	438
5.5.1 Answer, Playback, and Hangup Applications	439
5.5.2 Bridge Application	440
5.5.3 Dial Application	441
5.5.4 Directory Application	442
5.5.5 Early Media and the Progress Application	443
5.5.6 External IVR Interface	444
5.5.7 MacroExclusive	447
5.5.8 The Read Application	448
5.5.9 The Verbose and NoOp Applications	449
5.5.10 SMS	450
5.5.11 Voicemail	452
5.5.12 Conferencing Applications	464
5.6 Functions	482
5.6.1 Simple Message Desk Interface (SMDI) Integration	483
5.7 Reporting	486
5.7.1 Call Detail Records (CDR)	487
5.7.2 Channel Event Logging (CEL)	498
5.8 Interfaces	518
5.8.1 Asterisk Calendaring	519
5.8.2 Asterisk Call Files	525
5.8.3 Asterisk Gateway Interface (AGI)	527
5.8.4 Asterisk Manager Interface (AMI)	528
5.8.5 Asterisk REST Interface (ARI)	550
5.8.6 Back-end Database and Realtime Connectivity	704
5.8.7 Distributed Device State	727
5.8.8 Simple Network Management Protocol (SNMP) Support	740
5.8.9 Speech Recognition API	752
5.8.10 Utilizing the StatsD Dialplan Application	757
6. Deployment	759
6.1 Basic PBX Functionality	760
6.1.1 The Most Basic PBX	761
6.1.2 Creating SIP Accounts	762
6.1.3 Registering Phones to Asterisk	765
6.1.4 Creating Dialplan Extensions	766
6.1.5 Making a Phone Call	767
6.1.6 Auto-attendant and IVR Menus	768
6.1.7 Adding Voice Mail to Dialplan Extensions	775
6.2 Deployment In Your Network	778
6.3 Emergency Calling	779
6.4 Important Security Considerations	780
6.4.1 Network Security	781
6.4.2 Dialplan Security	782
6.4.3 Log Security	783
6.4.4 Asterisk Security Webinars	784
6.5 Privacy Configuration	785
6.5.1 FTC Don't Call List	786
6.5.2 Fighting Autodialers	787
6.5.3 Fighting Empty Caller ID	788
6.5.4 Using Welcome Menus for Privacy	789

6.5.5 Making life difficult for telemarketers .....	790
6.5.6 Using Call Screening .....	791
6.5.7 Call Screening Options .....	792
6.5.8 Screening Calls with Recorded Introductions .....	793
6.6 Internationalization and Localization .....	796
6.6.1 Asterisk Sounds Packages .....	797
6.6.2 Sound Prompt Searching based on Channel Language .....	800
6.7 Troubleshooting .....	801
6.7.1 SIP Retransmissions .....	802
6.7.2 Troubleshooting Asterisk Module Loading .....	804
6.7.3 Unable to connect to remote Asterisk .....	807
6.8 IPv6 Support .....	809
6.9 PSTN Connectivity .....	810
6.9.1 Advice of Charge .....	811
6.9.2 Caller ID in India .....	814
6.9.3 Signaling System Number 7 .....	816
6.10 Secure Calling .....	818
6.10.1 Secure Calling Specifics .....	819
6.10.2 Secure Calling Tutorial .....	820
6.10.3 SIP TLS Transport .....	831
6.11 Reference Use Cases for Asterisk .....	833
6.11.1 Super Awesome Company .....	834
7. Asterisk Security Vulnerabilities .....	837
8. Asterisk Community .....	838
8.1 Asterisk Community Code of Conduct .....	839
8.2 Asterisk Community Services .....	841
8.3 Asterisk Issue Guidelines .....	842
8.4 Asterisk Module Support States .....	846
8.5 Community Services Signup .....	852
8.6 IRC .....	853
8.7 Mailing Lists .....	854
8.8 Wiki Organization and Style Guide .....	855

# About the Project

# A Brief History of the Asterisk Project

## Linux Support Services

Way, way back in 1999 a young man named Mark Spencer was finishing his Computer Engineering degree at Auburn University when he hit on an interesting business concept. 1999 was the high point in the .com revolution (aka bubble), and thousands of businesses world-wide were discovering that they could save money by using the open source Linux operating system in place of proprietary operating systems. The lure of a free operating system with open access to the source code was too much to pass up. Unfortunately there was little in the way of commercial support available for Linux at that time. Mark decided to fill this gap by creating a company called "Linux Support Services". LSS offered a support hotline that IT professionals could (for a fee) call to get help with Linux.

The idea took off. Within a few months, Mark had a small office staffed with Linux experts. Within a few more months the growth of the business expanded demanded a "real" phone system that could distribute calls evenly across the support team, so Mark called up several local phone system vendors and asked for quotes. Much to his surprise, the responses all came back well above \$50,000 -- far more than Mark had budgeted for the project. Far more than LSS could afford.

## Finding a Solution

Rather than give in and take out a small business loan, Mark made a pivotal decision. He decided to write his own phone system. Why not? A phone system is really just a computer running phone software, right? Fortunately for us, Mark had no idea how big a project he had taken on. If he had known what a massive undertaking it was to build a phone system from the ground up might have gritted his teeth, borrowed the money and spent the next decade doing Linux support. But he didn't know what he didn't know, and so he started to code. And he coded. And he coded.

Mark had done his engineering co-op at Adtran, a communications and networking device manufacturer in Huntsville, AL. There he had cut his teeth on telecommunications system development, solving difficult problems generating a prodigious amount of complex code in short time. This experience proved invaluable as he began to frame out the system which grew into Asterisk. In only a few months Mark crafted the original Asterisk core code. As soon as he had a working prototype he published the source code on the Internet, making it available under the GPL license (the same license used for Linux).

Within a few months the idea of an "open source PBX" caught on. There had been a few other open source communications projects, but none had captured the imagination of the global population of communications geeks like Asterisk. As Mark labored on the core system, hundreds (now thousands) of developers from all over the world began to submit new features and functions.

## Digium

What became of Linux Support Services? In 2001, Linux Support Services changed its name to Digium. Digium continued to develop Asterisk in collaboration with the community, provide services to support the development community, as well as build commercial products and services around Asterisk which have fueled growth in both Digium and the Asterisk project. You can find out more about Digium at [Digium's website](#) and on [wikipedia](#).

## Asterisk in the Present

Asterisk is constantly evolving to meet the needs of the project's user-base. It's difficult to summarize the vast scope of everything that Asterisk can do as a communications toolkit. We'll list some resources that give you an idea of what is going on in the Asterisk project at present.

- [Asterisk Versions](#) :Shows release time lines, support and EOL schedules
- [Roadmap section](#) :Information from developer conferences and planning sessions
- [CHANGES](#) :A document in Asterisk trunk, shows functionality changes between major versions
- [UPGRADE](#) :A document in Asterisk trunk, shows breaking changes, deprecation of specific features and important info on upgrading.
- [Mailing lists](#) :The dev list is a great list to see what hot topics the developers are discussing in real-time.

# Asterisk as a Swiss Army Knife of Telephony

## What Is Asterisk?

People often tend to think of Asterisk as an "open source PBX" because that was the focus of the original development effort. But calling Asterisk a PBX is both selling it short (it is much more) and overstating it (it can be much less). It is true that Asterisk started out as a phone system for a small business (see the "Brief History" section for the juicy details) but in the decade since it was originally released it has grown into a universal tool for building communications applications. Today Asterisk powers not only IP PBX systems but also VoIP gateways, call center systems, conference bridges, voicemail servers and all kinds of other applications that involve real-time communications.

Asterisk is not a PBX but is the engine that powers PBXs. Asterisk is not an IVR but is the engine that powers IVRs. Asterisk is not a call center ACD but is the engine that powers ACD/queueing systems.

Asterisk is to communications applications what the Apache web server is to web applications. Apache is a web server. Asterisk is a communication server. Apache handles all the low-level details of sending and receiving data using the HTTP protocol. Asterisk handles all the low level details of sending and receiving data using lots of different communication protocols. When you install Apache, you have a web server but its up to you to create the web applications. When you install Asterisk, you have a communications server but its up to you to create the communications applications.

Web applications are built out of HTML pages, CSS style sheets, server-side processing scripts, images, databases, web services, etc. Asterisk communications applications are built out Dialplan scripts, configuration files, audio recordings, databases, web services, etc. For a web application to work, you need the web server connected to the Internet. For a communications application to work, you need the communications server connected to communication services (VoIP or PSTN). For people to be able to access your web site you need to register a domain name and set up DNS entries that point "www.yourdomain.com" to your server. For people to access your communications system you need phone numbers or VoIP URIs that send calls to your server.

In both cases the server is the plumbing that makes your application work. The server handles the low-level complexities and allows you, the application developer, to concentrate on the application logic and presentation. You don't have to be an expert on HTTP to create powerful web applications, and you don't have to be an expert on SIP or Q.931 to create powerful communications applications.

Here's a simple example. The following HTML script, installed on a working web server, prints "Hello World" in large type:

```
<html>
  <head>
    <title>Hello World Demo</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

The following Dialplan script answers the phone, waits for one second, plays back "hello world" then hangs up.

```
exten => 100,1,Answer()
exten => 100,n,Wait(1)
exten => 100,n,Playback(hello-world)
exten => 100,n,Hangup()
```

In both cases the server components are handling all of the low level details of the underlying protocols. Your application doesn't have to worry about the byte alignment, the packet size, the codec or any of the thousands of other critical details that make the application work. This is the power of an engine.

## Who Uses Asterisk?

Asterisk is created by communication system developers, for communication system developers. As an open source project, Asterisk is a collaboration between many different individuals and companies, all of which need a flexible communications engine to power their applications.

# Asterisk Versions

There are multiple supported feature frozen releases of Asterisk. Once a release series is made available, it is supported for some period of time. During this initial support period, releases include changes to fix bugs that have been reported. At some point, the release series will be deprecated and only maintained with fixes for security issues. Finally, the release will reach its End of Life, where it will no longer receive changes of any kind.

The type of release defines how long it will be supported. A Long Term Support (LTS) release will be fully supported for 4 years, with one additional year of maintenance for security fixes. Standard releases are supported for a shorter period of time, which will be at least one year of full support and an additional year of maintenance for security fixes.

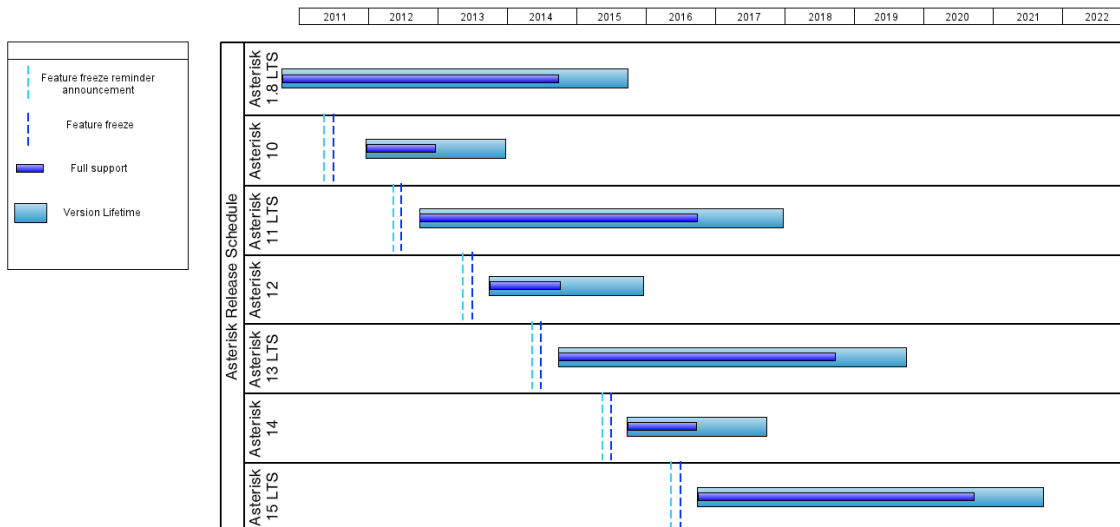
The following table shows the release time lines for all releases of Asterisk, including those that have reached End of Life.

Release Series	Release Type	Release Date	Security Fix Only	EOL
1.2.X		2005-11-21	2007-08-07	2010-11-21
1.4.X	LTS	2006-12-23	2011-04-21	2012-04-21
1.6.0.X	Standard	2008-10-01	2010-05-01	2010-10-01
1.6.1.X	Standard	2009-04-27	2010-05-01	2011-04-27
1.6.2.X	Standard	2009-12-18	2011-04-21	2012-04-21
1.8.X	LTS	2010-10-21	2014-10-21	2015-10-21
10.X	Standard	2011-12-15	2012-12-15	2013-12-15
11.x	LTS	2012-10-25	2016-10-25	2017-10-25
12.x	Standard	2013-12-20	2014-12-20	2015-12-20
13.x	LTS	2014-10-24	2018-10-24	2019-10-24
14.x	Standard	2016-10 (tentative)	2017-10 (tentative)	2018-10 (tentative)
15.x	LTS	2017-10 (tentative)	2021-10 (tentative)	2022-10 (tentative)

New releases of Asterisk will be made roughly once a year, alternating between standard and LTS releases. Within a given release series that is fully supported, bug fix updates are provided roughly every 4 weeks. For a release series that is receiving only maintenance for security fixes, updates are made on an as needed basis.

If you're not sure which one to use, choose either the latest release for the most up to date features, or the latest LTS release for a platform that may have less features, but will usually be around longer.

The schedule for Asterisk releases is visualized below (which is subject to change at any time):



For developers, it is useful to be aware of when the feature freeze for a particular branch will occur. The feature freeze for a branch will occur 3 months prior to the release of a new Asterisk version, and a reminder announcement will be posted to the asterisk-dev mailing list approximately 60 days prior to the feature freeze. Asterisk versions are slated to be released the 3rd Wednesday of October. The feature freeze for a branch will occur the 3rd Wednesday of July. An announcement reminder will be posted to the asterisk-dev mailing list the 3rd Wednesday of May. Feature freeze consists of the creation of two branches: One for the release series and one for the initial release. Features can continue to be placed into the release series branch according to policy but the initial release branch will be frozen.



<b>Feature Freeze Announcement Reminder</b>	3rd Wednesday of May
<b>Feature Freeze of Asterisk Branch</b>	3rd Wednesday of July
<b>First Release of Asterisk from Branch</b>	3rd Wednesday of October

# License Information

- [Asterisk License Information](#)
  - [Asterisk Sounds](#)
- [Frequently Asked Questions about Licensing](#)
  - [What is an open-source license?](#)
  - [What is the GNU General Public License?](#)
  - [What if I want to distribute or license Asterisk under a different license?](#)
  - [How can I contribute to Asterisk?](#)

## Asterisk License Information

Asterisk is distributed under the GNU General Public License version 2 and is also available under alternative licenses negotiated directly with Digium, Inc. If you obtained Asterisk under the GPL, then the GPL applies to all loadable Asterisk modules used on your system as well, except as defined below. The GPL (version 2) is included in this source tree in the file COPYING.

This package also includes various components that are not part of Asterisk itself; these components are in the 'contrib' directory and its subdirectories. These components are also distributed under the GPL version 2 as well.

Digium, Inc. (formerly Linux Support Services) holds copyright and/or sufficient licenses to all components of the Asterisk package, and therefore can grant, at its sole discretion, the ability for companies, individuals, or organizations to create proprietary or Open Source (even if not GPL) modules which may be dynamically linked at runtime with the portions of Asterisk which fall under our copyright/license umbrella, or are distributed under more flexible licenses than GPL.

If you wish to use our code in other GPL programs, don't worry -- there is no requirement that you provide the same exception in your GPL'd products (although if you've written a module for Asterisk we would strongly encourage you to make the same exception that we do).

Specific permission is also granted to link Asterisk with OpenSSL, OpenH323 and/or the UW IMAP Toolkit and distribute the resulting binary files.

In addition, Asterisk implements several management/control protocols. This includes the Asterisk Manager Interface (AMI), the Asterisk Gateway Interface (AGI), and the Asterisk REST Interface (ARI). It is our belief that applications using these protocols to manage or control an Asterisk instance do not have to be licensed under the GPL or a compatible license, as we believe these protocols do not create a 'derivative work' as referred to in the GPL. However, should any court or other judiciary body find that these protocols do fall under the terms of the GPL, then we hereby grant you a license to use these protocols in combination with Asterisk in external applications licensed under any license you wish.

The 'Asterisk' name and logos are trademarks owned by Digium, Inc., and use of them is subject to our trademark licensing policies. If you wish to use these trademarks for purposes other than simple redistribution of Asterisk source code obtained from Digium, you should contact our licensing department to determine the necessary steps you must take. For more information on this policy, please read: <http://www.digium.com/en/company/profile/trademarkpolicy.php>

If you have any questions regarding our licensing policy, please contact us:

+1.877.344.4861 (via telephone in the USA)  
+1.256.428.6000 (via telephone outside the USA)  
+1.256.864.0464 (via FAX inside or outside the USA)  
IAX2/pbx.digium.com (via IAX2)  
licensing@digium.com (via email)

Digium, Inc.  
445 Jan Davis Drive NW  
Huntsville, AL 35806  
United States

## Asterisk Sounds

License information for Asterisk sounds can be found in the [Voice Prompts and Music on Hold License](#) section.

## Frequently Asked Questions about Licensing

### What is an open-source license?

Wikipedia has a great article on [open-source licenses](#), and [opensource.org](#) is a pretty definitive resource.

### What is the GNU General Public License?

The GPL is a specific open-source license. Reading the preamble [at this link](#) is a great introduction, and below that is the full license text.

### What if I want to distribute or license Asterisk under a different license?

Digium distributes Asterisk under a multi-licensing model often referred to as [Dual-licensing](#) and is additionally made possible by a [Contributors License Agreement](#). This allows Digium to provide Asterisk under licenses other than the GPL. Digium provides information on their alternative commercial licensing

at their website.

## How can I contribute to Asterisk?

Documentation, new features, bug fixes, testing, protocol and programming expertise,, and general feedback are all welcome to the project. There is an [overview](#) that points to many resources for developers, also you can see the [guidelines](#) for contribution to see how it works.

## Voice Prompts and Music on Hold License

### Voice Prompts

All voice prompt contributions distributed with Asterisk or available on the Asterisk downloads site are licensed as [Creative Commons Attribution-Share Alike 3.0](#). The process for contributing sound files can be found in the [Asterisk Sounds Submission Process](#) section.

### Music On Hold

The Hold (on hold) music included with the Asterisk distribution has been sourced from [opsound.org](#) which itself distributes the music under [Creative Commons Attribution-ShareAlike 2.5](#) license.

# Getting Started

When learning Asterisk it is important to start off on the right foot, so this section of the wiki covers orientation for learning Asterisk as well as installation and a simple Hello World style tutorial. These items are foundational, as knowing how to install Asterisk right the first time and where to locate the right help resources will save you a ton of time down the road.

Those interested in Asterisk training courses and certifications may visit <http://www.asterisk.org/products/training>

# Beginning Asterisk

## Asterisk is...

- an [Open Source](#) software development project
- written in the [C Programming Language](#)
- running on [Linux](#) (or other types of [Unix](#) )
- powering [Business Telephone Systems](#)
- connecting many different [Telephony](#) protocols
- a toolkit for building many things:
  - an [IP PBX](#) with many powerful features and [applications](#)
  - VoIP Gateways
  - Conferencing systems
  - and much, much more
- supporting VoIP Phones as well as [PSTN](#) and [POTS](#)
- speaking [SIP](#) , the most common [VoIP](#) protocol, among others

## YouTube Videos

- [System 5 Episode on Asterisk](#) (from 2006 - see Asterisk Wiki for current installation instructions)
- [Official Asterisk Channel](#)
- [Asterisk 123: Intro to Asterisk](#) from Astricon 10
- [Asterisk 12 Overview](#) from Astricon 10

## Resources for understanding

- Acronyms and Terminology
  - [Telephony Terminology](#)
  - [Asterisk Terms Glossary](#)
  - [Telecom Acronyms](#) (very comprehensive)
- Telephony Protocols
  - [IP Telephony Protocols Overview](#)
  - [SIP Overview](#)
  - [A Hitchhiker's Guide to SIP](#)
- Linux & Unix
  - [Linux Newbie Guide](#)
  - [Beginner Tutorials](#)
  - [Unix Beginner Tutorial](#)
- Installing and Configuring Asterisk
  - [Asterisk: The Definitive Guide](#) 3rd Edition
  - [The Asterisk Wiki](#)
- C Programming
  - [C Programming Tutorial](#)
  - [Interactive C Tutorial](#)
  - [C Programming Quick Guide](#)

## Where to get help

- Email Lists and Live Chat (IRC)
  - [Asterisk Mailing List and IRC](#)
- Web Discussion Forums
  - [Asterisk Community Forums](#)
- Online Community
  - [Voip Users Conference](#) main site and on [Google+](#)

## Avoiding obsolete or incorrect information

When reading about Asterisk on the web, you may come across old or incorrect information.

- Check which version of Asterisk is mentioned. There are significant changes in every version.
- Check the published date of the article if the Asterisk version isn't provided.
- Take things with a grain of salt until checked with another resource or proven correct through your own testing.
- Refer to the [Asterisk Wiki](#) and the [Official Asterisk Youtube Channel](#) for the most accurate and up to date details on the specific version of Asterisk you are using.

Please note that it is always possible that even the official documentation does not match what is written into the source code itself. If you find something lacking or incorrect in the Asterisk documentation, please communicate it through comments on the [Asterisk Wiki](#) or by filing an issue through the [Asterisk Issues Tracker](#) .



# Installing Asterisk

Now that you know a bit about Asterisk and how it is used, it's time to get you up and running with your own Asterisk installation. There are various ways to get started with Asterisk on your own system:

- Install an Asterisk-based Linux distribution such as AsteriskNOW. This takes care of installing Linux, Asterisk, and some web-based interfaces all at the same time, and is the easiest way to get started if you're new to Linux and/or Asterisk.
- If you're already familiar with Linux or Unix, you can simply install packages for Asterisk and its related tools using the package manager in your operating system. We'll cover this in more detail below in [Alternate Install Methods](#).
- For the utmost in control of your installation, you can compile and install Asterisk (and its related tools) from source code. We'll explain how to do this in [Installing Asterisk From Source](#).



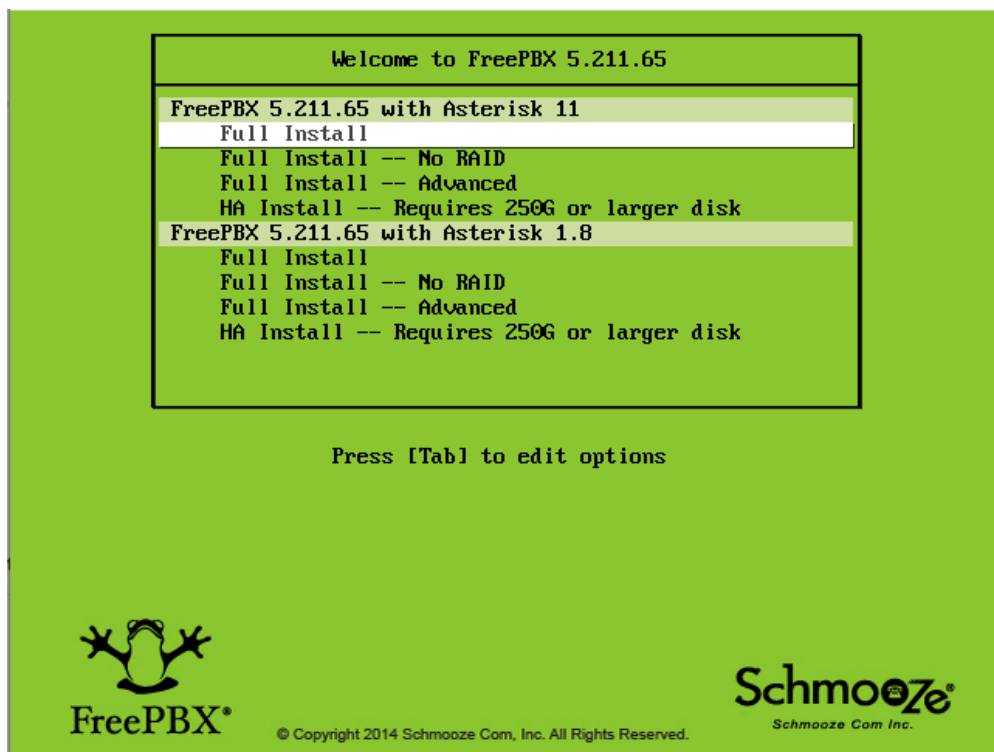
# Installing AsteriskNOW



## Installing AsteriskNOW Official Distro

The simplest way to install AsteriskNOW is to follow these instructions:

1. Download the ISO file from <http://asterisk.org/downloads/asterisknow>.
2. Burn the ISO file to a DVD or CD. If you need help doing this, simply Google "burn ISO to DVD" and you'll find plenty of instructions OR click this link so I can Google it for you: "Google burn ISO to DVD".
3. Select a computer to install AsteriskNOW. **EVERYTHING ON THIS COMPUTER WILL BE DELETED AND REPLACED WITH THE AsteriskNOW DISTRO.** Configure the computer to boot from a DVD or CD. Insert the DVD or CD into the computer and turn it on. NOTE: You must be connected to the internet to run the installer. If you're installing using a USB drive, you may encounter a "kickstart" error while installing. If you do, don't fret! Just keep hitting enter when the prompts appear and everything will probably work just fine.
4. The installer will begin with a prompt to select the Asterisk Version you wish to install.



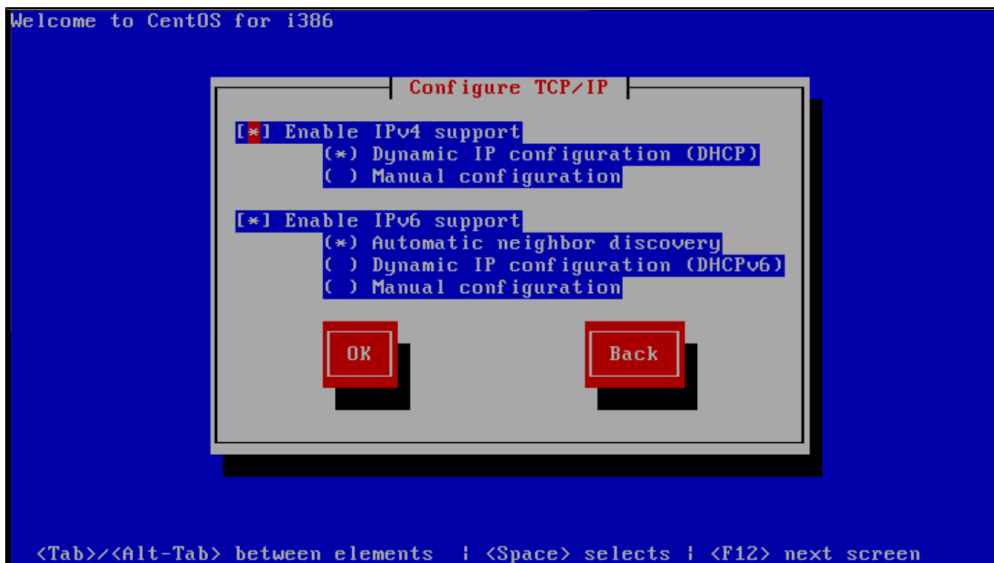
5. The system will present you with a window showing that it is retrieving images while it downloads the install package from the internet. This should take 3-5 minutes.

```

usbcore: registered new device driver usb
PCI: Using ACPI for IRQ routing
NetLabel: Initializing
NetLabel: domain hash size = 128
NetLabel: protocols = UNLABELED CIPSOv4
NetLabel: unlabeled traffic allowed by default
hpet0: at MMIO 0xfed00000, IRQs 2, 8, 0
hpet0: 3 comparators, 64-bit 100.000000 MHz counter
Switching to clocksource kvm-clock
pnp: PnP ACPI init
ACPI: bus type pnp registered
pnp: PnP ACPI: found 7 devices
ACPI: ACPI bus type pnp unregistered
NET: Registered protocol family 2
IP route cache hash table entries: 32768 (order: 5, 131072 bytes)
TCP established hash table entries: 131072 (order: 8, 1048576 bytes)
TCP bind hash table entries: 65536 (order: 7, 524288 bytes)
TCP: Hash tables configured (established 131072 bind 65536)
TCP reno registered
NET: Registered protocol family 1
pci 0000:00:00.0: Limiting direct PCI/PCI transfers
pci 0000:00:01.0: PIIX3: Enabling Passive Release
pci 0000:00:01.0: Activating ISA DMA hang workarounds
Trying to unpack rootfs image as initramfs...

```

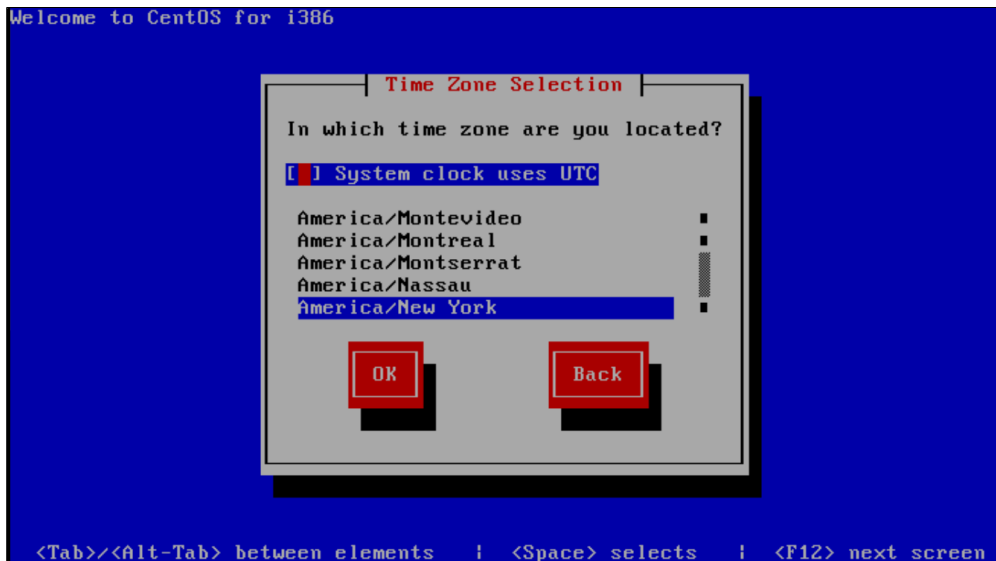
6. After the system boots you will see options to configure your network. The default selections are fine in most cases, so just press TAB until the red "OK" box is highlighted in white and then press ENTER. You can also choose the option to manually configure your network connections, if desired.



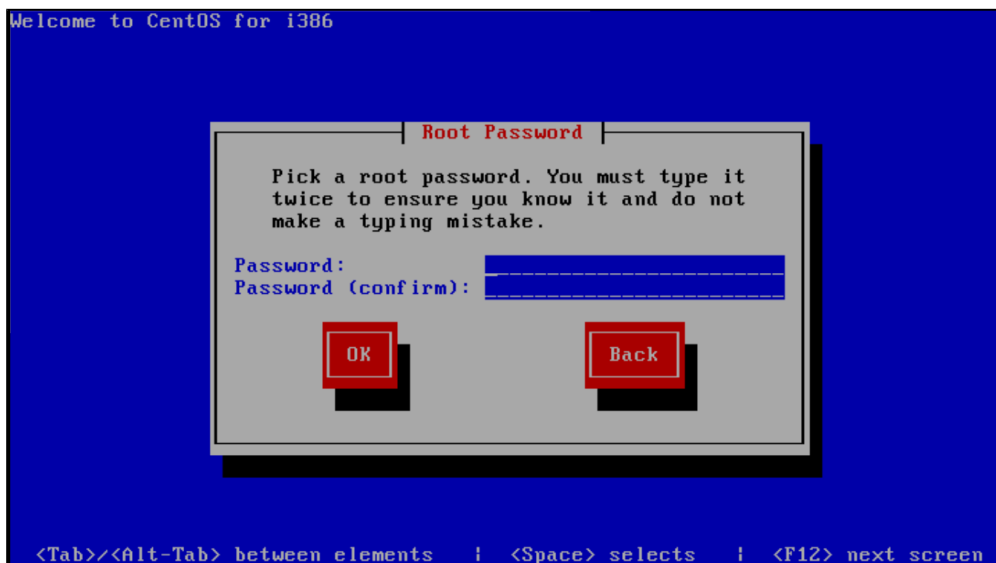
Once you hit "OK" the system will configure your Network Connections.



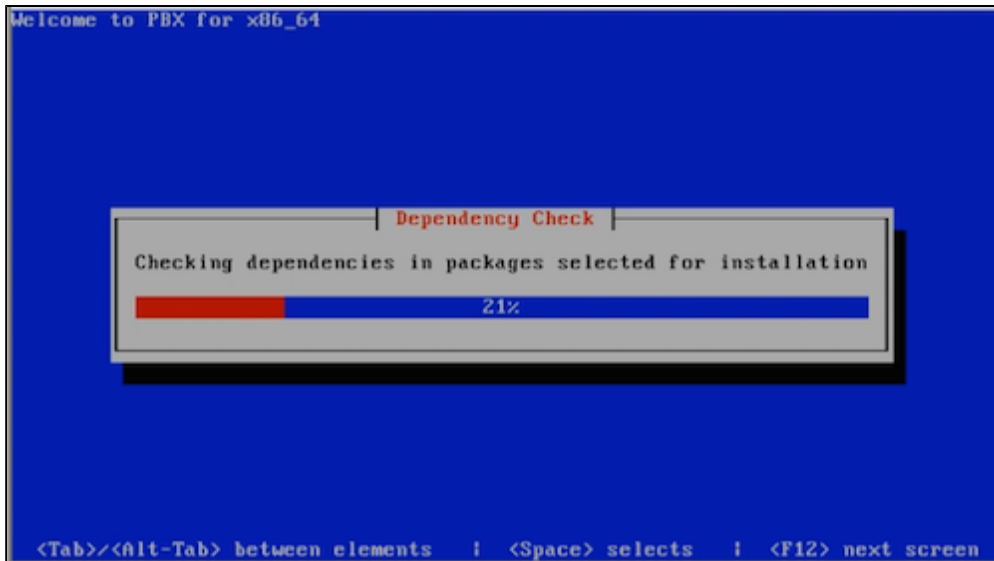
7. Eventually, you will reach the "Time Zone Selection" screen. If your system clock uses GMT (most do not), hit space. Then, hit TAB to move to the time zone selection area. Use the up and down arrows to select the time zone where you will use the system, and then hit TAB until the red "OK" button is highlighted in white. Then, hit ENTER.



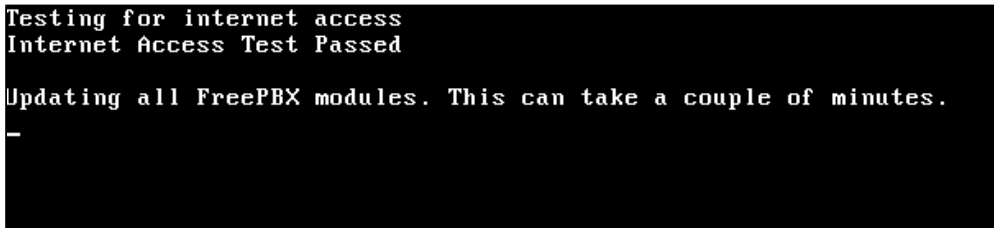
8. The installer will ask you to select your root password. The root password is the password you'll use to login to the Linux command prompt later. Selecting a secure password is very important. Type the password, hit TAB, type it again, hit TAB, and then hit ENTER.



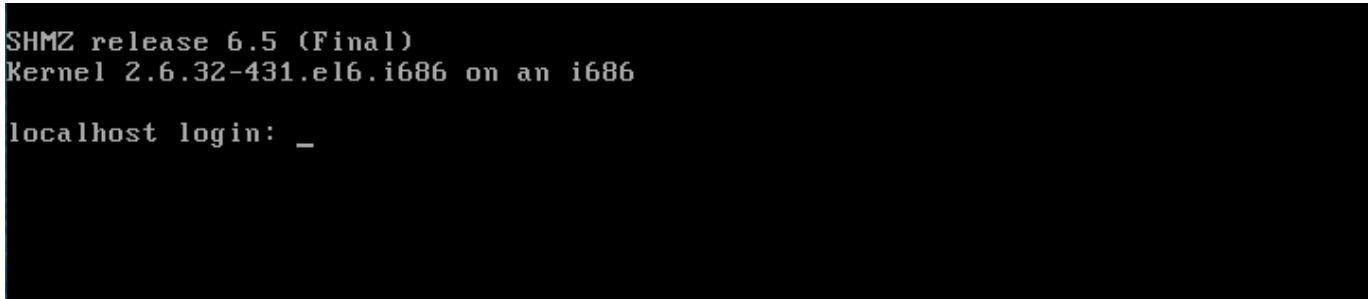
9. The installer will do a dependency check, format your hard drive, and then start the package installation process. There may be a significant delay before the installation actually starts, so be patient. Eventually, the installation will show you a progress bar indicating the percentage completed and the time elapsed/remaining. That process should take between 20 and 30 minutes, it will then reboot.



10. A few additional packages will be installed and updated after the reboot from the install. This can take 10-15 minutes.



11. Once completed you'll reach the Linux console/command prompt login. You can login here using the username "root" without quotes, and the root password you selected earlier.



12. After you login, you should see the IP address of your PBX as defined below.



15. You have now successfully installed AsteriskNOW with FreePBX GUI.

## Need additional assistance?

There is an active community of AsteriskNOW users, integrators, and developers, who provide community support on the AsteriskNOW [forums](#).

GUI support is provided by FreePBX. Please visit the [FreePBX](#) wiki for additional help by clicking [here](#).



The Asterisk Company

Digium also offers a complete line of cloud and hardware products to complete your AsteriskNOW solution. These products give you a way to connect to the PSTN so you can start using AsteriskNOW. These include:

- **SIP Trunking**
- **Telephony Cards for analog, digital, and BRI PSTN connectivity**
- **Asterisk IP Phones**
- **VoIP Gateways**
- **Asterisk Training**

### Digium is the Gold Standard in Asterisk Hardware

Digium is the leading manufacturer of analog and digital interface cards, voice compression modules, redundancy solutions, and IP media gateways for use with the Asterisk open source communications engine. Digium is also the developer and maintainer of Asterisk. Asterisk is the free and open source communications engine that AsteriskNOW is built on.

***When you purchase Digium products, not only are you getting the best products in the industry, you are also contributing and supporting the free Asterisk open source project! Thanks for your support!***

## Upgrading AsteriskNOW

Upgrades are only available from Version 3.0.1 and later.

*Versions prior to 3.0 are NOT available for upgrade.*

## Upgrading AsteriskNOW Version 3.0.1 to 5.211.65

Follow this link for instructions to [upgrade AsteriskNOW Version 3.0.1 to Version 5.211.65](#).



## Upgrading AsteriskNOW Version 5.211.65

Follow this link for instructions to [upgrade AsteriskNOW Version 5.211.65](#)

## Installing Asterisk From Source

One popular option for installing Asterisk is to download the source code and compile it yourself. While this isn't as easy as using package management or using an Asterisk-based Linux distribution, it does let you decide how Asterisk gets built, and which Asterisk modules are built.

In this section, you'll learn how to download and compile the Asterisk source code, and get Asterisk installed.

- [What to Download?](#)
- [Untarring the Source](#)
- [Building and Installing DAHDI](#)
- [Building and Installing LibPRI](#)
- [Building and Installing pjproject](#)
- [Checking Asterisk Requirements](#)
- [Using Menuselect to Select Asterisk Options](#)
- [Building and Installing Asterisk](#)
- [Installing Sample Files](#)
- [Installing Initialization Scripts](#)
- [Validating Your Installation](#)

# What to Download?

## Asterisk

Downloads of Asterisk are available at <http://downloads.asterisk.org/pub/telephony/asterisk/>. The currently supported versions of Asterisk will each have a symbolic link to their related release on this server, named **asterisk-(version)-current.tar.gz**. All releases ever made for the Asterisk project are available at <http://downloads.asterisk.org/pub/telephony/asterisk/releases/>.

The currently supported versions of Asterisk are documented on the [Asterisk Versions](#) page. It is highly recommended that you install one of the currently supported versions, as these versions continue to receive bug and security fixes.



### Which version should I install?

- If you want a rock solid communications framework, choose the latest **Long Term Support (LTS)** release.
- If you want the latest cool features and capabilities, choose the latest release of Asterisk. If that is a **Standard** release, note that these releases may have larger changes made in them than LTS releases.

Unless otherwise noted, for the purposes of this section we will assume that Asterisk 11 is being installed.

Review Asterisk's [System Requirements](#) in order to determine what needs to be installed for the version of Asterisk you are installing. While Asterisk will look for any missing system requirements during compilation, it's often best to install these prior to configuring and compiling Asterisk.

Asterisk does come with a script, **install\_prereq**, to aid in this process. If you'd like to use this script, download Asterisk first, then see [Checking Asterisk Requirements](#) for instructions on using this script to install prerequisites for your version of Asterisk.

### On this Page

- [Asterisk](#)
  - [Downloading Asterisk](#)
- [Other Projects](#)
  - [libpri](#)
  - [DAHDI](#)
- [Download Locations](#)

## Downloading Asterisk

Browse to <http://downloads.asterisk.org/pub/telephony/asterisk>, select **asterisk-11-current.tar.gz**, and save the file on your file system.

You can also get the latest releases from the downloads page on [asterisk.org](http://asterisk.org).

Alternatively, you can use `wget` to retrieve the latest release:

```
[root@server:/usr/local/src]# wget http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-11-current.tar.gz

--2014-04-15 15:36:45-- http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-11-current.tar.gz
Resolving downloads.asterisk.org (downloads.asterisk.org)... 76.164.171.238
Connecting to downloads.asterisk.org (downloads.asterisk.org)|76.164.171.238|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 34794750 (33M) [application/x-gzip]
Saving to: `asterisk-11-current.tar.gz'

100%[=====] 34,794,750 6.82M/s in 4.6s

2014-04-15 15:36:49 (7.16 MB/s) - `asterisk-11-current.tar.gz' saved [34794750/34794750]
```

## Other Projects

### libpri

The **libpri** library allows Asterisk to communicate with ISDN connections. You'll only need this if you are going to use DAHDI with ISDN interface hardware (such as T1/E1/J1/BR1 cards).

### DAHDI

The **DAHDI** library allows Asterisk to communicate with analog and digital telephones and telephone lines, including connections to the Public Switched Telephone Network, or PSTN.

DAHDI stands for Digium Asterisk Hardware Device Interface, and is a set of drivers and utilities for a number of analog and digital telephony cards, such as those manufactured by Digium. The DAHDI drivers are independent of Asterisk, and can be used by other applications. DAHDI was previously called ZapTel, as it evolved from the Zapata Telephony Project.

The DAHDI code can be downloaded as individual pieces (**dahdi-linux** for the DAHDI drivers, and **dahdi-tools** for the DAHDI utilities). They can also be downloaded as a complete package called **dahdi-linux-complete**, which contains both the Linux drivers and the utilities.

You will only need to install DAHDI if you are going to utilize DAHDI compatible analog or digital telephony interface boards.



**Why is DAHDI split into different pieces?**

DAHDI has been split into two pieces (the Linux drivers and the tools) as third parties have begun porting the DAHDI drivers to other operating systems, such as FreeBSD. Eventually, we may have dahdi-linux, dahdi-freebsd, and so on.

## Download Locations

Project	Location
Asterisk	<a href="http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-11-current.tar.gz">http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-11-current.tar.gz</a>
libpri	<a href="http://downloads.asterisk.org/pub/telephony/libpri/libpri-1.4-current.tar.gz">http://downloads.asterisk.org/pub/telephony/libpri/libpri-1.4-current.tar.gz</a>
dahdi-linux	<a href="http://downloads.asterisk.org/pub/telephony/dahdi-linux/dahdi-linux-current.tar.gz">http://downloads.asterisk.org/pub/telephony/dahdi-linux/dahdi-linux-current.tar.gz</a>
dahdi-tools	<a href="http://downloads.asterisk.org/pub/telephony/dahdi-tools/dahdi-tools-current.tar.gz">http://downloads.asterisk.org/pub/telephony/dahdi-tools/dahdi-tools-current.tar.gz</a>
dahdi-complete	<a href="http://downloads.asterisk.org/pub/telephony/dahdi-linux-complete/dahdi-linux-complete-current.tar.gz">http://downloads.asterisk.org/pub/telephony/dahdi-linux-complete/dahdi-linux-complete-current.tar.gz</a>

## Untarring the Source

When you download the source for libpri, DAHDI, and Asterisk you'll typically end up with files with a .tar.gz or .tgz file extension. These files are affectionately known as tarballs. The name comes from the tar Unix utility, which stands for tape archive. A tarball is a collection of other files combined into a single file for easy copying, and then often compressed with a utility such as GZip.

To extract the source code from the tarballs, we'll use the tar command. The commands below assume that you've downloaded the tarballs for libpri, DAHDI, and Asterisk to the /usr/local/src directory on a Linux machine. (You'll probably need to be logged in as the root user to be able to write to that directory.) We're also going to assume that you'll replace the letters X, Y, and Z with the actual version numbers from the tarballs you downloaded. Also please note that the command prompt may be slightly different on your system than what we show here. Don't worry, the commands should work just the same.

First, we'll change to the directory where we downloaded the source code:

```
[root@server ~]# cd /usr/local/src
```

Next, let's extract the source code from each tarball using the tar command. The -zxvf parameters to the tar command tell it what we want to do with the file. The z option tells the system to unzip the file before continuing, the x option tells it to extract the files from the tarball, the v option tells it to be verbose (write out the name of every file as it's being extracted, and the f option tells the tar command that we're extracting the file from a tarball file, and not from a tape.

```
[root@server src]# tar -zxvf libpri-1.X.Y.tar.gz
[root@server src]# tar -zxvf dahdi-linux-complete-2.X.Y+2.X.Y.tar.gz
[root@server src]# tar -zxvf asterisk-11-current.tar.gz
```

You should now notice that a new sub-directory was created for each of the tarballs, each containing the extracted files from the corresponding tarball. We can now compile and install each of the components.

# Building and Installing DAHDI

## Overview

Let's install DAHDI!

On Linux, we will use the **DAHDI-linux-complete** tarball, which contains the DAHDI Linux drivers, DAHDI tools, and board firmware files. Again, we're assuming that you've untarred the tarball in the `/usr/local/src` directory, and that you'll replace X and Y with the appropriate version numbers.

See [What to Download?](#) for more information on downloading the DAHDI tarballs.



### Install DAHDI before libpri

libpri 1.4.13 and later source code depends on DAHDI include files. So, one must install DAHDI before installing libpri.



### Don't need DAHDI?

If you are not integrating with any traditional telephony equipment and you are not planning on using the [MeetMe](#) dialplan application, then you do not have to install DAHDI or libpri in order to use Asterisk.

## On This Page

- [Overview](#)

Starting with DAHDI-Linux-complete version 2.8.0+2.8.0, all files necessary to install DAHDI are available in the complete tarball. Therefore, all you need to do to install DAHDI is:

```
[root@server src]# cd dahdi-linux-complete-2.X.Y+2.X.Y
[root@server dahdi-linux-complete-2.X.Y+2.X.Y]# make
[root@server dahdi-linux-complete-2.X.Y+2.X.Y]# make install
[root@server dahdi-linux-complete-2.X.Y+2.X.Y]# make config
```

## Building and Installing LibPRI



### Have you installed DAHDI?

Before you can build **libpri**, you'll need to [Build and Install DAHDI](#).

As in the other build and install sections, we'll assume that you'll replace the letters X, Y, and Z with the actual version numbers from the tarballs you downloaded.

```
[root@server src]# cd libpri-1.X.Y
```

This command changes directories to the **libpri** source directory.

```
[root@server libpri-1.X.Y]# make
```

This command compiles the **libpri** source code into a system library.

```
[root@server libpri-1.X.Y]# make install
```

This command installs the **libpri** library into the proper system library directory

# Building and Installing pjproject

## Overview

Asterisk 12 and later versions contain two SIP stacks: one, the original chan\_sip SIP channel driver that has been present in all previous releases of Asterisk, and a new SIP stack that is based on [pjproject](#). For more information on configuring the new SIP stack, see [Configuring res\\_pjsip](#).

Because earlier releases of pjproject cannot build shared object libraries, some changes were required in order to use it with Asterisk 12. As such, **Asterisk requires pjproject version 2.4 or later**. Alternatively, an Asterisk compatible version of pjproject is available on [github](#), or - depending on your Linux distribution - available as a package.

Earlier versions of pjproject downloaded from [www.pjsip.org](#) will **not** work with Asterisk 12.



**Asterisk 11** uses an embedded pjproject for the ICE, STUN and TURN libraries in its RTP engine for WebSockets support. Therefore you do not need to follow the instructions here for Asterisk 11.  
**Asterisk 12 and 13** dynamically link to pjproject.  
**Asterisk >= 13.8.0** can dynamically link to pjproject but also contains a bundled version of pjproject. See [Using the Bundled Version of pjproject](#)

### On this Page

- [Overview](#)
- [Building and Installing pjproject from Source](#)
  - [Downloading pjproject](#)
  - [Troubleshooting](#)
- [Uninstalling a Previous Version of pjproject](#)
- [Using the Bundled Version of pjproject](#)

## Building and Installing pjproject from Source

If you're using Asterisk 13.8.0 or greater, consider using the [Bundled Version of pjproject](#).



If you have previously installed a version of pjproject, you **must** remove that version of pjproject prior to building and installing the Asterisk 12 compatible version of pjproject. See [Uninstalling pjproject](#) for more information.

## Downloading pjproject

### Obtaining pjproject from Teluu:

1. Use `wget` to pull the latest version from [www.pjsip.org](#). Note that the instructions assume that this is 2.4.5; for the latest version, refer to [www.pjsip.org](#):

```
# wget http://www.pjsip.org/release/2.4.5/pjproject-2.4.5.tar.bz2
# tar -xjvf pjproject-2.4.5.tar.bz2
```

### Obtaining the latest pjproject from the svn repo:

1. Use `svn` to install the latest version from [www.pjsip.org](#).

```
# svn co http://svn.pjsip.org/repos/pjproject/trunk/ pjproject-trunk
```

### Obtaining (old asterisk) pjproject from the github repo:

1. If you do not have [git](#), install `git` on your local machine.



Downloading and installing `git` is beyond the scope of these instructions, but for Debian/Ubuntu systems, it should be as simple as:

```
apt-get install git
```

And for RedHat/CentOS systems:

```
yum install git
```

2. Checkout the Asterisk 12-compatible pjproject from the Asterisk [github repo](#):



```
# git clone https://github.com/asterisk/pjproject pjproject
```

And that's it!

## Building and Installing pjproject

The first step in building and installing pjproject is configuring it using **configure**. For Asterisk, this is arguably the **most** important step in this process. pjproject embeds a number of third party libraries which can conflict with versions of those libraries that may already be installed on your system. Asterisk **will not** use the embedded third party libraries within pjproject. As an example, if you are going to build the res\_srtp module in Asterisk, then you **must** specify "--with-external-srtp" when configuring pjproject to point to an external srtp library.

Additionally, Asterisk **REQUIRES** two or three options to be passed to **configure**:

- `--enable-shared` - Instruct pjproject to build shared object libraries. Asterisk will only use shared objects from pjproject.
- `--prefix` - Specify root install directory for pjproject. This will be dependent on your distribution of Linux; typically this is `/usr` for most systems. The default is `/usr/local`
- `--libdir` - Specify the installation location for object code libraries. This may need to be set to `/usr/lib64` for some 64-bit systems such as CentOS.



Failure to build Asterisk with shared pjproject object libraries **WILL** result in seemingly random crashes. For Asterisk to work properly with pjproject, pjproject **MUST** be built with shared object libraries.

## Compiler DEFINES

- Users who expect to deal with Contact URIs longer than 256 characters or hostnames longer than 128 characters should set `PJSIP_MAX_URL_SIZE` and `PJ_MAX_HOSTNAME` as appropriate.
- IPv6 support in pjproject is, by default, disabled. To enable it, set `PJ_HAS_IPV6` to 1.
- The default configuration of pjproject enables "assert" functions which can cause Asterisk to crash unexpectedly. To disable the asserts, set `NDEBUG` to 1.
- The default number of TCP/TLS incoming connections allowed is 64. If you plan on having more than that you'll need to set `PJ_IOQUEUE_MAX_HANDLES` to the new limit.

With the exception of `PJ_IOQUEUE_MAX_HANDLES`, the options can be set in `CFLAGS` and passed to `configure` as follows: `./configure CFLAGS="-DNDEBUG=1 -DPJ_HAS_IPV6=1"`, etc. A better way is to create or edit the `pjlib/include/pj/config_site.h` file and set them all there. Here's a reasonable starting point that also includes some performance tunings...

## pjlib/include/pj/config\_site.h

```
#define NDEBUG 1
#define PJ_HAS_IPV6 1
#define PJ_MAX_HOSTNAME 256
#define PJSIP_MAX_URL_SIZE 512

/* The upper limit on MAX_HANDLES is determined by
 * the value of FD_SETSIZE on your system. For Linux
 * this is usually 1024. The following code sets it
 * to whatever FD_SETSIZE is or you can set it to a
 * specific number yourself. pjproject will not
 * compile if you set it to greater than FD_SETSIZE.
 */
#include <sys/select.h>
#define PJ_IOQUEUE_MAX_HANDLES (FD_SETSIZE)

/* Set for maximum server performance.
 * In tests, setting these parameters reduced
 * CPU load by approximately 25% for the same number
 * of calls per second. Your results will vary,
 * of course.
 */
#define PJ_SCANNER_USE_BITWISE 0
#define PJ_OS_HAS_CHECK_STACK 0
#define PJ_LOG_MAX_LEVEL 3
#define PJ_ENABLE_EXTRA_CHECK 0
#define PJSIP_MAX_TSX_COUNT ((64*1024)-1)
#define PJSIP_MAX_DIALOG_COUNT ((64*1024)-1)
#define PJSIP_UDP_SO_SNDBUF_SIZE (512*1024)
#define PJSIP_UDP_SO_RCVBUF_SIZE (512*1024)
#define PJ_DEBUG 0
#define PJSIP_SAFE_MODULE 0
#define PJ_HAS_STRICMP_ALNUM 0
#define PJ_HASH_USE_OWN_TOLOWER 1
/*
 * It is imperative that PJSIP_UNESCAPE_IN_PLACE remain 0 or undefined.
 * Enabling it will result in SEGFaults when URIs containing escape sequences are
 * encountered.
 */
#define PJSIP_UNESCAPE_IN_PLACE 0
#undef PJ_TODO
#define PJ_TODO(x)
```

Other common **configure** options needed for pjproject are listed below:

Library	Configure option	Notes
libspeex shared objects	--with-external-speex	Make sure that the library development headers are accessible from pjproject. The CFLAGS and LDFLAGS environment variables may be used to set the include/lib paths.
libsrtp shared objects	--with-external-srtp	Make sure that the library development headers are accessible from pjproject. The CFLAGS and LDFLAGS environment variables may be used to set the include/lib paths.
GSM codec	--with-external-gsm	Make sure that the library development headers are accessible from pjproject. The CFLAGS and LDFLAGS environment variables may be used to set the include/lib paths.

Disable sound	--disable-sound	Let Asterisk perform sound manipulations.
Disable resampling	--disable-resample	Let Asterisk perform resample operations.
Disable video	--disable-video	Disable video support in pjproject's media libraries. This is not used by Asterisk.
Disable AMR	--disable-opencore-amr	Disable AMR codec support. This is not used by Asterisk

These are some of the more common options used to disable third party libraries in pjproject. However, other options may be needed depending on your system - see `configure --help` for a full list of configure options you can pass to pjproject.

- Now that you understand the pjproject configure options available, change directories to the pjproject source directory:

```
# cd pjproject
```

- In the pjproject source directory, run the configure script with the options needed for your system:

```
# ./configure --prefix=/usr --enable-shared --disable-sound --disable-resample --disable-video --disable-opencore-amr
CFLAGS='-O2 -DNDEBUG'
```

A few recommended options are shown. That includes setting a couple important CFLAGS, -O2 for common optimizations and -DNDEBUG to disable debugging code and assertions.

- Build pjproject:

```
# make dep
# make
```

- Install pjproject

```
# make install
```

- Update shared library links.

```
# ldconfig
```

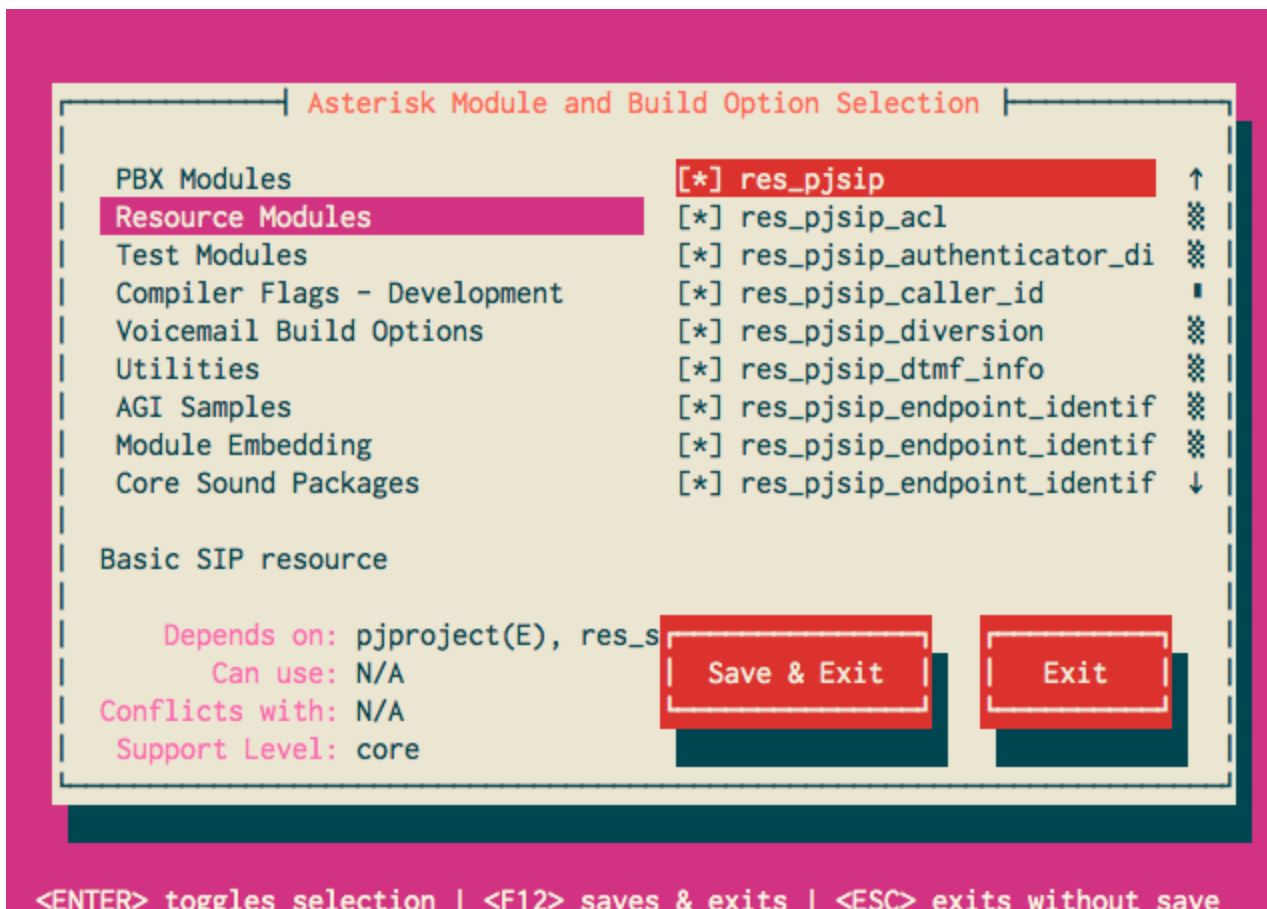
- Verify that pjproject has been installed in the target location by looking for, and finding the various pjproject modules:

```
# ldconfig -p | grep pj
libpjsua.so (libc6,x86-64) => /usr/lib/libpjsua.so
libpjsip.so (libc6,x86-64) => /usr/lib/libpjsip.so
libpjsip-ua.so (libc6,x86-64) => /usr/lib/libpjsip-ua.so
libpjsip-simple.so (libc6,x86-64) => /usr/lib/libpjsip-simple.so
libpjnath.so (libc6,x86-64) => /usr/lib/libpjnath.so
libpjmedia.so (libc6,x86-64) => /usr/lib/libpjmedia.so
libpjmedia-videodev.so (libc6,x86-64) => /usr/lib/libpjmedia-videodev.so
libpjmedia-codec.so (libc6,x86-64) => /usr/lib/libpjmedia-codec.so
libpjmedia-audiodev.so (libc6,x86-64) => /usr/lib/libpjmedia-audiodev.so
libpjlib-util.so (libc6,x86-64) => /usr/lib/libpjlib-util.so
libpj.so (libc6,x86-64) => /usr/lib/libpj.so
```

- Finally, verify that Asterisk detects the pjproject libraries. In your Asterisk source directory:

```
# ./configure
# make menuselect
```

- Browse to the **Resource Modules** category and verify that the `res_pjsip` modules are enabled:



9. You're all done! Now, build and install Asterisk as your normally would.

### Troubleshooting

First, if you're using Asterisk 13.8.0 or greater, consider switching to the [Bundled Version of pjproject](#)

#### ***Asterisk fails to detect pjproject libraries***

After building and installing pjproject, Asterisk fails to detect any of the libraries - the various res\_pjsip components cannot be selected in Asterisk's menuselect

#### **Solution**

Verify that Asterisk's config.log shows the following:

```
configure:23029: checking for PJPROJECT
configure:23036: $PKG_CONFIG --exists --print-errors "libpjproject"
Package libpjproject was not found in the pkg-config search path.
Perhaps you should add the directory containing `libpjproject.pc'
to the PKG_CONFIG_PATH environment variable
No package 'libpjproject' found
```

1. Make sure you have pkg-config installed on your system.
2. pjproject will install the package config file in /usr/lib/pkgconfig. Some distributions, notably Fedora, will instead look for the library in /usr/lib64. Update your PKG\_CONFIG\_PATH environment variable with /usr/lib/pkgconfig and re-run Asterisk's configure script.

#### ***pjproject fails to build: errors related to opencore\_amr***

When building pjproject, errors about opencore\_amr are displayed, e.g.:

```

output/pjmedia-codec-x86_64-unknown-linux-gnu/opencv_core_amr.o:(.rodata+0x60): multiple definition of
`pjmedia_codec_amrnb_framelenbits'
output/pjmedia-codec-x86_64-unknown-linux-gnu/opencv_core_amr.o:(.rodata+0x60): first defined here
output/pjmedia-codec-x86_64-unknown-linux-gnu/opencv_core_amr.o:(.rodata+0x80): multiple definition of
`pjmedia_codec_amrnb_framelen'
output/pjmedia-codec-x86_64-unknown-linux-gnu/opencv_core_amr.o:(.rodata+0x80): first defined here
output/pjmedia-codec-x86_64-unknown-linux-gnu/opencv_core_amr.o:(.rodata+0x20): multiple definition of
`pjmedia_codec_amrwb_framelenbits'
output/pjmedia-codec-x86_64-unknown-linux-gnu/opencv_core_amr.o:(.rodata+0x20): first defined here
output/pjmedia-codec-x86_64-unknown-linux-gnu/opencv_core_amr.o:(.rodata+0x40): multiple definition of
`pjmedia_codec_amrwb_framelen'
output/pjmedia-codec-x86_64-unknown-linux-gnu/opencv_core_amr.o:(.rodata+0x40): first defined here
...

```

### Solution

You already have the AMR codec installed. Run `configure` with the `--disable-opencore-amr` option specified.

### ***pjproject fails to build: video linker errors***

When building `pjproject`, linker errors referring to various video methods are displayed, e.g.:

```

/home/mjordan/projects/pjproject/pjmedia/lib/libpjmedia-videodev.so: undefined reference to `pjmedia_format_init_video'
/home/mjordan/projects/pjproject/pjmedia/lib/libpjmedia.so: undefined reference to `pjmedia_video_format_mgr_instance'
/home/mjordan/projects/pjproject/pjmedia/lib/libpjmedia-videodev.so: undefined reference to
`pjmedia_format_get_video_format_detail'
/home/mjordan/projects/pjproject/pjmedia/lib/libpjmedia-videodev.so: undefined reference to `pjmedia_get_video_format_info'

```

### Solution

Run `configure` with either or both `--disable-video` or `--disable-v4l2`

### ***ldconfig fails to display pjproject libraries***

After building `pjproject`, the dump provided by `ldconfig -p` doesn't display any libraries.

### Solution

Run `ldconfig` to re-configure dynamic linker run-time bindings. This will need to be run with super user permissions.

### ***pjproject fails to build on Raspberry Pi***

`pjproject/Asterisk` fails to compile on your Raspberry Pi (raspbian) due to `pjproject` configure scripts not detecting endianness:

```

/usr/include/pj/config.h:243:6: error: #error Endianness must be declared for this processor
In file included from /usr/include/pj/types.h:33:0,
                 from /usr/include/pjsip/sip_config.h:27,
                 from /usr/include/pjsip/sip_types.h:34,
                 from /usr/include/pjsip.h:24,
                 from conftest.c:290:
/usr/include/pj/config.h:1161:4: error: #error "PJ_IS_LITTLE_ENDIAN is not defined!"
/usr/include/pj/config.h:1165:4: error: #error "PJ_IS_BIG_ENDIAN is not defined!"

```

### Solution

1. Edit `/usr/include/pj/config.h` (using the editor of your choice)
2. Replace this code:

```

/*
 * ARM, bi-endian, so raise error if endianness is not configured
 */
# undef PJ_M_ARMV4
# define PJ_M_ARMV4          1
# define PJ_M_NAME           "armv4"
# define PJ_HAS_PENTIUM      0
# if !PJ_IS_LITTLE_ENDIAN && !PJ_IS_BIG_ENDIAN
#     error Endianness must be declared for this processor
# endif

```

With this:

```

/*
 * ARM, bi-endian, so raise error if endianness is not configured
 */
# undef PJ_M_ARMV4
# define PJ_M_ARMV4 1
# define PJ_M_NAME "armv4"
# define PJ_HAS_PENTIUM 0
# define PJ_IS_LITTLE_ENDIAN 1
# define PJ_IS_BIG_ENDIAN 0

```

Then recompile. This workaround was taken from issue [ASTERISK-23315](#).

## Uninstalling a Previous Version of pjproject

Typically, other versions of pjproject will be installed as static libraries. These libraries are not compatible with Asterisk and can confuse the build process for Asterisk 12. As such, any static libraries must be removed prior to installing the compatible version of pjproject.

pjproject provides an `uninstall` make target that will remove previous installations. It can be called from the pjproject source directory like:

```
# make uninstall
```

If you don't have an "uninstall" make target, you may need to fetch and merge the latest pjproject from <https://github.com/asterisk/pjproject>

Alternatively, the following should also remove all previously installed static libraries:

```
# rm -f /usr/lib/libpj*.a /usr/lib/libmilenage*.a /usr/lib/pkgconfig/libpjproject.pc
```

Finally, you will need to update shared library links:

```
# ldconfig
```

If you want to run a sanity check, you can verify that pjproject has been uninstalled by ensuring no pjproject modules remain on the system:

```
# ldconfig -p | grep pj
```

If running the above command yields no results, that's it! You have successfully uninstalled pjproject from your system.

## Using the Bundled Version of pjproject

Beginning with Asterisk 13.8.0, a stable version of pjproject is included in Asterisk's `./third-party` directory.

### Why would you want to do this? Several reasons:

- **Predictability:** When built with the bundled pjproject, you're always certain of the version you're running against, no matter where it's installed.
- **Scalability:** The default pjproject configuration is optimized for client applications. The bundled version's configuration is optimized for server use.
- **Usability:** Several feature patches, which have been submitted upstream to pjproject but not yet released, have been included in the bundled version.
- **Safety:** If a security or critical issue is identified in pjproject, it can be patched and made available with a new release of Asterisk instead of having to wait for a new release of pjproject.
- **Maintainability:** You don't need to build and install separate packages.
- **Supportability:** When asking others for help, there's no question about which version of pjproject you're using and what options it was compiled with.
- **Compatibility:** This is especially important from a development perspective because it means we can be sure that new pjproject APIs that have been introduced or old ones that have been deprecated, are handled and tested appropriately in Asterisk.
- **Reliability:** You can be sure that Asterisk was tested against the bundled version.

### Usage:

First, run `./contrib/scripts/install_prereq`. Building the bundled pjproject requires the python development libraries which `install_prereq` has already installed. All you have to do now is add the `--with-pjproject-bundled` option to your Asterisk `./configure` command line and remove any other `--with-pjproject` option you may have specified.

```
# cd /path/asterisk-source-dir
# ./configure --with-pjproject-bundled
# make && make install
```

The configure and make processes will download the correct version of pjproject, patch it, configure it, build it, and finally link Asterisk to it statically. No changes in runtime configuration are required. You can leave your system-installed version of pjproject in place if needed; once compiled with the `--with-pjproject-bundled` option, Asterisk will ignore any other installed versions of pjproject.

# Checking Asterisk Requirements

## Configuring Asterisk

Now it's time to compile and install Asterisk. Let's change to the directory which contains the Asterisk source code.


```
[root@server]# cd /usr/local/src/asterisk-11.X.Y
```

Next, we'll run a command called **.configure**, which will perform a number of checks on the operating system, and get the Asterisk code ready to compile on this particular server.

```
[root@server asterisk-11.X.Y]# ./configure
```

This will run for a couple of minutes, and warn you of any missing system libraries or other dependencies. Unless you've installed all of the [System Requirements](#) for your version of Asterisk, the **configure** script is likely to fail. If that happens, resolve the missing dependency manually, or use the `install_prereq` script to resolve all of the dependencies on your system.

Once a dependency is resolved, run **configure** again to make sure the missing dependency is fixed.

 If you have many missing dependencies, you may find yourself running **configure** a lot. If that is the case, you'll do yourself a favour by checking the [System Requirements](#) or installing all dependencies via the `install_prereq` script.

**On this Page**


- [Configuring Asterisk](#)
- [Using install\\_prereq](#)

Upon successful completion of **.configure**, you should see a message that looks similar to the one shown below. (Obviously, your host CPU type may be different than the below.)

```

      .$$$$$$$$$$$$$$$$$.
      .$7$7..          .7$$7:.
      .$7$7..          .7$$7:.
      .$$:.            ,7$.7
      .7$.          7$$$$          .$$77
      ..$$..         $$$$$          .$$$7
      .7$ .?.       $$$$$ .?.       7$$$$.
      $.$.          .$$$7. $$$7 .7$$$ . $$$
      .777.         .$$$$$$$7$$$$7$$$$7. $$$
      $$$~         .7$$$$$$$$$$$$7.    $$$
      .$$7         .7$$$$$$$7:         ?$$$
      $$$         ?7$$$$$$$$$$$I       .$$$7
      $$$         .7$$$$$$$$$$$$$$$    :$$$
      $$$         $$$$$$7$$$$$$$$$$$    .$$$
      $$$         $$$ 7$$$$7 .$$$      .$$$
      $$$         $$$7          .$$$
      7$$$$7       7$$$$          7$$$
      $$$$$        $$$          $$$
      $$$7.        $$ (TM)
      $$$$$$.      .7$$$$$ $
      $$$$$$$$$$7$$$$$$$$$. $$$$$
      $$$$$$$$$$$$$$.

configure: Package configured for:&nbsp;
configure: OS type &nbsp;: linux-gnu
configure: Host CPU : x86_64
configure: build-cpu:vendor:os: x86_64 : unknown : linux-gnu :
configure: host-cpu:vendor:os: x86_64 : unknown : linux-gnu :
```

 **Cached Data**  
The **.configure** command caches certain data to speed things up if it's invoked multiple times. To clear all the cached data, you can use the following command to completely clear out any cached data from the Asterisk build system.

```
[root@server asterisk-11.X.Y]# make distclean
```

You can then re-run **.configure**.

## Using install\_prereq

The `install_prereq` script is included with every release of Asterisk in the `contrib/scripts` subdirectory. The script has the following options:

- **test** - print only the libraries to be installed.



- **install** - install package dependencies only. Depending on your distribution of Linux, version of Asterisk, and capabilities you wish to use, this may be sufficient.
- **install-unpackaged** - install dependencies that don't have packages but only have tarballs. You may need these dependencies for certain capabilities in Asterisk.



You should always use your operating system's package management tools to ensure that your system is running the latest software **before** running `install_prereq`. Ubuntu 14's `libsnmp-dev` package, for instance, has an issue where it will attempt to remove critical system packages if the system isn't updated before an attempt is made to install that package.

```
[root@server asterisk-11.X.Y]# cd contrib/scripts
[root@server asterisk-11.X.Y/contrib/scripts]# ./install_prereq install
[root@server asterisk-11.X.Y/contrib/scripts]# ./install_prereq install-unpackaged
```

# Using Menuselect to Select Asterisk Options

## Using Menuselect

The next step in the build process is to tell Asterisk which [modules](#) to compile and install, as well as set various compiler options. These settings are all controlled via a menu-driven system called **Menuselect**. To access the Menuselect system, type:

```
[root@server asterisk-11.X.Y]# make menuselect
```



### Terminal Window

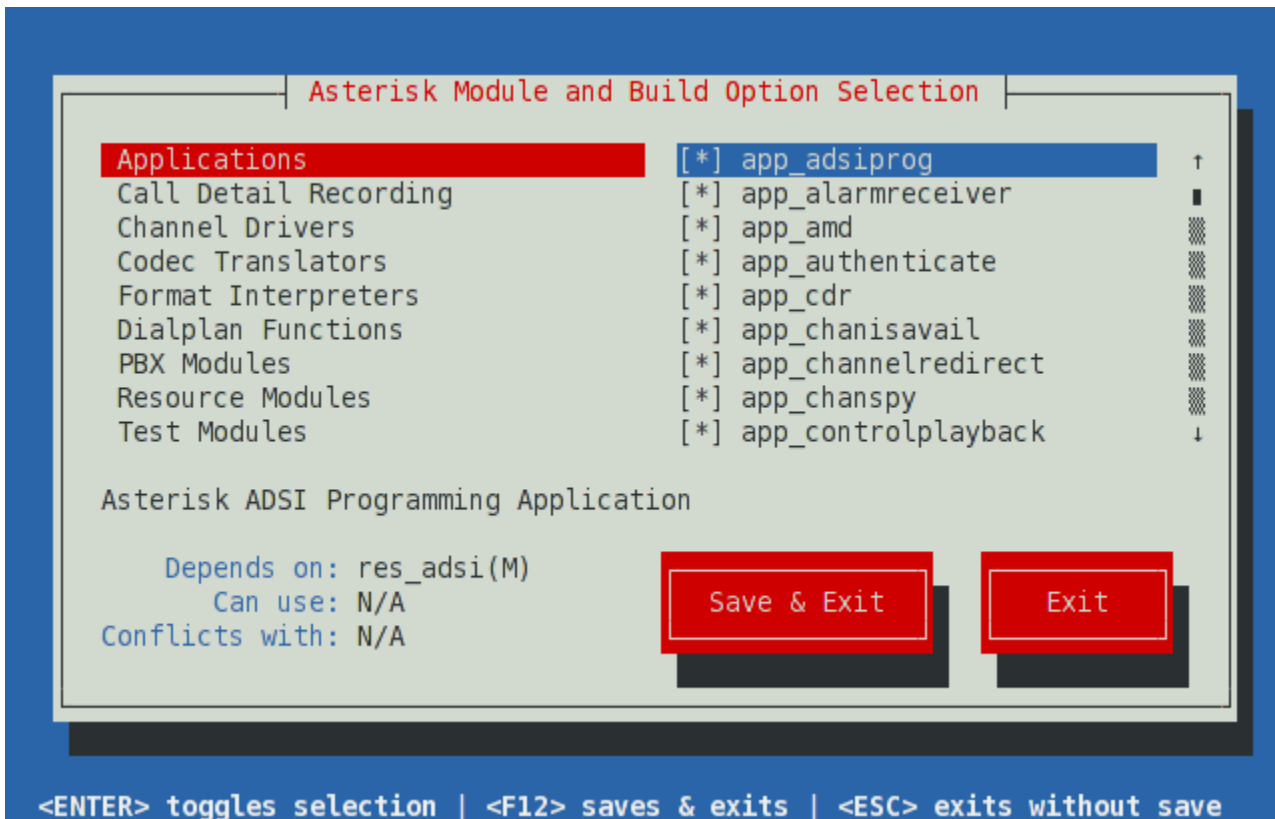
Your terminal window size must be at least eighty characters wide and twenty-seven lines high, or Menuselect will not work. Instead, you'll get an error message stating

```
Terminal must be at least 80 x 27.
```

The **Menuselect** menu should look like the screen-shot below. On the left-hand side, you have a list of categories, such as **Applications**, **Channel Drivers**, and **PBX Modules**. On the right-hand side, you'll see a list of modules that correspond with the select category. At the bottom of the screen you'll see two buttons. You can use the **Tab** key to cycle between the various sections, and press the **Enter** key to select or unselect a particular module. If you see **[\*]** next to a module name, it signifies that the module has been selected. If you see **\*XXX** next to a module name, it signifies that the select module cannot be built, as one of its dependencies is missing. In that case, you can look at the bottom of the screen for the line labeled **Depends upon**: for a description of the missing dependency.

### On this Page

- Using Menuselect
  - Module Support Levels
- Menuselect Categories
- Controlling Menuselect
  - Listing Options
  - Enabling an Option
  - Disabling an Option
  - Enabling a Category




When you're first learning your way around Asterisk on a test system, you'll probably want to stick with the default settings in **Menuselect**. If you're building a production system, however, you may not wish to build all of the various modules, and instead only build the modules that your system is using.

When you are finished selecting the modules and options you'd like in **Menuselect**, press **F12** to save and exit, or highlight the **Save and Exit** button and press enter.

## Module Support Levels

**Menuselect** will also show the support level for a selected module or build option. The support level will always be one of `core`, `extended`, or `deprecate`. For more information on these support levels, see [Asterisk Module Support States](#).

## Menuselect Categories

Category	Description
<b>Add-ons</b>	Modules that link with libraries that have licensing restrictions beyond what is allowed via the GPLv2 and Asterisk's dual licensing model. See <a href="#">README-addons.txt</a> , delivered with Asterisk, for more information.
<b>Applications</b>	Modules that provide call functionality to the system. An application might answer a call, play a sound prompt, hang up a call, and so forth.
<b>Bridging Modules</b>	Modules that provide various bridge mixing technologies and other bridge related functionality.
<b>Call Detail Recording</b>	Modules that provide Call Detail Record (CDR) drivers for various permanent storage backends.
<b>Channel Event Logging</b>	Modules that provide Channel Event Logging (CEL) drivers for various permanent storage backends.
<b>Channel Drivers</b>	Modules that provide communications with devices outside of Asterisk, and translate that particular signalling or protocol to the core.
<b>Codec Translators</b>	Modules that provide encoding/decoding for audio or video. Typically codecs are used to encode media so that it takes less bandwidth.
<b>Format Interpreters</b>	Modules used to save media to disk in a particular file format, and to convert those files back to media streams on the network.
<b>Dialplan Functions</b>	Modules that are used to retrieve or set various settings on a call. A function might be used to set the Caller ID on an outbound call, for example.
<b>PBX Modules</b>	Modules that implement dialplan functionality or enhancements.
<b>Resource Modules</b>	Modules that provide additional resources to Asterisk. This can include music on hold, calendar integration, database integration, various protocol stacks, etc.
<b>Test Modules</b>	Unit test modules. These are typically only available when Asterisk has: <ul style="list-style-type: none"> <li>• Been configured with the <code>--enable-dev-mode</code> setting</li> <li>• The <code>TEST_FRAMEWORK</code> compilation option has been selected in <b>Compiler Flags - Development</b></li> </ul>
<b>Compiler Flags - Development</b>	<p>Various compilation flags that alter Asterisk's behaviour. These flags are often useful in debugging Asterisk, or obtaining information for Asterisk developers.</p> <div style="border: 1px solid green; padding: 10px; margin: 10px 0;"> <p> <b>Easier Debugging of Asterisk Crashes</b>  As much as we may hate to admit it, Asterisk <i>may</i> sometimes have problems.</p> <p>If you're finding that Asterisk is crashing on you, there's are settings under <b>Compiler Flags - Development</b> that are critical for developers attempting to assist you. For detailed instructions on enabling these settings, see <a href="#">Getting a Backtrace</a>.</p> </div>
<b>Voicemail Build Options</b>	Compilation flags that enable different Voicemail (via <code>app_voicemail</code> ) storage backends.
<b>Utilities</b>	Various utilities for Asterisk. These include Asterisk Database upgrade utilities, Asterisk monitoring utilities, and other potentially useful tools.
<b>AGI Samples</b>	Sample AGI applications.
<b>Module Embedding</b>	Compilation flags to enable embedding of Asterisk dynamic modules into the Asterisk binary.
<b>Core Sound Packages</b>	Core sounds used by Asterisk. Different sound formats can be selected in this menu; when Asterisk is installed, these sounds will be downloaded and installed.

<b>Music On Hold File Packages</b>	Sample Music on Hold media used by Asterisk. Different formats can be selected in this menu; when Asterisk is installed, the various media samples will be downloaded and installed.
<b>Extras Sound Packages</b>	Extra sounds that can be used by Asterisk integrators. Different sound formats can be selected in this menu; when Asterisk is installed, these sounds will be downloaded and installed.

## Controlling Menuselect

Options in **Menuselect** can be controlled from the command line. **Menuselect** can be built without invoking the user interface via the `menuselect.makeopts` target:

```
[root@server asterisk-11.X.Y]# make menuselect.makeopts
```

Available options can be viewed using the `--help` command line parameter:

```
[root@server asterisk-11.X.Y]# menuselect/menuselect --help
```

Some of the more common options are shown below.



### Menuselect Output

Asterisk expects all **Menuselect** options to be written to the `menuselect.makeopts` file. When enabling/disabling **Menuselect** options via the command line, your output should typically be to that file.

## Listing Options

To list all options in **Menuselect**, use the `--list-options` command line parameter:

```
[root@server asterisk-11.X.Y]# menuselect/menuselect --list-options
```

To list only the categories, use the `--category-list` command line parameter:

```
[root@server asterisk-11.X.Y]# menuselect/menuselect --category-list
MENSELECT_ADDONS
MENSELECT_APPS
MENSELECT_BRIDGES
MENSELECT_CDR
MENSELECT_CEL
MENSELECT_CHANNELS
MENSELECT_CODECS
MENSELECT_FORMATS
MENSELECT_FUNCS
MENSELECT_PBX
MENSELECT_RES
MENSELECT_TESTS
MENSELECT_CFLAGS
MENSELECT_OPTS_app_voicemail
MENSELECT_UTILS
MENSELECT_AGIS
MENSELECT_EMBED
MENSELECT_CORE_SOUNDS
MENSELECT_MOH
MENSELECT_EXTRA_SOUNDS
```

To list the options in a category, use the `--list-category` command line parameter:

```
[root@server asterisk-11.X.Y]# menuselect/menuselect --list-category MENSELECT_OPTS_app_voicemail
+ FILE_STORAGE
- ODBC_STORAGE
- IMAP_STORAGE
```

## Enabling an Option

To enable an option in **Menuselect**, use the `--enable` command line parameter:

```
[root@server asterisk-11.X.Y]# menuselect/menuselect --enable IMAP_STORAGE menuselect.makeopts
```



### Chaining Options

Multiple options can be chained together:

```
[root@server asterisk-11.X.Y]# menuselect/menuselect --enable app_voicemail --enable IMAP_STORAGE menuselect.makeopts
```

### Disabling an Option

To disable an option in **Menuselect**, use the `--disable` command line parameter:

```
[root@server asterisk-11.X.Y]# menuselect/menuselect --disable app_voicemail menuselect.makeopts
```

### Enabling a Category

An entire category can be enabled in **Menuselect** using the `--enable-category` command line parameter:

```
[root@server asterisk-11.X.Y]# menuselect/menuselect --enable-category MENUSELECT_ADDONS menuselect.makeopts
```

# Building and Installing Asterisk

## Build and Install Instructions

Now we can compile and install Asterisk. To compile Asterisk, simply type `make` at the Linux command line.

```
[root@server asterisk-11.X.Y]# make
```

The compiling step will take several minutes, and you'll see the various file names scroll by as they are being compiled. Once Asterisk has finished compiling, you'll see a message that looks like:

```
+----- Asterisk Build Complete -----+
+ Asterisk has successfully been built, and +
+ can be installed by running:           +
+                                       +
+             make install                +
+----- Asterisk Build Complete -----+
```

### On this Page

- [Build and Install Instructions](#)
- [Advanced Build and Install Options](#)
  - [Customizing the Build/Installation](#)
    - [Passing compilation and linkage flags to gcc](#)
    - [Debugging compilation](#)
    - [Building for non-native architectures](#)
    - [Installing to a custom directory](#)
  - [Other Make Targets](#)

As the message above suggests, our next step is to install the compiled Asterisk program and modules. To do this, use the **make install** command.

```
[root@server asterisk-11.X.Y]# make install
```

When finished, Asterisk will display the following warning:

```
+---- Asterisk Installation Complete -----+
+                                       +
+   YOU MUST READ THE SECURITY DOCUMENT   +
+                                       +
+ Asterisk has successfully been installed. +
+ If you would like to install the sample +
+ configuration files (overwriting any    +
+ existing config files), run:           +
+                                       +
+             make samples                 +
+----- Asterisk Installation Complete -----+
```



### Security Precautions

As the message above suggests, we very strongly recommend that you read the security documentation before continuing with your Asterisk installation. Failure to read and follow the security documentation can leave your system vulnerable to a number of security issues, including toll fraud.

## Advanced Build and Install Options

### Customizing the Build/Installation

In some environments, it may be necessary or useful to modify parts of the build or installation process. Some common scenarios are listed here

#### ***Passing compilation and linkage flags to gcc***

Specific flags can be passed to `gcc` when Asterisk is configured, using the `CFLAGS` and `LDFLAGS` environment variables:

```
[root@server asterisk-11.X.Y]# ./configure CFLAGS=-pg LDFLAGS=-pg
```

#### ***Debugging compilation***

To see all of the flags passed to `gcc`, build using the `NOISY_BUILD` setting set to `YES`:

```
[root@server asterisk-11.X.Y]# make NOISY_BUILD=yes
```

## Building for non-native architectures

Generally, Asterisk attempts to optimize itself for the machine on which it is built on. On some virtual machines with virtual CPU architectures, the defaults chosen by Asterisk's compilation options will cause Asterisk to build but fail to run. To disable native architecture support, disable the `BUILD_NATIVE` option in `menuselect`:

```
[root@server asterisk-11.X.Y]# menuselect/menuselect --disable BUILD_NATIVE menuselect.makeopts
[root@server asterisk-11.X.Y]# make
```



## Installing to a custom directory

While there are multiple ways to sandbox an instance of Asterisk, the preferred mechanism is to use the `--prefix` option with the `configure` script:

```
[root@server asterisk-11.X.Y]# ./configure --prefix=/usr/local/my_special_folder
```

Note that the default value for `prefix` is `/usr/local`.

## Other Make Targets

Target	Description
	Executing <code>make</code> with no target is equivalent to the <code>all</code> target.
<b>all</b>	Compiles everything everything selected through the <code>configure</code> and <code>menuselect</code> scripts.
<b>full</b>	This is equivalent to <code>make</code> or <code>make all</code> , save that it will perform a more thorough investigation of the source code for documentation. This is needed to generate <a href="#">AMI event documentation</a> . Note that your system must have Python in order for this make target to succeed. <div style="border: 1px solid #ffc107; padding: 5px;"> <b>Version Notice</b> This build target is only available in Asterisk 11 and later versions.</div>
<b>install</b>	Installs Asterisk, building Asterisk if it has not already been built. In general, this should be executed after Asterisk has successfully compiled.
<b>uninstall</b>	Removes Asterisk binaries, sounds, man pages, headers, modules and firmware builds from the system.
<b>uninstall-all</b>	Same as the <code>uninstall</code> target, but additionally removes configuration, spool directories and logs. All traces of Asterisk. <div style="border: 1px solid #ffc107; padding: 5px;"> As just noted, this will remove all Asterisk configuration from your system. Do not execute <code>uninstall-all</code> unless you are sure that is what you want to do.</div>
<b>clean</b>	Remove all files generated by <code>make</code> .
<b>dist-clean</b>	Remove pretty much all files generated by <code>make</code> and <code>configure</code> .
<b>samples</b>	Install all sample configuration files ( <code>.conf</code> files) to <code>/etc/asterisk/</code> . Overwrites existing config files.
<b>config</b>	Install init scripts (startup scripts) on your system.
<b>progdocs</b>	Uses <code>doxygen</code> to locally generate HTML development documentation from the source code. Generated in the <code>doc/</code> subdirectory of the source; see <code>doc/index.html</code> .

## Installing Sample Files



### **Asterisk Sample Configs: not a sample PBX configuration**

For many of the sample configuration files that **make samples** installs, the configuration contains more than just an example configuration. The sample configuration files historically were used predominately for documentation of available options. As such, they contain many examples of configuring Asterisk that may not be ideal for standard deployments.

While installing the sample configuration files may be a good starting point for some people, they should not be viewed as recommended configuration for an Asterisk system.

To install a set of sample configuration files for Asterisk, type:

```
[root@server asterisk-11.X.Y]# make samples
```

Any existing sample files which have been modified will be given a **.old** file extension. For example, if you had an existing file named **extensions.conf**, it would be renamed to **extensions.conf.old** and the sample dialplan would be installed as **extensions.conf**.



## Installing Initialization Scripts

Now that you have Asterisk compiled and installed, the last step is to install the initialization script, or `initscript`. This script starts Asterisk when your server starts, will monitor the Asterisk process in case anything *bad* happens to it, and can be used to stop or restart Asterisk as well. To install the `initscript`, use the **make config** command.

```
[root@server asterisk-11.X.Y]# make config
```

As your Asterisk system runs, it will generate logfiles. It is recommended to install the `logrotation` script in order to compress and rotate those files, to save disk space and to make searching them or cataloguing them easier. To do this, use the **make install-logrotate** command.

```
[root@server asterisk-11.X.Y]# make install-logrotate
```

## Validating Your Installation

Before continuing on, let's check a few things to make sure your system is in good working order. First, let's make sure the DAHDI drivers are loaded. You can use the **lsmod** under Linux to list all of the loaded kernel modules, and the **grep** command to filter the input and only show the modules that have **dahdi** in their name.

```
[root@server asterisk-11.X.Y]# lsmod | grep dahdi
```

If the command returns nothing, then DAHDI has not been started. Start DAHDI by running:

```
[root@server asterisk-11.X.Y]# /etc/init.d/dahdi start
```



### Different Methods for Starting Initscripts

Many Linux distributions have different methods for starting initscripts. On most Red Hat based distributions (such as Red Hat Enterprise Linux, Fedora, and CentOS) you can run:

```
[root@server asterisk-11.X.Y]# service dahdi start
```

Distributions based on Debian (such as Ubuntu) have a similar command, though it's not commonly used:

```
[root@server asterisk-11.X.Y]# invoke-rc.d dahdi start
```

If you have DAHDI running, the output of **lsmod | grep dahdi** should look something like the output below. (The exact details may be different, depending on which DAHDI modules have been built, and so forth.)

```
[root@server asterisk-11.X.Y]# lsmod | grep dahdi
dahdi_transcode  7928 1 wctc4xxp
dahdi_voicebus  40464 2 wctdm24xxp,wctel2xp
dahdi            196544 12 wctdm24xxp,wctel1xp,wctl1xp,wctel2xp,wct4xxp
crc_ccitt       2096 1 dahdi
```

Now that DAHDI is running, you can run **dahdi\_hardware** to list any DAHDI-compatible devices in your system. You can also run the **dahdi\_tool** utility to show the various DAHDI-compatible devices, and their current state.

To check if Asterisk is running, you can use the Asterisk initscript.

```
[root@server asterisk-11.X.Y]# /etc/init.d/asterisk status
asterisk is stopped
```

To start Asterisk, we'll use the initscript again, this time giving it the start action:

```
[root@server asterisk-11.X.Y]# /etc/init.d/asterisk start
Starting asterisk:
```

When Asterisk starts, it runs as a background service (or daemon), so you typically won't see any response on the command line. We can check the status of Asterisk and see that it's running using the command below. (The process identifier, or pid, will obviously be different on your system.)

```
[root@server asterisk-11.X.Y]# /etc/init.d/asterisk status
asterisk (pid 32117) is running...
```

And there you have it! You've compiled and installed Asterisk, DAHDI, and libpri from source code.

## Alternate Install Methods

If you already have a Linux system that you can dedicate to Asterisk, simply use the package manager in your operating system to install Asterisk, DAHDI, and libpri. Most modern Linux distributions such as Debian, Ubuntu, and Fedora have these packages in their repositories.

Linux distro maintained packages may be old, so watch out for that. There are no currently maintained official repositories for Asterisk packages.

## Asterisk Packages



There is currently no official repository for asterisk packages. Asterisk source code is distributed by Digium via tarballs and Git. Various community distributions of Asterisk may utilize packages provided and hosted by the distribution maintainer.

Read through [Installing Asterisk](#) for more detail on installing Asterisk via source.

## Historical Packaging Information



At one time, Asterisk packages were also available for Ubuntu. Currently, packages are not being made by the Asterisk project for this distribution. Information detailing the Ubuntu build environment has been moved onto this page for historical purposes.

### Prerequisites

All of Ubuntu's Code, Translations, Packages, Bugs, access control lists, team information, etc. live in [Launchpad](#). So for you to be able to contribute to bug discussions, upload packages, contribute code and translations, it's important that you:

- Create an [account](#) on launchpad.
- Create a [GPG key](#) and import it.
- Create a [SSH key](#) and import it.

### Create a Build Environment

#### **Install Ubuntu 10.04 (Lucid)**

Installing Ubuntu 10.04 (Lucid)

#### **Enable Backports**

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository "deb http://ca.archive.ubuntu.com/ubuntu/ $(lsb_release --short
--codename)-backports main universe"
```

#### **Upgrade Lucid to the latest release:**

```
$ sudo apt-get update
$ sudo apt-get dist-upgrade
$ sudo apt-get autoremove
$ sudo reboot
```

#### **Install required software**

```
$ sudo apt-get install build-essential pbuilder debian-archive-keyring ccache
```

#### **pbuilder**

```
$ sudo mkdir -p /var/cache/pbuilder/ccache
$ sudo mkdir -p /var/cache/pbuilder/hook.d
```

```
$ sudo vi /etc/pbuilder/pbuilderrc
```

#### **/etc/pbuilder/pbuilderrc**

```
export CCACHE_DIR="/var/cache/pbuilder/ccache"
export PATH="/usr/lib/ccache:${PATH}"
EXTRAPACKAGES="ccache"
BINDMOUNTS="${CCACHE_DIR}"

# Codenames for Debian suites according to their alias. Update these when
# needed.
UNSTABLE_CODENAME="sid"
TESTING_CODENAME="wheezy"
STABLE_CODENAME="squeeze"
```

```

OLDSTABLE_CODENAME="lenny"
STABLE_BACKPORTS_SUITE="$STABLE_CODENAME-backports"

# List of Debian suites.
DEBIAN_SUITES=($UNSTABLE_CODENAME $TESTING_CODENAME $STABLE_CODENAME $OLDSTABLE_CODENAME
    "unstable" "testing" "stable" "oldstable")

# List of Ubuntu suites. Update these when needed.
UBUNTU_SUITES=("oneiric" "natty" "maverick" "lucid")

# Mirrors to use. Update these to your preferred mirror.
DEBIAN_MIRROR="ftp.us.debian.org"
UBUNTU_MIRROR="mirrors.kernel.org"

# Optionally use the changelog of a package to determine the suite to use if
# none set.
if [ -z "${DIST}" ] && [ -r "debian/changelog" ]; then
    DIST=$(dpkg-parsechangelog | awk '/^Distribution: / {print $2}')
    # Use the unstable suite for certain suite values.
    if $(echo "experimental UNRELEASED" | grep -q $DIST); then
        DIST="$UNSTABLE_CODENAME"
    fi
fi

# Optionally set a default distribution if none is used. Note that you can set
# your own default (i.e. ${DIST:="unstable"}).
: ${DIST:="$(lsb_release --short --codename)"}

# Optionally change Debian release states in $DIST to their names.
case "$DIST" in
    unstable)
        DIST="$UNSTABLE_CODENAME"
        ;;
    testing)
        DIST="$TESTING_CODENAME"
        ;;
    stable)
        DIST="$STABLE_CODENAME"
        ;;
    oldstable)
        DIST="$OLDSTABLE_CODENAME"
        ;;
esac

# Optionally set the architecture to the host architecture if none set. Note
# that you can set your own default (i.e. ${ARCH:="i386"}).
: ${ARCH:="$(dpkg --print-architecture)"}

NAME="$DIST"
if [ -n "${ARCH}" ]; then
    NAME="$NAME-$ARCH"
    DEBOOTSTRAPOPTS=( "--arch" "$ARCH" "${DEBOOTSTRAPOPTS[@]}" )
fi

DEBBUILD_OPTS="-b"
if [ "${ARCH}" == "i386" ]; then
    DEBBUILD_OPTS="-B"
fi

```

```
BASETGZ="/var/cache/pbuilder/$NAME-base.tgz"
# Optionally, set BASEPATH (and not BASETGZ) if using cowbuilder
# BASEPATH="/var/cache/pbuilder/$NAME/base.cow/"
DISTRIBUTION="$DIST"
BUILDRRESULT="/var/cache/pbuilder/$NAME/result/"
APTCACHE="/var/cache/pbuilder/$NAME/aptcache/"
BUILDPPLACE="/var/cache/pbuilder/build/"

if $(echo ${DEBIAN_SUITES[@]} | grep -q $DIST); then
    # Debian configuration
    MIRRORSITE="http://$DEBIAN_MIRROR/debian/"
    COMPONENTS="main contrib non-free"
    DEBOOTSTRAPOPTS=("${DEBOOTSTRAPOPTS[@]}")
    "--keyring=/usr/share/keyrings/debian-archive-keyring.gpg")
elif $(echo ${UBUNTU_SUITES[@]} | grep -q $DIST); then
    # Ubuntu configuration
    MIRRORSITE="http://$UBUNTU_MIRROR/ubuntu/"
    COMPONENTS="main universe"
    DEBOOTSTRAPOPTS=("${DEBOOTSTRAPOPTS[@]}")
    "--keyring=/usr/share/keyrings/ubuntu-archive-keyring.gpg")
else
    echo "Unknown distribution: $DIST"
```

```
    exit 1
fi
```

### Debian

```
$ for x in unstable testing stable; do for y in i386 amd64; do sudo DIST=${x} ARCH=${y}
pbuilder create; done; done
```

### Ubuntu

```
$ for x in lucid maverick natty; do for y in i386 amd64; do sudo DIST=${x} ARCH=${y}
pbuilder create; done; done
```

### svn-buildpackage

```
$ vi ~/.svn-buildpackage.conf
```

```
svn-builder=debuild
svn-noautodch
```

### quilt

```
$ vi ~/.quiltrc
```

```
QUILT_PATCHES="debian/patches"

QUILT_PATCH_OPTS="--unified-reject-files"
QUILT_REFRESH_ARGS="-p ab --no-timestamps --no-index"
QUILT_DIFF_OPTS="--show-c-function"
QUILT_DIFF_ARGS="-p ab --no-timestamps --no-index --color=auto"
```

### devscripts

```
$ vi ~/.devscripts
```

```
DEBCHANGE_RELEASE_HEURISTIC=changelog
DEBCHANGE_MULTIMAIN_MERGE=yes
DEBCHANGE_MAINTTRAILER=no
DEBUILD_ROOTCMD=fakeroot
DEBUILD_LINTIAN=yes
DEBUILD_LINDA=yes
DEFAULT_DEBRELEASE_DEBS_DIR=../build-area/
USCAN_DESTDIR=../tarballs
```

### Create a GPG Key

<https://help.ubuntu.com/community/GnuPrivacyGuardHowto>

```
$ vi ~/.bashrc
```

```
export DEBFULLNAME='Paul Belanger'
export DEBEMAIL='pabelanger@digium.com'
export GPGKEY=8C3B0FA6
export EDITOR=vi
```



## See also

- [Ubuntu Packaging Guide](#)

## Updating an Ubuntu Package

### *New upstream release*

#### Checkout source

```
$ mkdir -p ~/digium
$ cd ~/digium
$ svn http://blah.org/svn/blah
```

#### Upstream tarball

```
$ uscan --verbose
```

#### Update the changelog file

```
$ dch -e
```

#### Update patches

```
$ while quilt push; do quilt refresh; done
```

#### Release package

```
$ dch -r
```

#### Build package source

```
$ svn-buildpackage -S
```

#### Compile package

```
$ dput ppa:pabelanger/testing ../build-area/*.changes
```

## rebuildd

### *Introduction*

### *Prerequisites*

[Creating a Build Environment](#)

### *Getting started*

```
sudo apt-get install rebuildd reprepro apache2
```

#### reprepro

```
$ sudo adduser --system --shell /bin/bash --gecos 'Reprepro Daemon' --group
--disabled-password reprepro
```

```
$ sudo su reprepro
```

```
$ cd ~
$ mkdir bin conf incoming
```

```
$ vi ~/conf/distributions
```

### distributions

```
Suite: lucid-proposed
Version: 10.04
Codename: lucid-proposed
Architectures: i386 amd64 source
Components: main
SignWith: yes
Log: logfile
  --changes ~/bin/build_sources
```

```
$ vi ~/conf/incoming
```

### incoming

```
Name: incoming
IncomingDir: incoming
Allow: lucid-proposed
Cleanup: on_deny on_error
TempDir: tmp
```

```
$ vi ~/conf/apache.conf
```

### apache.conf

```
Alias /debian /home/reprepo/
<Directory /home/reprepo>
  Options +Indexes
  AllowOverride None
  order allow,deny
  allow from all
</Directory>
```

```
$ vi ~/bin/build_sources
```

```
#!/bin/bash

action=$1
release=$2
package=$3
version=$4
changes_file=$5

# Only care about packages being added
if [ "$action" != "accepted" ]
then
    exit 0
fi

# Only care about source packages
echo $changes_file | grep -q _source.changes
if [ $? = 1 ]
then
    exit 0
fi

# Kick off the job
echo "$package $version 1 $release" | sudo rebuildd-job add
```

```
$ reprepro -V -b . createsymlinks
$ reprepro -V -b . processincoming incoming
```

```
$ exit
```

## rebuildd

```
$ sudo vi /etc/default/rebuildd
```

```
START_REBUILD=1
START_REBUILD_HTTPD=1
DIST="lucid"
```

## Also see

- [http://alioth.debian.org/scm/viewvc.php/\\*checkout\\*/mirrorer/docs/manual.html?revision=HEAD&root=mirrorer](http://alioth.debian.org/scm/viewvc.php/*checkout*/mirrorer/docs/manual.html?revision=HEAD&root=mirrorer)
- <http://inodes.org/2009/09/14/building-a-private-ppa-on-ubuntu/>

## Working with Source Packages

```
$ sudo apt-get build-dep asterisk
```

```
$ DEB_BUILD_OPTIONS="debug" apt-get -b source asterisk
```

# Installing Asterisk on Non-Linux Operating Systems

Sub-pages here should provide guidance for installation on Non-Linux operating systems. Contributions are welcome!

# Asterisk on (Open)Solaris

## Asterisk on Solaris 10 and OpenSolaris

On this page

- [Asterisk on Solaris 10 and OpenSolaris](#)
  - [Digium's Support Status](#)
  - [Build Notes](#)
    - [Prerequisites](#)
    - [LDAP dependencies](#)
    - [Makefile layouts](#)
    - [FAX support with SpanDSP](#)
  - [Gotchas](#)
    - [Runtime issues](#)
    - [Build issues](#)

### **Digium's Support Status**

According to the README file from 1.6.2: "Asterisk has also been 'ported' and reportedly runs properly on other operating systems as well, including Sun Solaris, Apple's Mac OS X, Cygwin, and the BSD variants." Digium's developers have also been doing a good job of addressing build and run-time issues encountered with Asterisk on Solaris.

### **Build Notes**

#### **Prerequisites**

The following packages are recommend for building Asterisk 1.6 and later on OpenSolaris:

- SUNWlibm (math library)
- gcc-dev (compiler and several dependencies)
- SUNWflexlex (GNU flex)
- SUNWggrp (GNU grep)
- SUNWgsed (GNU sed)
- SUNWdoxygen (optional; needed for "make progdocs")
- SUNWopenldap (optional; needed for res\_config\_ldap; see below)
- SUNWgnu-coreutils (optional; provides GNU install; see below)

Caution: installing SUNW gnu packages will change the default application run when the user types 'sed' and 'grep' from /usr/bin/sed to /usr/gnu/bin/sed. Just be aware of this change, as there are differences between the Sun and GNU versions of these utilities.

#### **LDAP dependencies**

Because OpenSolaris ships by default with Sun's LDAP libraries, you must install the SUNWopenldap package to provide OpenLDAP libraries. Because of namespace conflicts, the standard LDAP detection will not work.

There are two possible solutions:

1. Port res\_config\_ldap to use only the RFC-specified API. This should allow it to link against Sun's LDAP libraries.
  - The problem is centered around the use of the OpenLDAP-specific ldap\_initialize() call.
2. Change the detection routines in configure to use OpenSolaris' layout of OpenLDAP.
  - This seems doubtful simply because the filesystem layout of SUNWopenldap is so non-standard.

Despite the above two possibilities, there is a workaround to make Asterisk compile with res\_config\_ldap.

- Modify the "configure" script, changing all instances of "-lldap" to "-lldap-2.4".
  - At the time of this writing there are only 4 instances. This alone will make configure properly detect LDAP availability. But it will not compile.
- When running make, specify the use of the OpenLDAP headers like this:

```
"make LDAP_INCLUDE=-I/usr/include/openldap"
```

#### **Makefile layouts**

This has been fixed in Asterisk 1.8 and is no longer an issue.

In Asterisk 1.6 the Makefile overrides any usage of --prefix. I suspect the assumptions are from back before configure provided the ability to set the installation prefix. Regardless, if you are building on OpenSolaris, be aware of this behavior of the Makefile!

If you want to alter the install locations you will need to hand-edit the Makefile. Search for the string "SunOS" to find the following section:

```

# Define standard directories for various platforms
# These apply if they are not redefined in asterisk.conf
ifeq ($(OSARCH),SunOS)
    ASTETCDIR=/etc/asterisk
    ASTLIBDIR=/opt/asterisk/lib
    ASTVARLIBDIR=/var/opt/asterisk
    ASTDBDIR=$(ASTVARLIBDIR)
    ASTKEYDIR=$(ASTVARLIBDIR)
    ASTSPOOLDIR=/var/spool/asterisk
    ASTLOGDIR=/var/log/asterisk
    ASTHEADERDIR=/opt/asterisk/include/asterisk
    ASTBINDIR=/opt/asterisk/bin
    ASTSBINDIR=/opt/asterisk/sbin
    ASTVARRUNDIR=/var/run/asterisk
    ASTMANDIR=/opt/asterisk/man
else

```

Note that, despite the comment, these definitions have build-time and run-time implications. Make sure you make these changes BEFORE you build!

### FAX support with SpanDSP

I have been able to get this to work reliably, including T.38 FAX over SIP. If you are running Asterisk 1.6 note [Ticket 16342](#) if you do not install SpanDSP to the default locations (/usr/include and /usr/lib).

There is one build issue with SpanDSP that I need to document (FIXME)

### Gotchas

#### Runtime issues

- WAV and WAV49 files are not written correctly (see [Ticket 16610](#))
- 32-bit binaries on Solaris are limited to 255 file descriptors by default. (see [http://developers.sun.com/solaris/articles/stdio\\_256.html](http://developers.sun.com/solaris/articles/stdio_256.html))

#### Build issues

- bootstrap.sh does not correctly detect OpenSolaris build tools (see [Ticket 16341](#))
- Console documentation is not properly loaded at startup (see [Ticket 16688](#))
- Solaris sed does not properly create AEL parser files (see [Ticket 16696](#); workaround is to install GNU sed with SUNWgsed)
- Asterisk's provided install script, install-sh, is not properly referenced in the makeopts file that is generated during the build. One workaround is to install GNU install from the SUNWgnu-coreutils package. (See [Ticket 16781](#))

Finally, Solaris memory allocation seems far more sensitive than Linux. This has resulted in the discovery of several previously unknown bugs related to uninitialized variables that Linux handled silently. Note that this means, until these bugs are found and fixed, you may get segfaults.

At the time of this writing I have had a server up and running reasonably stable. However, there are large sections of Asterisk's codebase I do not use and likely contain more of these uninitialized variable problems and associated potential segfaults.

# Hello World

- Hello World with Asterisk and SIP
  - Requirements
  - Configuration files needed
  - Configure extensions.conf
  - Configure a SIP channel driver
    - Configure chan\_sip
    - Configure chan\_pjsip
  - Configure your SIP phone
  - Start Asterisk
  - Make the call

You've just installed Asterisk and you have read about basic configuration. Now let's quickly get a phone call working so you can get a taste for a simple phone call to Asterisk.

## Hello World with Asterisk and SIP

### Requirements

This tutorial assumes the following:

- You have a SIP phone plugged into the same LAN where the Asterisk server is plugged in, or can install the Zoiper softphone used in the example
- If you use your own hardware phone, we assume both the phone and Asterisk can reach each other and are on the same subnet.
- When you built Asterisk, you should have made sure to build the SIP channel driver you wanted to use, which may imply other requirements. For example if you want to use chan\_pjsip, then make sure you followed the [Installing pjproject](#) guide.

### Configuration files needed

You should have already run "make samples" if you installed from source, otherwise you may have the sample config files if you installed from packages.

If you have no configuration files in `/etc/asterisk/` then grab the sample config files from the [source directory](#) by navigating to it and running "make samples".

Files needed for this example:

- asterisk.conf
- modules.conf
- extensions.conf
- sip.conf or pjsip.conf

You can use the defaults for asterisk.conf and modules.conf, we'll only need to modify extensions.conf and sip.conf or pjsip.conf.

To get started, go ahead and move to the `/etc/asterisk/` directory where the files are located.

```
cd /etc/asterisk
```

### Configure extensions.conf

Backup the sample extensions.conf and create a new one

```
mv extensions.conf extensions.sample
vim extensions.conf
```

I'm assuming you use the VI/VIM editor here, after all, it is the best.

We are going to use a very simple dialplan. A dialplan is simply instructions telling Asterisk what to do with a call.

Edit your blank extensions.conf to reflect the following:

```
[from-internal]
exten = 100,1,Answer()
same = n,Wait(1)
same = n,Playback(hello-world)
same = n,Hangup()
```

When a phone dials extension 100, we are telling Asterisk to **Answer** the call, **Wait** one second, then Play (**Playback**) a sound file (hello-world) to the channel and **Hangup**.

## Configure a SIP channel driver

Depending on the version of Asterisk in use, you may have the option of more than one SIP channel driver. You'll have to pick one to use for the example.



- Asterisk 11 and previous: chan\_sip is the primary option.
- Asterisk 12 and beyond: You'll probably want to use chan\_pjsip (the newest driver), but you still have the option of using chan\_sip as well

Follow the instructions below for the channel driver you chose.

### Configure chan\_sip

Backup and edit a new blank **sip.conf**, just like you did with extensions.conf.

Then add the following to your sip.conf file:

```
[general]
context=default

[6001]
type=friend
context=from-internal
host=dynamic
secret=unsecurepassword
disallow=all
allow=ulaw
```

Basic configuration will be explained in more detail in other sections of the wiki. For this example to work, just make sure you have everything exactly as written above. For the sake of terminology, it is useful to note that though we have this SIP configuration configured with "type=friend", most people refer to this as configuring a SIP peer.

### Configure chan\_pjsip

Backup and edit a new blank **pjsip.conf**, just like you did with extensions.conf.

Then add the following to your pjsip.conf file:

```
[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0

[6001]
type=endpoint
context=from-internal
disallow=all
allow=ulaw
auth=6001
aors=6001

[6001]
type=auth
auth_type=userpass
password=unsecurepassword
username=6001

[6001]
type=aor
max_contacts=1
```

## Configure your SIP phone

You can use any SIP phone you want of course, but for this demonstration we'll use Zoiper, a Softphone which just happens to be easily demonstrable.

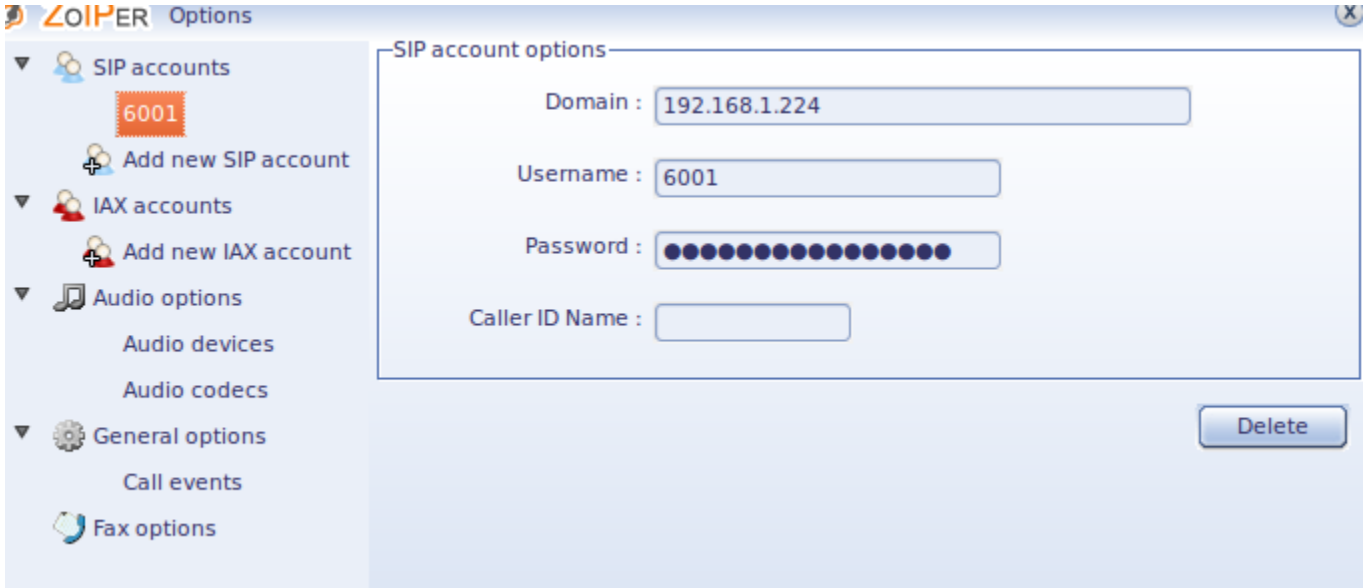
You can find the latest version of Zoiper for your platform [at their website](#). You can install it on the same system you are running Asterisk on, or it may make more sense to you if you install on another system on the same LAN (though you might find complication with software firewalls in that case).

Once you have Zoiper installed. Configure a new SIP account in Zoiper.

1. Once Zoiper is opened, click the wrench icon to get to settings.
2. Click "Add new SIP account"
3. Enter 6001 for the account name, click OK
4. Enter the IP address of your Asterisk system in the **Domain** field
5. Enter 6001 in the **Username** field
6. Enter your SIP peer's password in the **Password** field



7. Enter whatever you like in **Caller ID Name** or leave it blank
8. Click OK



Your results should look like the above screen shot.

## Start Asterisk

Back at the Linux shell go ahead and start Asterisk. We'll start Asterisk with a control console (-c) and level 5 verbosity (vvvvv).

```
asterisk -cvvvvv
```

Or if Asterisk is already running, restart Asterisk from the shell and connect to it.

```
asterisk -rx "core restart now"
asterisk -rvvvvv
```

## Make the call

Go back to the main Zoiper interface, and make sure the account is registered. Select the account from the drop down list and click the **Register** button next to it. If it says registered, you are good to go. If it doesn't register, then double check your configuration.

Once registered, enter extension 100 and click the Dial button. The call should be made and you should hear the sound file hello-world!

On the Asterisk CLI, you should see something like:

```
centosclean*CLI>
== Using SIP RTP CoS mark 5
-- Executing [100@from-internal:1] Answer("SIP/6001-00000001", "") in new stack
-- Executing [100@from-internal:2] Wait("SIP/6001-00000001", "1") in new stack
-- Executing [100@from-internal:3] Playback("SIP/6001-00000001", "hello-world") in new stack
-- <SIP/6001-00000001> Playing 'hello-world.gsm' (language 'en')
-- Executing [100@from-internal:4] Hangup("SIP/6001-00000001", "") in new stack
== Spawn extension (from-internal, 100, 4) exited non-zero on 'SIP/6001-00000001'
centosclean*CLI> _
```

Now that you have made a very simple call, you may want to start reading through the other sections on the wiki to learn more about [Operation](#), [Fundamentals](#) and [Configuration](#).



# Operation

Top-level page for a section for documentation concerning the operation of the Asterisk program and it's environment. Such as: , How to run Asterisk, System requirements, Maintenance, Logging, CLI usage, etc.

# System Requirements

In order to compile and install Asterisk, you'll need to install a C compiler and a number of system libraries on your system.

- [Compiler](#)
- [System Libraries](#)

## Compiler

The compiler is a program that takes source code (the code written in the C programming language in the case of Asterisk) and turns it into a program that can be executed. Currently, Asterisk version 1.8 and later depend on extensions offered by the **GCC** compiler for its `RAII_VAR` macro implementation, so **GCC** must be used to compile Asterisk. There are currently efforts under way to make Asterisk compatible with Clang's equivalent extensions.

If the **GCC** compiler isn't already installed on your machine, simply use appropriate package management system on your machine to install it. You'll also want to install **GCC**'s C++ compiler (`g++`) as well since certain Asterisk modules require it.

# System Libraries

In addition to the C compiler, you'll also need a set of system libraries. Essential libraries are used by Asterisk and must be installed before you can compile Asterisk. Core libraries allow compilation of additional core supported features. On most operating systems, you'll need to install both the library and its corresponding development package.



## Development libraries

For most operating systems, the development packages will have `-dev` or `-devel` on the end of the name. For example, on a Red Hat Linux system, you'd want to install both the `"openssl"` and `"openssl-devel"` packages.

## Asterisk 1.8

### Essential Libraries

- openssl (dependency for chan\_sip)
- ncurses
- libxml2

### Core Libraries

- DAHD1
- unixodbc
- libspeex
- libspeexdsp
- libresample
- libcurl3
- libvorbis
- libogg
- libsrtp
- libical
- libiksemel
- libneon
- libgmime

## Asterisk 11

### Additional Essential Libraries

- uuid
- libsqlite3

## Asterisk 12

### Additional Essential Libraries

- libxslt
- libjansson

### Additional Core Libraries

- [pjproject](#)

We recommend you use the package management system of your operating system to install these libraries before compiling and installing Asterisk.



## Help Finding the Right Libraries

If you're installing Asterisk 1.6.1.0 or later, it comes with a shell script called `install_prereq.sh` in the `contrib/scripts` sub-directory. If you run `install_prereq test`, it will give you the exact commands to install the necessary system libraries on your operating system. If you run `install_prereq install`, it will attempt to download and install the prerequisites automatically.

# Running Asterisk

## Running Asterisk from the Command Line

- By default, starting Asterisk will run it in the background:

```
# asterisk

# ps aux | grep asterisk
my_user      26246  2.0  4.1 2011992 165520 ?        Ssl  16:35   0:16 asterisk
```

- In order to connect to a running Asterisk process, you can attach a **remote console** using the `-r` option:

```
# asterisk -r

Asterisk 11.9.0, Copyright (C) 1999 - 2014 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=====
Connected to Asterisk 11.9.0 currently running on asterisk-server (pid = 26246)
asterisk-server*CLI>
```



The `-R` option will also attach a remote console - however, it will attempt to automatically reconnect to Asterisk if for some reason the connection is broken. This is particularly useful if your remote console restarts Asterisk.

### On this Page

- [Running Asterisk from the Command Line](#)
  - [Adding Verbosity](#)
    - [Remote Console Verbosity](#)
  - [Executing as another User](#)
  - [More Options](#)
- [Running Asterisk as a Service](#)
  - [Supported Distributions](#)

- To disconnect from a connected remote console, simply hit **Ctrl+C**:

```
asterisk-server*CLI>
Disconnected from Asterisk server
Asterisk cleanly ending (0).
Executing last minute cleanups
```

- To shut down Asterisk, issue `core stop gracefully`:

```
asterisk-server*CLI> core stop gracefully
Disconnected from Asterisk server
Asterisk cleanly ending (0).
Executing last minute cleanups
```



You can stop/restart Asterisk in many ways. See [Stopping and Restarting Asterisk From The CLI](#) for more information.

- You can start Asterisk in the foreground, with an attached **root console**, using the `-c` option:

```
# asterisk -c

Asterisk 11.9.0, Copyright (C) 1999 - 2014 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=====
Connected to Asterisk 11.9.0 currently running on asterisk-server (pid = 26246)
[May 16 17:02:50] NOTICE[27035]: loader.c:1323 load_modules: 287 modules will be loaded.
Asterisk Ready.
*CLI>
```

## Adding Verbosity

Asterisk provides a number of mechanisms to control the verbosity of its logging. One way in which this can be controlled is through the command line parameter `-v`. For each `-v` specified, Asterisk will increase the level of `VERBOSE` messages by 1. The following will create a console and set the `VERBOSE` message level to 2:

```
# asterisk -c -v -v
```

Command line parameters can be combined. The previous command can also be invoked in the following way:

```
# asterisk -cvv
```



The `VERBOSE` message level set via the command line is only applicable if the `asterisk.conf` `verbose` setting is not set.

## Remote Console Verbosity



This feature is only available in Asterisk 11 and later versions.

The verbosity of a remote console is set independently of the verbosity of other consoles and the core. A root console can be created with no verbosity:

```
# asterisk -c
```

While a remote console can be attached to that Asterisk process with a different verbosity:

```
# asterisk -rvvv
```

Multiple remote consoles can be attached, each with their own verbosity:

```
# asterisk -rv
```

## Executing as another User



### Do not run as root

Running Asterisk as `root` or as a user with super user permissions is dangerous and not recommended. There are many ways Asterisk can affect the system on which it operates, and running as `root` can increase the cost of small configuration mistakes.

For more information, see the [README-SERIOUSLY.bestpractices.txt](#) file delivered with Asterisk.

Asterisk can be run as another user using the `-U` option:

```
# asterisk -U asteriskuser
```

Often, this option is specified in conjunction with the `-G` option, which specifies the group to run under:

```
# asterisk -U asteriskuser -G asteriskuser
```

When running Asterisk as another user, make sure that user owns the various directories that Asterisk will access:

```
# sudo chown -R asteriskuser:asteriskuser /usr/lib/asterisk
# sudo chown -R asteriskuser:asteriskuser /var/lib/asterisk
# sudo chown -R asteriskuser:asteriskuser /var/spool/asterisk
# sudo chown -R asteriskuser:asteriskuser /var/log/asterisk
# sudo chown -R asteriskuser:asteriskuser /var/run/asterisk
# sudo chown asteriskuser:asteriskuser /usr/sbin/asterisk
```

## More Options

There are many more command line options available. For more information, use the `-h` option:

```
# asterisk -h
Asterisk 11.9.0, Copyright (C) 1999 - 2014, Digium, Inc. and others.
Usage: asterisk [OPTIONS]
...
```



## Running Asterisk as a Service

The most common way to run Asterisk in a production environment is as a service. Asterisk includes both a `make config` target for installing Asterisk as a service, as well as a script - `live_asterisk` - that will manage the service and automatically restart Asterisk in case of errors.

- Asterisk can be installed as a service using the `make config` target:

```
# make config
/etc/rc0.d/K91asterisk -> ../init.d/asterisk
/etc/rc1.d/K91asterisk -> ../init.d/asterisk
/etc/rc6.d/K91asterisk -> ../init.d/asterisk
/etc/rc2.d/S50asterisk -> ../init.d/asterisk
/etc/rc3.d/S50asterisk -> ../init.d/asterisk
/etc/rc4.d/S50asterisk -> ../init.d/asterisk
/etc/rc5.d/S50asterisk -> ../init.d/asterisk
```

- Asterisk can now be started as a service:

```
# service asterisk start
* Starting Asterisk PBX: asterisk                                     [ OK ]
```

- And stopped:

```
# service asterisk stop
* Stopping Asterisk PBX: asterisk                                   [ OK ]
```

- And restarted:

```
# service asterisk restart
* Stopping Asterisk PBX: asterisk                                     [ OK ]
* Starting Asterisk PBX: asterisk                                     [ OK ]
```

## Supported Distributions

Not all distributions of Linux/Unix are supported by the `make config` target. The following distributions are supported - if not using one of these distributions, the `make config` target may or may not work for you.

- RedHat/CentOS
- Debian/Ubuntu
- Gentoo
- Mandrake/Mandriva
- SuSE/Novell

## Stopping and Restarting Asterisk From The CLI

There are three common commands related to stopping the Asterisk service. They are:

1. **core stop now** - This command stops the Asterisk service immediately, ending any calls in progress.
2. **core stop gracefully** - This command prevents new calls from starting up in Asterisk, but allows calls in progress to continue. When all the calls have finished, Asterisk stops.
3. **core stop when convenient** - This command waits until Asterisk has no calls in progress, and then it stops the service. It does not prevent new calls from entering the system.

There are three related commands for restarting Asterisk as well.

1. **core restart now** - This command restarts the Asterisk service immediately, ending any calls in progress.
2. **core restart gracefully** - This command prevents new calls from starting up in Asterisk, but allows calls in progress to continue. When all the calls have finished, Asterisk restarts.
3. **core restart when convenient** - This command waits until Asterisk has no calls in progress, and then it restarts the service. It does not prevent new calls from entering the system.

There is also a command if you change your mind.

- **core abort shutdown** - This command aborts a shutdown or restart which was previously initiated with the gracefully or when convenient options.

See the [Asterisk Command Line Interface](#) section for more on that topic.

# Maintenance and Upgrades

So you have an Asterisk system running in production. Now what do you do?

Maintaining an Asterisk system can include tasks such as monitoring the system for issues, performing backups and keeping the system up to date. The topics of [Asterisk Backups](#) and [Updating or Upgrading Asterisk](#) are discussed on the linked sub-pages. A few miscellaneous maintenance topics are discussed below.

## File sizes

Files generated by various Asterisk modules or core features may grow to significant sizes depending on how you use Asterisk and the configuration of those sub-systems.

Systems that may generate large files are:

- [Logging](#)
- [Reporting](#)
- Audio recording applications
  - There are multiple ways to record audio files, applications such as [MixMonitor](#) exist for the purpose of audio recording, other applications, e.g. [ConfBridge](#) provide audio recording as a sub-feature. The recordings will either go to your default sounds directory (Specified in asterisk.conf) or a directory specified via the application or a configuration file related to the responsible module.

The key is to know where these components store their output and to have some mechanism in place to prevent the files from growing to a point where you have no storage space remaining.

Managing log files in general is outside the scope of this documentation, however a little Internet searching will get you a long way in that area.

The [Directory and File Structure](#) wiki page will tell you where most Asterisk files are stored on the Linux file-system.

## Security

It is in the interest of every Asterisk administrator to perform due diligence for security concerns. Most security measures are a matter of configuration and prevention, however for a production system already running there are a few things to consider in the context of maintenance.

- The [Asterisk Security Event Logger](#) can generate log output for security events. You may want to monitor these manually or have scripts and applications that take action on these events or log messages.
- Be aware of [security vulnerability announcements](#). There are a few places these are announced:
  - <http://www.asterisk.org/downloads/security-advisories>
  - [Asterisk-security mailing list](#)
  - [Asterisk-announce mailing list](#)

## Interfaces for Monitoring Asterisk Status

Maintenance can mean keeping an eye on the system and its state. The [wiki discusses Asterisk interfaces](#), such as SNMP or APIs such as AMI. Through these interfaces you can monitor Asterisk in a variety of ways or even affect control over calls.



### Reporting Issues that You Encounter

In the process of monitoring the operation of your system, you might spot an issue. If you believe the issue is a bug with Asterisk rather than a configuration issue, then you should follow the guidelines at the [Asterisk Issue Guidelines](#) page to report a bug.

# Asterisk Backups

## Backing up Asterisk Data

Backing up Asterisk is not a complex task. Mostly, you just need to know where the files are and then employ common tools for archiving and storing those files somewhere.

### Files to consider for backup

- [Asterisk configuration](#)
- [Asterisk internal DB](#)
- [Other database used by Asterisk](#)
- [Asterisk logs and reports](#)

The [Directory and File structure](#) page should direct you to where most of these files reside. Otherwise check the individual wiki pages for information on the location of their output.

Other than just using `tar` to archive and compress the files, you might set up a [cron job](#) in Linux to regularly perform that process and send the files off-site. In general, use whatever backup processes you use for any other Linux applications that you manage.

## Restoring a Backup

Restoring a backup, in most cases should be as simple as placing the files back in [their original locations](#) and [starting Asterisk](#).

When restoring a backup to a new major version of Asterisk you'll need to take the same steps as if you were [upgrading Asterisk](#). That is because a [new major version](#) may include changes to the format or syntax of configuration, required database schema, or applications and functions could be deprecated, removed or just have different behavior.

# Updating or Upgrading Asterisk

## Keeping the System Up-to-date

### Definition of Updates and Upgrades

**Updates** involve installing a new minor version. That is, moving from 11.X.X to 11.X.X vs moving from 11 to 12. New minor versions include bug fixes, some of which may be security related. Typically, there should not be features or new functionality included in a minor version except when the fixing of a bug requires it.

**Upgrades** involve installing a new major version. For example, moving from 11 to 12. New major versions can include bug fixes, new features and functionality, changes to old features or functionality, deprecation of functionality or change in support status.

#### Updates and upgrades should only be performed when necessary

- Reason to Update
  - Your install is affected by a bug or [security vulnerability](#) and you believe the new minor version will fix your issue.
- Reason to Upgrade
  - You require new features or enhancements only available in a new major version and are ready for the work involved in upgrading.

When considering an update or upgrade you should be familiar with the [Asterisk support life-cycle](#). It is useful to know the support status of the version you may be moving to.

### Researching a New Asterisk Version

Included with Asterisk releases are a few files that are useful for researching the version you are considering for update or upgrade. These can be found in the root of the Asterisk source directory.

1. UPGRADE.txt - Documents any changes that you need to know about when moving between major versions. Especially changes that break backwards compatibility.
2. CHANGES - Documents new or enhanced functionality between versions listed in the file.
3. ChangeLog - A log showing all changes (bug fixes, new features, security fixes, etc) between released versions. It can be useful if you are searching for a specific fix or change, but this could be overly verbose for looking at changes between two major versions.

### Performing Updates

#### Process

1. Research the new minor version you intend to update to.
2. Be sure you [have a backup](#) of any essential data on the system.
3. If you determine one of those changes will be beneficial for you, only then proceed with an update.
4. Download the new version and [install Asterisk](#).

### Performing Upgrades

#### Process

1. Research the new major version you are considering for an upgrade.
2. Be sure you [have a backup](#) of any essential data on the system.
3. If you determine the new functionality or changes will be beneficial then proceed with the upgrade.
4. On a test system, a non-production system, download and [install the new version](#).
5. Migrate backups of configuration, databases and other data to the new Asterisk install.
6. Test this new system, or simulate your production environment before moving this new system into production.
  - a. Especially test any areas of Asterisk where behavior changes have been noted in the UPGRADE.txt or CHANGES files. APIs, like AGI, AMI or ARI connecting to custom applications or scripts should be thoroughly tested. You should always try to extensively test your dialplan.

### Third Party Modules

When updating or upgrading Asterisk you should also check for updates to any third party modules you use. That is, modules that are not distributed with Asterisk. Those third party modules may require updates to work with your new version of Asterisk.

### Update and Upgrade Tips



Updates and upgrades could include changes to configuration samples. Sample files will not be updated unless you run "make samples" again or copy the new sample files from the source directory. Be careful not to overwrite your current configuration.



Keep old menuselect.makeopts files (see Asterisk source directory) and use them when building a new version to avoid customizing menuselect again when building a new version. This may only work for updates and not upgrades.



If you forget to re-build all Asterisk modules currently installed on the system then you may be prompted after compilation with a warning about those modules. That can be resolved by simply re-building those modules or re-installing them if you obtain them in binary form from a third party.

# Logging

Logging in Asterisk is a powerful mechanism that can be utilized to extract vital information from a running system. Asterisk currently has the capability to log messages to a variety of places that include files, consoles, and the syslog facility. Logging in Asterisk is configured in the `logger.conf` file. See [Logging Configuration](#) page for more information.

Along with the options defined in the logger configuration file, [commands are available at runtime](#) that allow a user to manipulate and even override certain settings via the CLI. Additionally [flags are available](#) with the Asterisk binary that allow similar configuration.

Most of the configuration for logging is concerning the activation or direction of various logging channels and their verbosity. It is important to note that the verbosity of logging messages is independent between root (or foreground) consoles and remote consoles. An example is provided in the [Verbosity in Core and Remote consoles sub-section](#).

## Dialplan Logging Applications

Logging can also be done in the dialplan utilizing the following applications:

### ***Log(<level>, <message>)***

Send arbitrary text to a selected log level, which must be one of the following: ERROR, WARNING, NOTICE, DEBUG, or VERBOSE.

### ***Verbose([<level>], <message>)***

Send arbitrary text to verbose output. "Level" here is an optional integer value (defaults to 0) specifying the verbosity level at which to output the message.

## Other Logging Resources

For information about extensive and detailed tracing of queued calls see the [queue logs](#) page. For instructions on how to help administrators and support givers to more quickly understand problems that occur during the course of calls see [call identifier logging](#) page. Also, if a problem is suspected see [collecting debug information](#) for help on how to collect debugging logs from an Asterisk machine (this can greatly help support and bug marshals). For details about logging security specific events see the [asterisk security event logger](#) page. Lastly, for advice on logging event data that can be grouped together to form a billing record see the [channel event logging \(CEL\)](#) page.

# Basic Logging Commands

Here is a selection of basic logging commands to get you started with manipulating log settings at the Asterisk CLI.

## **core set verbose**

Set the level of verbose messages to be displayed on the console. "0" or "off" means no verbose messages should be displayed. The silent option means the command does not report what happened to the verbose level. Equivalent to -v[v[...]] on start up.

Usage: core set verbose [atleast] <level> [silent]

## **core set debug**

Set the level of debug messages to be displayed or set a module name to display debug messages from. "0" or "off" means no messages should be displayed. Equivalent to -d[d[...]] on start up.

Usage: core set debug [atleast] <level> [module]

## **logger show channels**

List configured logger channels.

Usage: logger show channels

## **logger rotate**

Rotates and Reopens the log files.

Usage: logger rotate

## **logger reload**

Reloads the logger subsystem state. Use after restarting syslogd(8) if using syslog logging.

Usage: logger reload [<alt-conf>]

## **core show settings**

Show miscellaneous core system settings. Along with showing other various settings, issuing this command will show the current debug level as well as the root and current console verbosity levels. These log settings can be found under the "PBX Core Settings" section after executing the command.

Usage: core show settings



## Basic Logging Start-up Options

As previously indicated, at start up both the debug and verbose levels can also be set via command line arguments. To set the debug level on start up use the "-d" argument optionally followed by more "d"s. Asterisk will start with a debug level represented by the number of "d"s specified. As an example, the following will start Asterisk with a debug level of "3":

```
asterisk -ddd
```

To set the verbose level on start up use the "-v" argument optionally followed by more "v"s. Asterisk will start with a verbose level represented by the number of "v"s specified. As an example, the following will start Asterisk with a verbose level of "3":

```
asterisk -vvv
```

And of course both of these arguments can be specified at the same time:

```
asterisk -dddvvv
```

# Call Identifier Logging

## Overview

Call ID Logging (which has nothing to do with caller ID) is a new feature of Asterisk 11 intended to help administrators and support givers to more quickly understand problems that occur during the course of calls. Channels are now bound to call identifiers which can be shared among a number of channels, threads, and other consumers.

## Usage

No configuration is needed to take advantage of this feature. Asterisk 11 will simply apply an additional bracketed tag to all log messages generated by a thread with a call ID bound or to any log messages specially written to use call identifiers. For example:

- Asterisk receives a request for a non existent extension from SIP/gold
- The following log message is displayed:

```
[Oct 18 10:26:11] NOTICE[27538][C-00000000]: chan_sip.c:25107 handle_request_invite:  
Call from 'gold' (10.24.22.201:5060) to extension '645613' rejected because  
extension not found in context 'default'.
```

C-00000000 is the call identifier associated with this attempted call. All call identifiers are represented as C-XXXXXXXX where XXXXXXXX is an 8 digit hexadecimal value much like what you will see with SIP and local channel names.

Aside from log messages, call identifiers are also shown in the output for the 'core show channel <channel name>' command.

## Transfers

Transfers can be a little tricky to follow with the call ID logging feature. As a general rule, an attended transfer will always result in a new call ID being made because a separate call must occur between the party that initiates the transfer and whatever extension is going to receive it. Once the attended transfer is completed, the channel that was transferred will use the Call ID created when the transferrer called the recipient.

Blind transfers are slightly more variable. If a SIP peer 'peer1' calls another SIP peer 'peer2' via the dial application and peer2 blind transfers peer1 elsewhere, the call ID will persist. If on the other hand, peer1 blind transfers peer2 at this point a new call ID will be created. When peer1 transfers peer2, peer2 has a new channel created which enters the PBX for the first time, so it creates a new call ID. When peer1 is transferred, it simply resumes running PBX, so the call is still considered the same call. By setting the debug level to 3 for the channel internal API (channel\_internal\_api.c), all call ID settings for every channel will be logged and this may be able to help when trying to keep track of calls through multiple transfers.

# Collecting Debug Information

## Collecting Debug Information for the Asterisk Issue Tracker

This document will provide instructions on how to collect debugging logs from an Asterisk machine, for the purpose of helping bug marshals troubleshoot an issue on <https://issues.asterisk.org>

### STEPS

- Collecting Debug Information for the Asterisk Issue Tracker
- STEPS
  - Configure Asterisk logging
  - Configure verbosity levels and rotate logs
  - Enable channel tech or feature specific debug
  - Issue reproduction and clean up
  - Provide debug to the developers

### Configure Asterisk logging

1. Edit the logger.conf file to enable specific logger channels to output to your filesystem. The word "debug\_log\_123456" can be changed to anything you want, as that is the filename the logging will be written to.

Modify the file name "debug\_log\_123456" to reflect your [issues.asterisk.org](https://issues.asterisk.org) issue number.

```
logger.conf

[logfiles]
debug_log_123456 => notice,warning,error,debug,verbose,dtmf
```



#### Asterisk 13+

In Asterisk 13 and later, you can dynamically create log channels from the CLI using the `logger add channel` command. For example, to create the log file above, you would enter:

```
logger add channel debug_log_123456 notice,warning,error,debug,verbose,dtmf
```

The new log channel persists until Asterisk is restarted, the logger module is reloaded, or the log files are rotated. If using this CLI command, do **not** reload/restart/rotate the log files in Step 2.

### Configure verbosity levels and rotate logs

2. From the Asterisk CLI, set the verbose and debug levels for logging (this affects CLI and log output) and then restart the logger module:

```
*CLI> core set verbose 5
*CLI> core set debug 5
*CLI> module reload logger
```

Optionally, if you've used this file to record data previously, then rotate the logs:

```
*CLI> logger rotate
```

### Enable channel tech or feature specific debug

2.1. Depending on your issue and if a protocol level trace is requested, be sure to enable logging for the channel driver or other module.

Module (version)	CLI Command
New PJSIP driver (12 or higher)	<code>pjsip set logger on</code>
SIP (1.6.0 or higher)	<code>sip set debug on</code>
SIP (1.4)	<code>sip set debug</code>
IAX2 (1.6.0 or higher)	<code>iax2 set debug on</code>

IAX2 (1.4)	iax2 set debug
CDR engine	cdr set debug on

## Issue reproduction and clean up

3. Now that logging is configured, enabled and verbosity is turned up you should reproduce your issue.

4. Once finished, be sure to disable the extra debugging:

```
*CLI> core set verbose 0
*CLI> core set debug 0
```

4.1. Again, remember to disable any extra logging for channel drivers or features.

SIP (1.4 or higher)

```
*CLI> sip set debug off
```

IAX2 (1.4 or higher)

```
*CLI> iax2 set debug off
```

5. Disable logging to the filesystem. Edit the `logger.conf` file and comment out or delete the line you added in step 1. Using a semi-colon as the first character on the line will comment out the line.

```
logger.conf
[logfiles]
;debug_log_123456 => notice,warning,error,debug,verbose,dtmf
```

Then reload the logger module (or restart Asterisk) as you did in step 2:

```
*CLI> module reload logger
```

## Provide debug to the developers

6. Upload the file located in `/var/log/asterisk/debug_log_123456` to the issue tracker.



1. Do **NOT** post the output of your file as a comment. This clutters the issue and will only result in your comment being deleted.
2. Attach the file with a `.txt` extension to make it easy for the developers to quickly open the file without downloading. Files are attached on the issue page with following menu items: ( More > Attach files )

# Queue Logs

In order to properly manage ACD queues, it is important to be able to keep track of details of call setups and teardowns in much greater detail than traditional call detail records provide. In order to support this, extensive and detailed tracing of every queued call is stored in the queue log, located (by default) in `/var/log/asterisk/queue_log`.

## How do I interpret the lines in the Queue log?

The actual `queue_log` file will contain lines looking like the following:

```
1366720340|1366720340.303267|MYQUEUE|SIP/8007|RINGNOANSWER|1000
```

The pipe delimited fields from left to right are:

- UNIX timestamp
- Typically a Unique ID for the queue callers channel (based on the UNIX timestamp), also possible "REALTIME" or "NONE"
- Queue name
- Queue member channel
- Event type (see below reference)
- All fields to the right of the event type are event parameters

## Queue log event types

These are the events (and associated information) in the queue log:

- **ABANDON**(position|origposition|waittime) - The caller abandoned their position in the queue. The position is the caller's position in the queue when they hungup, the origposition is the original position the caller was when they first entered the queue, and the waittime is how long the call had been waiting in the queue at the time of disconnect.
- **ADDMEMBER** - A member was added to the queue. The bridged channel name will be populated with the name of the channel added to the queue.
- **AGENTDUMP** - The agent dumped the caller while listening to the queue announcement.
- **AGENTLOGIN**(channel) - The agent logged in. The channel is recorded.
- **AGENTCALLBACKLOGIN**(exten@context) - The callback agent logged in. The login extension and context is recorded.
- **AGENTLOGOFF**(channel|logintime) - The agent logged off. The channel is recorded, along with the total time the agent was logged in.
- **AGENTCALLBACKLOGOFF**(exten@context|logintime|reason) - The callback agent logged off. The last login extension and context is recorded, along with the total time the agent was logged in, and the reason for the logoff if it was not a normal logoff (e.g., Autologoff, Chanunavail)
- **ATTENDEDTRANSFER**(method|method-data|holdtime|calltime|origposition) - (Added in 12) This message will indicate the method by which the attended transfer was completed: `BRIDGE` for a bridge merge, `APP` for running an application on a bridge or channel, or `LINK` for linking two bridges together with local channels.
- **BLINDTRANSFER**(extension|context|holdtime|calltime|origposition) - (Added in 12) A blind transfer will result in a `BLINDTRANSFER` message with the destination context and extension.
- **COMPLETEAGENT**(holdtime|calltime|origposition) - The caller was connected to an agent, and the call was terminated normally by the agent. The caller's hold time and the length of the call are both recorded. The caller's original position in the queue is recorded in origposition.
- **COMPLETECALLER**(holdtime|calltime|origposition) - The caller was connected to an agent, and the call was terminated normally by the caller. The caller's hold time and the length of the call are both recorded. The caller's original position in the queue is recorded in origposition.
- **CONFIGRELOAD** - The configuration has been reloaded (e.g. with asterisk `-rx reload`)
- **CONNECT**(holdtime|bridgedchanneluniqueid|ringtime) - The caller was connected to an agent. Hold time represents the amount of time the caller was on hold. The bridged channel unique ID contains the unique ID of the queue member channel that is taking the call. This is useful when trying to link recording filenames to a particular call in the queue. Ringtime represents the time the queue members phone was ringing prior to being answered.
- **ENTERQUEUE**(url|callerid) - A call has entered the queue. URL (if specified) and Caller\*ID are placed in the log.
- **EXITEMPTY**(position|origposition|waittime) - The caller was exited from the queue forcefully because the queue had no reachable members and it's configured to do that to callers when there are no reachable members. The position is the caller's position in the queue when they hungup, the origposition is the original position the caller was when they first entered the queue, and the waittime is how long the call had been waiting in the queue at the time of disconnect.
- **EXITWITHKEY**(key|position|origposition|waittime) - The caller elected to use a menu key to exit the queue. The key and the caller's position in the queue are recorded. The caller's entry position and amount of time waited is also recorded.

- EXITWITHTIMEOUT(position|origposition|waittime) - The caller was on hold too long and the timeout expired. The position in the queue when the timeout occurred, the entry position, and the amount of time waited are logged.
- QUEUESTART - The queueing system has been started for the first time this session.
- REMOVEMEMBER - A queue member was removed from the queue. The bridge channel field will contain the name of the member removed from the queue.
- RINGNOANSWER(ringtime) - After trying for ringtime ms to connect to the available queue member, the attempt ended without the member picking up the call. Bad queue member!
- SYSCOMPAT - A call was answered by an agent, but the call was dropped because the channels were not compatible.
- TRANSFER(extension|context|holdtime|calltime|origposition) - Caller was transferred to a different extension. Context and extension are recorded. The caller's hold time and the length of the call are both recorded, as is the caller's entry position at the time of the transfer. PLEASE remember that transfers performed by SIP UA's by way of a reinvite may not always be caught by Asterisk and trigger off this event. The only way to be 100% sure that you will get this event when a transfer is performed by a queue member is to use the built-in transfer functionality of Asterisk.

## Queue log options

There are one or more options for queue logging in queues.conf, such as "log\_membername\_as\_agent". See the queues.conf sample file for explanations of those options.

## Verbosity in Core and Remote Consoles

If one issues the "core show settings" command from the Asterisk CLI it will show both a "Root" and "Current" console verbosity levels. This is because each console, core or remote has an independent verbosity setting. For instance, if you start asterisk with the following command:

```
asterisk -cv
```

This starts Asterisk in console mode (will be the root console) with a verbose level set to "1". Now if one issues a "core show settings" from this console's CLI the following should be observed (note, not showing all settings):

```
*CLI> core show settings

PBX Core settings
-----
...
Root console verbosity: 1
Current console verbosity: 1
...
```

A remote console can now be attached with an initial verbosity level of "3" and a "core show settings" on that console should show a root console verbosity of "1" and a current console verbosity of "3":

```
asterisk -rvvv

*CLI> core show settings

PBX Core settings
-----
...
Root console verbosity: 1
Current console verbosity: 3
...
```

Changing the root console's verbosity will be reflected on both:

```
*CLI> core set verbose 2
Console verbose was 1 and is now 2.
*CLI> core show settings

PBX Core settings
-----
...
Root console verbosity: 2
Current console verbosity: 2
...
```

and then on the remote console:

```
*CLI> core show settings

PBX Core settings
-----
...
Root console verbosity: 2
Current console verbosity: 3
...
```

# Asterisk Command Line Interface

## What is the CLI?

The Command Line Interface, or console for Asterisk, serves a variety of purposes for an Asterisk administrator.

- Obtaining information on Asterisk system components
- Affecting system configuration
- Seeing log output, errors and warnings in real-time
- Generating calls for testing
- Viewing embedded help documentation such as for APIs, applications, functions and module configuration

The sub-sections under this page will discuss how to access and use the CLI. That is, CLI syntax, command line help, and other CLI related topics.

For information on configuration of the CLI, see the [Asterisk CLI Configuration](#) section of the wiki.



## Connecting to the Asterisk CLI

There are two ways to connect to an Asterisk console, either a **foreground console** when starting the Asterisk process or by connecting to a **remote console** after Asterisk is [already running](#).

### Connecting to a foreground console

This is as simple as passing the `-c` flag when starting Asterisk.

```
-c Provide a control console on the calling terminal. The console is similar to the remote console provided by -r. Specifying this option implies -f and will cause asterisk to no longer fork or detach from the controlling terminal. Equivalent to console = yes in asterisk.conf.
```

When working on a foreground console, you won't be able to `exit`, instead you'll have to stop Asterisk to return to the Linux shell. Most people will use a remote console when performing administration tasks.

After Asterisk has finished loading, you'll see the default CLI prompt. The text "server" will be replaced with your system's hostname.

```
server*CLI>
```

### Connecting to a remote console

Connecting to a remote console using the `-r` or `-R` flags.

```
-r Instead of running a new Asterisk process, attempt to connect to a running Asterisk process and provide a console interface for controlling it.
-R Much like -r. Instead of running a new Asterisk process, attempt to connect to a running Asterisk process and provide a console interface for controlling it. Additionally, if connection to the Asterisk process is lost, attempt to reconnect for as long as 30 seconds.
```

To **exit a remote console**, simply type 'quit' or 'exit'. Please note that you can differentiate between a remote console and a foreground Asterisk console, as 'quit' or 'exit' will not function on the main console, which prevents an accidental shutdown of the daemon. If you would like to shutdown the Asterisk daemon from a remote console, there are [various commands available](#).



#### Executing Command Outside Of CLI

You can execute an Asterisk command from outside the CLI:

```
$ asterisk -rx "core reload"
```

```
$ asterisk -rx "core show help" | grep -i "sip"
```

# CLI Syntax and Help Commands

## Command Syntax and Availability

Commands follow a general syntax of **<module name> <action type> <parameters>**.

For example:

- **sip show peers** - returns a list of chan\_sip loaded peers
- **voicemail show users** - returns a list of app\_voicemail loaded users
- **core set debug 5** - sets the core debug to level 5 verbosity.



Commands are provided by the core, or by Asterisk modules. If the component that provides the commands is not loaded, then the commands it provides won't be available.

Asterisk does support command aliases. You can find information in the [Asterisk CLI Configuration](#) section.

## Finding Help at the CLI

### Command-line Completion

The Asterisk CLI supports command-line completion on all commands, including many arguments. To use it, simply press the **<Tab>** key at any time while entering the beginning of any command. If the command can be completed unambiguously, it will do so, otherwise it will complete as much of the command as possible. Additionally, Asterisk will print a list of all possible matches, if possible.

#### On this Page

- [Command Syntax and Availability](#)
- [Finding Help at the CLI](#)
  - [Command-line Completion](#)
  - [Listing commands and showing usage](#)
  - [Help for functions, applications and more](#)
  - [Module Configuration Help](#)

### Listing commands and showing usage

Once on the console, the 'help' alias (for 'core show help') may be used to see a large list of commands available for use.

```
*CLI> help
!                -- Execute a shell command
acl show         -- Show a named ACL or list all named ACLs
ael reload       -- Reload AEL configuration
...
```

The 'help' alias may also be used to obtain more detailed information on how to use a particular command and listing sub-commands. For example, if you type 'help core show', Asterisk will respond with a list of all commands that start with that string. If you type 'help core show version', specifying a complete command, Asterisk will respond with a usage message which describes how to use that command. As with other commands on the Asterisk console, the help command also responds to tab command line completion.

```
*CLI> help core show
core show applications [like|describing] -- Shows registered dialplan applications
core show application          -- Describe a specific dialplan application
...
```

```
*CLI> help core show version
Usage: core show version
Shows Asterisk version information.
```

### Help for functions, applications and more

A big part of working with Asterisk involves making use of Asterisk applications and functions. Often you'll want to know usage details for these, including their overall behavior or allowed arguments and parameters.

The command **core show application <application name>** or similarly **core show function <function name>** will show you the usage and arguments.

```
*CLI> core show application Wait
  -= Info about application 'Wait' -=
[Synopsis]
Waits for some time.

[Description]
This application waits for a specified number of <seconds>.

[Syntax]
Wait(seconds)

[Arguments]
seconds
  Can be passed with fractions of a second. For example, '1.5' will
  ask the application to wait for 1.5 seconds.
```

## Module Configuration Help

A very useful addition to Asterisk's help and documentation features is the command `config show help`. This command provides detailed information about configuration files, option sections in those files, and options within the sections.

```
*CLI> help config show help
Usage: config show help [<module> [<type> [<option>]]
Display detailed information about module configuration.
  * If nothing is specified, the modules that have
  configuration information are listed.
  * If <module> is specified, the configuration types
  for that module will be listed, along with brief
  information about that type.
  * If <module> and <type> are specified, detailed
  information about the type is displayed, as well
  as the available options.
  * If <module>, <type>, and <option> are specified,
  detailed information will be displayed about that
  option.
NOTE: the help documentation is partially generated at run
time when a module is loaded. If a module is not loaded,
configuration help for that module may be incomplete.
```

For example maybe we see the 'callerid' option in a pjsip.conf file sent to us from a friend. We want to know what that option configures. If we know that pjsip.conf is provided by the res\_pjsip module then we can find help on that configuration option.

```
*CLI> config show help res_pjsip endpoint callerid
[endpoint]
callerid = [(null)] (Default: n/a) (Regex: False)

CallerID information for the endpoint

Must be in the format 'Name <Number>', or only '<Number>'.
```

## Creating and Manipulating Channels from the CLI

Here we'll mention a few commands that allow you to create or manipulate channels at the CLI during runtime.

### channel request hangup

Provided by the core, this command simply allows you to request that a specified channel or all channels be hungup.

```
Usage: channel request hangup <channel>|<all>
Request that a channel be hung up. The hangup takes effect
the next time the driver reads or writes from the channel.
If 'all' is specified instead of a channel name, all channels
will see the hangup request.
```

An example:

```
newtonr-laptop*CLI> core show channels
Channel      Location      State  Application(Data)
SIP/6001-00000001  (None)      Up     Playback(demo-congrats)
1 active channel

newtonr-laptop*CLI> channel request hangup SIP/6001-00000001
Requested Hangup on channel 'SIP/6001-00000001'
[May 2 09:51:19] WARNING[7045][C-00000001]: app_playback.c:493 playback_exec: Playback failed on SIP/6001-00000001 for
demo-congrats
```

Here I made a call to an extension calling Playback, then from the CLI I requested that the established channel be hung up. You can see that it hung up in the middle of playing a sound file, so that sound file fails to continue playing.

### channel originate

Provided by `res_clioriginate.so`, this command allows you to create a new channel and have it connect to either a dialplan extension or a specific application.

```
There are two ways to use this command. A call can be originated between a
channel and a specific application, or between a channel and an extension in
the dialplan. This is similar to call files or the manager originate action.
Calls originated with this command are given a timeout of 30 seconds.
Usage1: channel originate <tech/data> application <appname> [appdata]
This will originate a call between the specified channel tech/data and the
given application. Arguments to the application are optional. If the given
arguments to the application include spaces, all of the arguments to the
application need to be placed in quotation marks.
Usage2: channel originate <tech/data> extension [exten@][context]
This will originate a call between the specified channel tech/data and the
given extension. If no context is specified, the 'default' context will be
used. If no extension is given, the 's' extension will be used.
```

An example:

```
newtonr-laptop*CLI> channel originate SIP/6001 extension 9999@somecontext
== Using SIP RTP CoS mark 5
-- Called 6001
-- SIP/6001-00000004 is ringing
> 0x7f0828067710 -- Probation passed - setting RTP source address to 10.24.18.16:4046
-- SIP/6001-00000004 answered
-- Executing [9999@somecontext:1] VoiceMailMain("SIP/6001-00000004", "") in new stack
-- <SIP/6001-00000004> Playing 'vm-login.gsm' (language 'en')
> 0x7f0828067710 -- Probation passed - setting RTP source address to 10.24.18.16:4046
```

We originated a call to the `chan_sip` peer 6001 in this case. The extension parameter tells it what extension to connect that channel to once the channel answers. In this case we connect it to an extension calling `VoiceMailMain`.

### channel redirect

Provided by `res_clioriginate.so`, this command allows you to redirect an existing channel to a dialplan extension.

```
Usage: channel redirect <channel> <[[context,]exten,]priority>
Redirect an active channel to a specified extension.
```

An example:

```
-- Executing [100@from-internal:1] Playback("SIP/6001-00000005", "demo-congrats") in new stack
> 0x7f07ec03e560 -- Probation passed - setting RTP source address to 10.24.18.16:4048
-- <SIP/6001-00000005> Playing 'demo-congrats.gsm' (language 'en')
newtonr-laptop*CLI> channel redirect SIP/6001-00000005 somecontext,9999,1
Channel 'SIP/6001-00000005' successfully redirected to somecontext,9999,1
[May  2 09:56:28] WARNING[7056][C-00000005]: app_playback.c:493 playback_exec: Playback failed on SIP/6001-00000005 for
demo-congrats
-- Executing [9999@somecontext:1] VoiceMailMain("SIP/6001-00000005", "") in new stack
-- <SIP/6001-00000005> Playing 'vm-login.gsm' (language 'en')
```

Here we make a call from SIP/6001 to a 100@from-internal, which results in a call to Playback. After the call is established, we issue a 'channel redirect' to redirect that channel to the extension 9999 in the context 'somecontext'. It is immediately placed into that extension and we hear the VoicemailMain prompt.

# Simple CLI Tricks

## Quit Typing the Whole Command

There are a couple of tricks that will help you on the Asterisk command-line interface. The most popular is tab completion. If you type the beginning of a command and press the Tab key, Asterisk will attempt to complete the name of the command for you, or show you the possible commands that start with the letters you have typed. For example, type `co` and then press the Tab key on your keyboard.

```
server*CLI> co[Tab]
config core
server*CLI> co
```

Now press the `r` key, and press tab again. This time Asterisk completes the word for you, as `core` is the only command that begins with `cor`. This trick also works with sub-commands. For example, type `core show` and press tab. (You may have to press tab twice, if you didn't put a space after the word `show`.) Asterisk will show you all the sub-commands that start with `core show`.

```
server*CLI> core show [Tab]
application      applications    calls          channel
channels         channeltype   channeltypes  codec
codecs           config        file           function
functions        help          hint           hints
image            license       profile        settings
switches         sysinfo      taskprocessors threads
translation      uptime       version        warranty
server*CLI> core show
```

## Repeat Previous Commands

Another trick you can use on the CLI is to cycle through your previous commands. Asterisk stores a history of the commands you type and you can press the **up arrow** key to cycle through the history.

## Run Linux Shell Commands from The Asterisk CLI

If you type an exclamation mark at the Asterisk CLI, you will get a Linux shell. When you exit the Linux shell (by typing `exit` or pressing **Ctrl+D**), you return to the Asterisk CLI. You can also type an exclamation mark and a Linux command, and the output of that command will be shown to you, and then you'll be returned to the Asterisk CLI.

```
server*CLI> !whoami
root
server*CLI>
```

# Asterisk Audio and Video Capabilities



PAGE UNDER CONSTRUCTION -Rusty



This page may not be accurate for Asterisk versions older than 11

## On This Page

- Overview of Media Support
- Enabling specific media support
  - Module compilation and loading
  - Channel driver configuration
    - Configuring allowed media for a PJSIP endpoint
    - Configuring app\_voicemail file formats for recordings
  - Endpoint device configuration
- Audio Support
  - Speex Support
  - Signed Linear PCM
- Video and Image Support

## Overview of Media Support

Asterisk supports a variety of audio and video media. Asterisk provides CODEC modules to facilitate encoding and decoding of audio streams. Additionally file format modules are provided to handle writing to and reading from the file-system.

The tables on this page describe what capabilities Asterisk supports and specific details for each format.

## Enabling specific media support

There are three basic requirements for making use of specific audio or video media with Asterisk.

1. The Asterisk core must support the format or a module may be required to add the support.
2. Asterisk configuration must be modified appropriately.
3. The devices interfacing with Asterisk must support the format and be configured to use it.

## Module compilation and loading

For audio or video capabilities that require a module - you should make sure that the module is built and installed on the system.

See the section on [Using Menuselect to Select Asterisk Options](#) if you need help figuring out how to get a module built and then section on [Configuring the Asterisk Module Loader](#) to verify that a module gets loaded when Asterisk starts up.

## Channel driver configuration

Audio or video capabilities for Asterisk are used on a per channel or per feature basis. To tell Asterisk what CODECs or formats to use in a particular scenario you may need to configure your channel driver, or modify configuration for the feature itself.

We'll provide two examples, but you should look at the documentation for the channel driver or feature to better understand how to configure media in that context.

## Configuring allowed media for a PJSIP endpoint

### pjsip.conf

```
[CATHY]
type=endpoint
context=from-internal
allow=!all,ulaw
auth=CATHY
aors=CATHY
```

We set the option "allow" to a string of values "!all,ulaw".

- The value **"!all"** means "Disallow all" and is identical to "disallow=all". This tells Asterisk to disallow all codecs except what we further define in the allow option.
- The value **"ulaw"** instructs Asterisk to allow ulaw audio during media negotiation for this endpoint.

See the section [Configuring res\\_pjsip](#) for more information on the PJSIP channel driver.

## Configuring app\_voicemail file formats for recordings

```

voicemail.conf

[general]
format=wav49,wav,gsm

```

In the general section of voicemail.conf you can set the formats used when writing voicemail to the file-system. We set the option "format" to a string of file format names.

- The value "wav49" represents GSM in a WAV|wav49 container.
- The value "wav" represents SLIN in a wav container.
- The value "gsm" represents GSM in straight gsm format.

## Endpoint device configuration

Configuring your particular device is outside the scope of the Asterisk documentation.

Consult your devices user/admin manual to find out where you define codecs or media to be used.

For VoIP desk phones there are typically two places to look for media configuration.

1. The web GUI for the phone.
2. The provisioning files that are pulled down by the phones on your network.

## Audio Support

A variety of audio capabilities are supported by Asterisk.

Name	Config Value	Capability: (T)ranscoding (P)assthrough	CODEC Module	Format Module	Distributed w/ Asterisk?	Commercial License Required?
ADPCM	adpcm	T	codec_adpcm	format_vox	YES	NO
G.711 A-law	alaw	T	codec_alaw	format_pcm	YES	NO
G.711 $\mu$ -law	ulaw	T	codec_ulaw	format_pcm	YES	NO
G.719	g719	P	?	format_g719	YES	NO
G.722	g722	T	codec_g722	format_pcm	YES	NO
G.722.1 Siren7	siren7	T	codec_siren7	format_siren7	Codec(NO) Format(YES)	NO
G.722.1C Siren14	siren14	T	codec_siren14	format_siren14	Codec(NO) Format(YES)	NO
G.723.1	g723	T	codec_g723	format_g723	Codec(NO) Format(YES)	YES(hardware required)
G.726	g726	T	codec_g726	format_g726	YES	NO
G.726 AAL2	g726aal2	T	codec_g726	format_g726	YES	NO
G.729A	g729	T	codec_g729a	format_g729	Codec(NO) Format(YES)	YES
GSM	gsm	T	codec_gsm	format_gsm	YES	NO
ILBC	ilbc	T	codec_ilbc	format_ilbc	YES	NO
LPC-10	lpc10	T	codec_lpc10	?	YES	NO
SILK	silk	T	codec_silk	?	Codec(NO) Format(YES)	NO
Speex	speex	T	codec_speex	?	YES	NO



Signed Linear PCM	slin	T	codec_resample	format_sln	YES	NO
Ogg Vorbis	N/A	?	?	format_ogg_vorbis	YES	NO
Opus	opus	P	N/A	N/A	YES	NO
wav (SLIN)	wav	T	N/A	format_wav	YES	NO
WAV (GSM)	wav49	T	N/A	format_wav_gsm	YES	NO

## Speex Support

Asterisk supports 8, 16, and 32kHz Speex. Use of the 32kHz Speex mode is, like the other modes, controlled in the respective channel driver's configuration file, e.g. chan\_sip's sip.conf or PJSIP's pjsip.conf.

## Signed Linear PCM

Asterisk can resample between several different sampling rates and can read/write raw 16-bit signed linear audio files from/to disk. The complete list of supported sampling rates and file format is found in the expansion link below:

↕ [Click here to expand...](#)

Sampling Rate	Asterisk File format
8kHz	.sln
12kHz	.sln12
16kHz	.sln16
24kHz	.sln24
32kHz	.sln32
44.1kHz	.sln44
48kHz	.sln48
96kHz	.sln96
192kHz	.sln192



Users can create 16-bit Signed Linear files of varying sampling rates from WAV files using the sox command-line audio utility.

```
sox input.wav -t raw -b 16 -r 32000 output.sln
mv output.sln output.sln32
```

In this example, an input WAV file has been converted to Signed Linear at a depth of 16-bits and at a rate of 32kHz. The resulting output.sln file is then renamed output.sln32 so that it can be processed correctly by Asterisk.

## Video and Image Support

You'll notice the CODEC module column is missing. Video transcoding or image transcoding is not currently supported.

Name	Config Value	Capability: (T)ranscoding (P)assthrough	Format Module	Distributed w/ Asterisk
JPEG	jpeg	P	format_jpeg	YES
H.261	h261	P	N/A	YES
H.263	h263	P	format_h263	YES
H.263+	h263p	P	format_h263	YES

H.264	h264	P	format_h264	YES
VP8	vp8	P	N/A	YES

# Fundamentals

## Asterisk Architecture

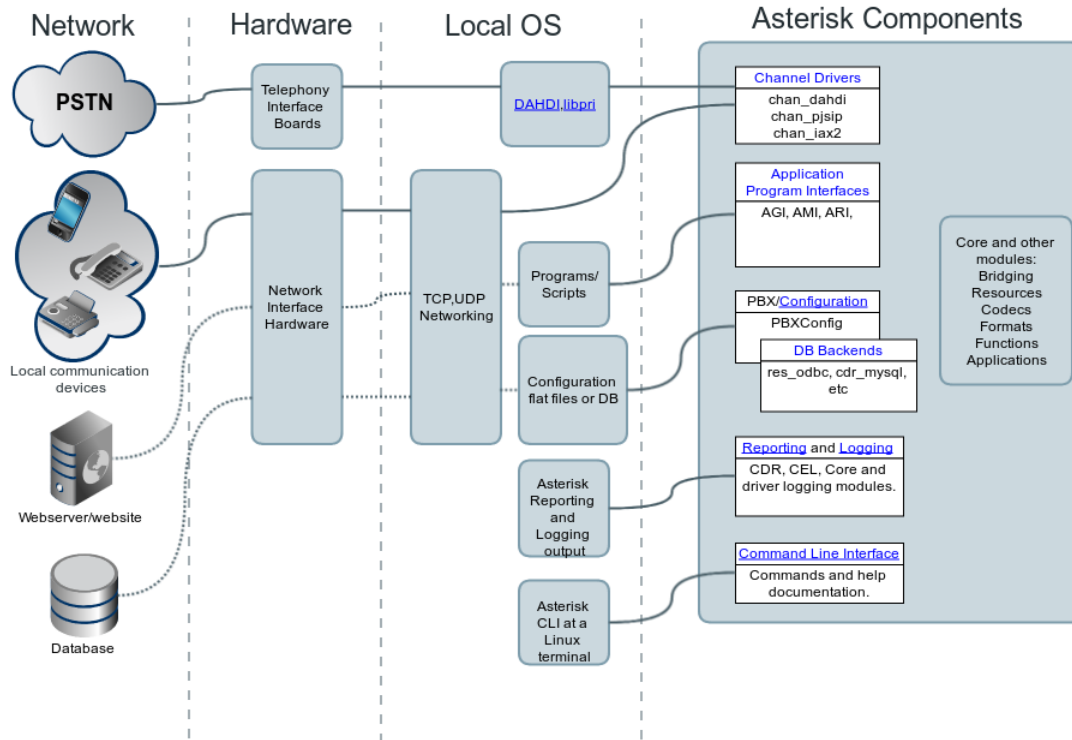
From an architectural standpoint, Asterisk is made up of many different modules. This modularity gives you an almost unlimited amount of flexibility in the design of an Asterisk-based system. As an Asterisk administrator, you have the choice on which modules to load and the configuration of each module. Each module that you load provides different capabilities to the system. For example, one module might allow your Asterisk system to communicate with analog phone lines, while another might add call reporting capabilities. In this section, we'll discuss the overall relationships of some Asterisk component, the various types of modules and their capabilities.

# Asterisk Architecture, The Big Picture

Before we dive too far into the various types of modules, let's first take a step back and look at the overall architecture of Asterisk.

Asterisk a big program with many components, with complex relationships. To be able to use it, you don't have to know how everything relates in extreme detail. Below is a simplified diagram intended to illustrate the relationships of some major components to each other and to entities outside Asterisk. It is useful to understand how a component may relate to things outside Asterisk as Asterisk is not typically operating without some connectivity or interaction with other network devices or files on the local system.

## An Asterisk System



Remember this is not an exhaustive diagram. It covers only a few of the common relationships between certain components.

### On this Page

- An Asterisk System
- Asterisk Architecture
  - The Core
  - Modules
  - Calls and Channels
  - Dialplan

## Asterisk Architecture

Asterisk has a **core** that can interact with many **modules**. Modules called channel drivers provide **channels** that follow Asterisk **dialplan** to execute programmed behavior and facilitate communication between devices or programs outside Asterisk. Channels often use **bridging** infrastructure to interact with other channels. We'll describe some of these concepts in brief below.

### The Core

The heart of any Asterisk system is the **core**. The PBX core is the essential component that provides a lot of infrastructure. Among many functions it reads the configuration files, including dialplan and loads all the other **modules**, distinct components that provide more functionality.

The core loads and builds the dialplan, which is the logic of any Asterisk system. The **dialplan** contains a list of instructions that Asterisk should follow to know how to handle incoming and outgoing **calls** on the system.

### Modules

Other than functionality provided by the core of Asterisk, modules provide all other functionality. The source for many modules is distributed with Asterisk, though other modules may be available from community members or even businesses that make commercial modules. The modules distributed with Asterisk can be [optionally be built](#) when Asterisk is built.

Modules are not only optionally built, but you can affect at load-time whether they will be loaded at all, the loading order or even unload/load them during run-time. Most modules are independently configurable and have their own [configuration](#) files. Some modules have support for configuration to be read statically or dynamically(realtime) from database backends.

From a logistical standpoint, these modules are typically files with a [.so](#) file extension, which live in the Asterisk [modules directory](#) (which is typically [/usr/lib/asterisk/modules](#)). When [Asterisk starts up](#), it loads these files and adds their functionality to the system.

Asterisk modules which are part of the core have a file name that look like [pbx\\_xxxx.so](#). All of the modules types are discussed in the section [Types of Asterisk Modules](#).



#### A Plethora of Modules

Take just a minute and go look at the Asterisk modules directory on your system. You should find a wide variety of modules. A default installation of Asterisk has over one hundred fifty different modules!

### A Few Module Examples

- [chan\\_pjsip](#) uses [res\\_pjsip](#) and many other [res\\_pjsip](#) modules to provide a SIP stack for SIP devices to interact with Asterisk and with each other through Asterisk.
- [app\\_voicemail](#) provides traditional PBX-type voicemail features.
- [app\\_confbridge](#) provides conference bridges with many optional features.
- [res\\_agi](#) provides the Asterisk Gateway Interface, an API that allows call control from external scripts and programs.

### Calls and Channels

As was mentioned in the [Asterisk as a Swiss Army Knife of Telephony](#) section, the primary purpose of Asterisk is being an engine for building Real Time Communication systems and applications.

In most but not all cases this means you'll deal with the concept of "calls". Calls in telephony terminology typically refer to one phone communicating with (calling) another phone over a medium, such as a [PSTN](#) line. However in the case of Asterisk a call typically references one or more [channels](#) existing in Asterisk.

Here are some example "calls".

- A phone calling another phone through Asterisk.
- A phone calling many phones at once (for example, paging) through Asterisk.
- A phone calls an application or the reverse happens. e.g., [app\\_voicemail](#) or [app\\_queue](#)
- A local channel is created and interacts with an application or another channel.

Note that I primarily use phones as an example, however you could refer to any channel or group of channels as a call. It doesn't matter if the devices are phones or something else, like an alarm system sensor or garage door opener.

### Channels

[Channels](#) are created by Asterisk using [Channel Drivers](#). They can utilize other resources in the Asterisk system to facilitate various types of communication between one or more devices. Channels can be [bridged](#) to other channels and be affected by [applications](#) and [functions](#). Channels can make use of many other resources provided by other modules or external libraries. For example SIP channels when passing audio will make use of the [codec](#) and [format](#) modules. Channels may interact with many different modules at once.

### Dialplan

Dialplan is the one main method of directing Asterisk behavior. Dialplan exists as text files (for example [extensions.conf](#)) either in the built-in dialplan scripting language, AEL or LUA formats. Alternatively dialplan could be read from a [database](#), along with other module configuration. When writing dialplan, you will make heavy use of [applications](#) and [functions](#) to affect channels, configuration and features.

Dialplan can also call out through other interfaces such as [AGI](#) to receive call control instruction from external scripts and programs. The [Dialplan](#) section of the wiki goes into detail on the usage of dialplan.

# Types of Asterisk Modules

There are many different types of modules, each providing their own functionality and capabilities to Asterisk. Configuration of loading is described in [Configuring the Asterisk Module Loader](#).



Use the CLI command **module show** to see all the loaded modules in your Asterisk system. See the command usage for details on how to filter the results shown with a pattern.

▼ [Click here for example "module show" output...](#)

```
mypbx*CLI> module show
```

Module	Description	Use Count	Status	Support Level
app_adsiprog.so	Asterisk ADSI Programming Application	0	Running	extended
app_agent_pool.so	Call center agent pool applications	0	Running	core
app_alarmreceiver.so	Alarm Receiver for Asterisk	0	Running	extended
app_amd.so	Answering Machine Detection Application	0	Running	extended
app_authenticate.so	Authentication Application	0	Running	core

## Various Module Types

- **Channel Drivers**

Channel drivers communicate with devices outside of Asterisk, and translate that particular signaling or protocol to the core.

- **Dialplan Applications**

Applications provide call functionality to the system. An application might answer a call, play a sound prompt, hang up a call or provide more complex behavior such as queuing, voicemail or conferencing feature sets.

- **Dialplan Functions**

Functions are used to retrieve, set or manipulate various settings on a call. A function might be used to set the Caller ID on an outbound call, for example.

- **Resources**

As the name suggests, resources provide resources to Asterisk and its modules. Common examples of resources include music on hold and call parking.

- **CODECs**

A CODEC (which is an acronym for COder/DECoder) is a module for encoding or decoding audio or video. Typically codecs are used to encode media so that it takes less bandwidth. These are essential to translating audio between the audio codecs and payload types used by different devices.

- **File Format Drivers**

File format drivers are used to save media to disk in a particular file format, and to convert those files back to media streams on the network.

- **Call Detail Record (CDR) Drivers**

CDR drivers write call logs to a disk or to a database.

- **Call Event Log (CEL) Drivers**

Call event logs are similar to call detail records, but record more detail about what happened inside of Asterisk during a particular call.

- **Bridge Drivers**

Bridge drivers are used by the bridging architecture in Asterisk, and provide various methods of bridging call media between participants in a call.

The next sub-sections will include detail on each of the module types.

## Channel Driver Modules

All calls from the outside of Asterisk go through a channel driver before reaching the core, and all outbound calls go through a channel driver on their way to the external device.

The PJSIP channel driver (`chan_pjsip`), for example, communicates with external devices using the SIP protocol. It translates the SIP signaling into the core. This means that the core of Asterisk is signaling agnostic. Therefore, Asterisk isn't just a SIP or VOIP communications engine, it's a multi-protocol engine.

For more information on the various channel drivers, see the configuration section for [Channel Drivers](#).

All channel drivers have a file name that look like `chan_xxxx.so`, such as `chan_pjsip.so` or `chan_dahdi.so`.



## Dialplan Application Modules

The application modules provide call functionality to the system. These applications are then scripted sequentially in the dialplan. For example, a call might come into Asterisk dialplan, which might use one application to answer the call, another to play back a sound prompt from disk, and a third application to allow the caller to leave voice mail in a particular mailbox.

For more information on dialplan applications, see [Applications](#).

All application modules have file names that looks like **app\_XXXX.so**, such as **app\_voicemail.so**, however applications and functions can also be provided by the core and other modules. Modules like `res_musiconhold` and `res_xmpp` provide applications related to their own functionality.

## Dialplan Function Modules

Dialplan functions are somewhat similar to dialplan applications, but instead of doing work on a particular channel or call, they simply retrieve or set a particular setting on a channel, or perform text manipulation. For example, a dialplan function might retrieve the Caller ID information from an incoming call, filter some text, or set a timeout for caller input.

For more information on dialplan functions, see [PBX Features](#).

All dialplan function modules have file names that look like **func\_xxxx.so**, such as **func\_callerid.so**, however applications and functions can also be provided by the core and other modules. Modules like `res_musiconhold` and `res_xmpp` provide applications related to their own functionality.

## Resource Modules

Resources provide functionality to Asterisk that may be called upon at any time during a call, even while another application is running on the channel. Resources are typically used as asynchronous events such as playing hold music when a call gets placed on hold, or performing call parking.

Resource modules have file names that look like **res\_xxxxx.so**, such as **res\_musiconhold.so**.

Resource modules can provide [Dialplan Applications](#) and [Dialplan Functions](#) even if those apps or functions don't have separate modules.

## Codec Modules

CODEC modules have file names that look like `codec_XXXXX.so`, such as `codec_alaw.so` and `codec_ulaw.so`.

CODECs represent mathematical algorithms for encoding (compressing) and decoding (decompression) media streams. Asterisk uses CODEC modules to both send and receive media (audio and video). Asterisk also uses CODEC modules to convert (or transcode) media streams between different formats.

### Modules Provided by Default

Asterisk is provided with CODEC modules for the following media types:

- ADPCM, 32kbit/s
- G.711 A-law, 64kbit/s
- G.711  $\mu$ -law, 64kbit/s
- G.722, 64kbit/s
- G.726, 32kbit/s
- GSM, 13kbit/s
- LPC-10, 2.4kbit/s

### Other Formats and Modules

The Asterisk core provides capability for 16 bit Signed Linear PCM, which is what all of the CODECs are encoding from or decoding to. There is another CODEC module, `codec_resample` which allows re-sampling of Signed Linear into different sampling rates 12,16,24,32,44,48,96 or 192 kHz to aid translation.

Various other CODEC modules will be built and installed if their dependencies are detected during Asterisk compilation.

- If the [DAHDI](#) drivers are detected then `codec_dahdi` will be installed.
- If the Speex ([www.speex.org](http://www.speex.org)) development libraries are detected, `codec_speex` will also be installed.
- If the iLBC ([www.ilbcfreeware.org](http://www.ilbcfreeware.org)) development libraries are detected, `codec_ilbc` will also be installed.

Support for the patent-encumbered G.729A or G.723.1 CODECs is provided by Digium on a commercial basis through software (G.729A) or hardware (G.729A and G.723.1) products. For more information about purchasing licenses or hardware to use the G.729A or G.723.1 CODECs with Asterisk, please see Digium's website.

Support for Polycom's patent-encumbered but free G.722.1 Siren7 and G.722.1C Siren14 CODECs, or for Skype's SILK CODEC, can be enabled in Asterisk by [downloading the binary CODEC modules](#) from Digium's website.



On the Asterisk Command Line Interface, use the command "core show translation" to show the translation times between all registered audio formats.

## File Format Drivers

Asterisk uses file format modules to take media (such as audio and video) from the network and save them on disk, or retrieve said files from disk and convert them back to a media stream. While often related to CODECs, there may be more than one available on-disk format for a particular CODEC.

File format modules have file names that look like **format\_xxxx.so**, such as **format\_wav.so** and **format\_jpeg.so**.

Below is a list of format modules included with recent versions of Asterisk:

- format\_g719
- format\_g723
- format\_g726
- format\_g729
- format\_gsm
- format\_h263
- format\_h264
- format\_ilbc
- format\_jpeg
- format\_ogg\_vorbis
- format\_pcm
- format\_siren7
- format\_siren14
- format\_sln
- format\_vox
- format\_wav\_gsm
- format\_wav

## Call Detail Record (CDR) Drivers

CDR modules are used to store [Call Detail Records \(CDR\)](#) in a variety of formats. Popular storage mechanisms include comma-separated value (CSV) files, as well as relational databases such as MySQL or PostgreSQL. Call detail records typically contain one record per call, and give details such as who made the call, who answered the call, the amount of time spent on the call, and so forth.

Call detail record modules have file names that look like `cdr_XXXXX.so`, such as `cdr_csv.so`. The recommended module to use for connecting to [CDR Storage Backends](#) is `cdr_adaptive_odbc.so`.

## Call Event Log (CEL) Driver Modules

Call Event Logs record the various actions that happen on a call. As such, they are typically more detailed than call detail records. For example, a call event log might show that Alice called Bob, that Bob's phone rang for twenty seconds, then Bob's mobile phone rang for fifteen seconds, the call then went to Bob's voice mail, where Alice left a twenty-five second voicemail and hung up the call. The system also allows for custom events to be logged as well.

For more information about Call Event Logging, see [Call Event Logging](#).

Call event logging modules have file names that look like **cel\_XXXXX.so**, such as **cel\_custom.so** and **cel\_adaptive\_odbc.so**.

## Bridging Modules

Beginning in Asterisk 1.6.2, Asterisk introduced a new method for bridging calls together. It relies on various bridging modules to control how the media streams should be mixed for the participants on a call. The new bridging methods are designed to be more flexible and more efficient than earlier methods.

Bridging modules have file names that look like **bridge\_xxxx.so**, such as **bridge\_simple.so** and **bridge\_multiplexed.so**.



# Directory and File Structure

The top level directories used by Asterisk can be configured in the `asterisk.conf` configuration file.

Here we'll describe what each directory is used for, and what sub-directories Asterisk will place in each by default. Below each heading you can also see the correlating configuration line in `asterisk.conf`.

## Asterisk Configuration Files

```
astetcdir => /etc/asterisk
```

This location is used to store and read Asterisk [configuration files](#). That is generally files with a `.conf` extension, but other configuration types as well, for example `.lua` and `.ael`.

## Asterisk Modules

```
astmoddir => /usr/lib/asterisk/modules
```

Loadable modules in Shared Object format (`.so`) installed by Asterisk or the user should go here.

## Various Libraries

```
astvarlibdir => /var/lib/asterisk
```

Additional library elements and files containing data used in runtime are put here.

### On This Page

- [Asterisk Configuration Files](#)
- [Asterisk Modules](#)
- [Various Libraries](#)
- [Database Directory](#)
- [Encryption Keys](#)
- [System Data Directory](#)
- [AGI\(Asterisk Gateway Interface\) Directory](#)
- [Spool Directories](#)
- [Running Process Directory](#)
- [Logging Output](#)
- [System Binary Directory](#)

## Database Directory

```
astdbdir => /var/lib/asterisk
```

This location is used to store the data file for Asterisk's internal database. In Asterisk versions using the SQLite3 database, the file will be named `astdb.sqlite3`.

## Encryption Keys

```
astkeydir => /var/lib/asterisk
```

When configuring key-based encryption, Asterisk will look in the **keys** subdirectory of this location for the necessary keys.

## System Data Directory

```
astdatadir => /var/lib/asterisk
```

By default, Asterisk sounds are stored and read from the **sounds** subdirectory at this location.

## AGI(Asterisk Gateway Interface) Directory

```
astagidir => /var/lib/asterisk/agi-bin
```

When using various AGI [applications](#), Asterisk looks here for the AGI scripts by default.

## Spool Directories

```
astspooldir => /var/spool/asterisk
```

This directory is used for storing spool files from various core and module-provided components of Asterisk.

Most of them use their own subdirectories, such as the following:

- dictate
- meetme
- monitor
- outgoing
- recording
- system
- tmp
- voicemail

## Running Process Directory

```
astrundir => /var/run/asterisk
```

When Asterisk is [running](#), you'll see two files here, **asterisk.ctl** and **asterisk.pid**. That is the control socket and the PID(Process ID) files for Asterisk.

## Logging Output

```
astlogdir => /var/log/asterisk
```

When Asterisk is configured to [provide log file](#) output, it will be stored in this directory.

## System Binary Directory

```
astsbin => /usr/sbin
```

By default, Asterisk looks in this directory for any system binaries that it uses, if you move the Asterisk binary itself or any others that it uses, you'll need to change this location.

# Asterisk Configuration

The top-level page for all things related to Asterisk configuration

## Asterisk Configuration Files

This Asterisk Configuration Files section covers the following:

- The [Config File Format](#) and syntax.
- How to use [Comments](#) in the files.
- [Using The include, tryinclude and exec Constructs](#) to include file content into other files or get external program output into a file
- [Adding to an existing section](#) settings from other configuration sections
- The syntax and usage of [Templates](#) for avoiding redundant configuration.

### See also

If you haven't read it already, the [Asterisk Architecture](#) section will help you to understand the context within which the configuration files are used. The [Directory and File Structure](#) will tell you exactly where to find the configuration files which we generalize in this section. See the [Configuration](#) section for information on how to configure files related to **specific** components of Asterisk.

## Config File Format

Asterisk is a very flexible telephony engine. With this flexibility, however, comes a bit of complexity. Asterisk has quite a few configuration files which control almost every aspect of how it operates. The format of these configuration files, however, is quite simple. The Asterisk configuration files are plain text files, and can be edited with any text editor.

## Sections and Settings

The configuration files are broken into various sections, with the section name surrounded by square brackets. Section names should not contain spaces, and are case sensitive. Inside of each section, you can assign values to various settings. Note that settings are also referred to as configuration options or just, options. In general, settings in one section are independent of values in another section. Some settings take values such as true or false, while other settings have more specific settings. The syntax for assigning a value to a setting is to write the setting name, an equals sign, and the value, like this:

```
[section-name]
setting=true

[another_section]
setting=false
setting2=true
```

Additionally here is closer to real-life example from the `pjsip.conf.sample` file:

```
[transport-udp-nat]
type=transport
protocol=udp
bind=0.0.0.0
local_net=192.0.2.0/24
external_media_address=203.0.113.1
external_signaling_address=203.0.113.1
```

## Objects

Some Asterisk configuration files also create objects. The syntax for objects is slightly different than for settings. To create an object, you specify the type of object, an arrow formed by the equals sign and a greater-than sign (`=>`), and the settings for that object.

```
[section-name]
some_object => settings
```



### Confused by Object Syntax?

In order to make life easier for newcomers to the Asterisk configuration files, the developers have made it so that you can also create objects with an equal sign. Thus, the two lines below are functionally equivalent.

```
some_object => settings
some_object=settings
```

It is common to see both versions of the syntax, especially in online Asterisk documentation and examples. This book, however, will denote objects by using the arrow instead of the equals sign.

```
[section-name]
label1=value1
label2=value2
object1 => name1

label1=value0
label3=value3
object2 => name2
```

In this example, **object1** inherits both **label1** and **label2**. It is important to note that **object2** also inherits **label2**, along with **label1** (with the new overridden value **value0**) and **label3**.

In short, objects inherit all the settings defined above them in the current section, and later settings override earlier settings.

## Comments

We can (and often do) add comments to the Asterisk configuration files. Comments help make the configuration files easier to read, and can also be used to temporarily disable certain settings.



## Comments on a Single Line

Single-line comments begin with the semicolon (;) character. The Asterisk configuration parser treats everything following the semicolon as a comment. To expand on our previous example:

```
[section-name]
setting=true

[another_section]
setting=false ; this is a comment
; this entire line is a comment
;awesome=true
; the semicolon on the line above makes it a
; comment, disabling the setting
```

## Block Comments

Asterisk also allows us to create block comments. A block comment is a comment that begins on one line, and continues for several lines. Block comments begin with the character sequence

```
!--
```

and continue across multiple lines until the character sequence

```
--;
```

is encountered. The block comment ends immediately after `--;` is encountered.

```
[section-name]
setting=true
!-- this is a block comment that begins on this line
and continues across multiple lines, until we
get to here --;
```

# Using The include, tryinclude and exec Constructs

## include, tryinclude and exec



You might have arrived here looking for [Include Statements](#) specific to Asterisk dialplan.

There are two other constructs we can use within all of our configuration files. They are **#include** and **#exec**.

The **#include** construct tells Asterisk to read in the contents of another configuration file, and act as though the contents were at this location in this configuration file. The syntax is **#include filename**, where **filename** is the name of the file you'd like to include. This construct is most often used to break a large configuration file into smaller pieces, so that it's more manageable. The asterisk/star character will be parsed in the path, allowing for the inclusion of an entire directory of files. If the target file specified does not exist, then Asterisk will not load the module that contains configuration with the **#include** directive.

The **#tryinclude** construct is the same as **#include** except it won't stop Asterisk from loading the module when the target file does not exist.

The **#exec** takes this one step further. It allows you to execute an external program, and place the output of that program into the current configuration file. The syntax is **#exec program**, where **program** is the name of the program you'd like to execute.

The **#exec**, **#include**, and **#tryinclude** constructs do not work in the following configuration files:

- asterisk.conf
- modules.conf



### Enabling #exec Functionality

The **#exec** construct is not enabled by default, as it has some risks both in terms of performance and security. To enable this functionality, go to the **asterisk.conf** configuration file (by default located in */etc/asterisk*) and set **execincludes=yes** in the **[options]** section. By default both the **[options]** section heading and the **execincludes=yes** option have been commented out, you you'll need to remove the semicolon from the beginning of both lines.

## Examples

Let's look at example of both constructs in action. This is a generic example meant to illustrate the syntax usage inside a configuration file.

```
[section-name]
setting=true
#include otherconfig.conf      ; include another configuration file
#include my_other_files/*.conf ; include all .conf files in the subdirectory
my_other_files
#exec otherprogram            ; include output of otherprogram
```

You can use **#tryinclude** if there is any chance the target file may not exist and you still want Asterisk to load the configuration for the module.

Here is a more realistic example of how **#exec** might be used with real-world commands.

```
#exec /usr/bin/curl -s http://example.com/mystuff > /etc/asterisk/mystuff
#include mystuff
```

## Adding to an existing section

If you want to add settings to an existing section of a configuration file (either later in the file, or when using the **#include** and **#exec** constructs), add a plus sign in parentheses after the section heading, as shown below:

```
[section-name]
setting1=value1

[section-name](+)
setting2=value2
```

This example shows that the **setting2** setting was added to the existing section of the configuration file.

## Templates

Another construct we can use within most Asterisk configuration files is the use of templates. A template is a section of a configuration file that is only used as a base (or template, as the name suggests) to create other sections from.

## Template Syntax

To define a section as a template **only** (not to be loaded for use as configuration by itself), place an exclamation mark in parentheses after the section heading, as shown in the example below.

```
[template-name](!)
setting=value
```

Alternatively the [Using Templates](#) page will also discuss how to have a section inherit another section's settings without defining a template. In effect, using an "active" or "live" configuration section as your template.

## Using Templates

To use a template when creating another section, simply put the template name in parentheses after the section heading name, as shown in the example below. If you want to inherit from multiple templates, use commas to separate the template names).

```
[template-name](!)
setting=value

[template-2](!)
setting2=value2

[not-a-template]
setting4=value4

[section-name](template-name,template-2,not-a-template)
setting3=value3
```

This works even when the section name referenced in parentheses is **not defined as a template** as in the case of the "not-a-template" section.

The newly-created section will inherit all the values and objects defined in the template(s), as well as any new settings or objects defined in the newly-created section. The settings and objects defined in the newly-created section override settings or objects of the same name from the templates. Consider this example:

```
[test-one](!)
permit=192.168.0.2
host=alpha.example.com
deny=192.168.0.1

[test-two](!)
permit=192.168.1.2
host=bravo.example.com
deny=192.168.1.1

[test-three](test-one,test-two)
permit=192.168.3.1
host=charlie.example.com
```

The [test-three] section will be processed as though it had been written in the following way:

```
[test-three]
permit=192.168.0.2
host=alpha.example.com
deny=192.168.0.1
permit=192.168.1.2
host=bravo.example.com
deny=192.168.1.1
permit=192.168.3.1
host=charlie.example.com
```

### chan\_sip Template Example

Here is a more extensive and realistic example from the chan\_sip channel driver's sample configuration file.

```

[basic-options](!)           ; a template
    dtmfmode=rfc2833
    context=from-office
    type=friend

[natted-phone](!,basic-options) ; another template inheriting basic-options
    nat=yes
    directmedia=no
    host=dynamic

[public-phone](!,basic-options) ; another template inheriting basic-options
    nat=no
    directmedia=yes

[my-codecs](!)               ; a template for my preferred codecs
    disallow=all
    allow=ilbc
    allow=g729
    allow=gsm
    allow=g723
    allow=ulaw

[ulaw-phone](!)              ; and another one for ulaw-only
    disallow=all
    allow=ulaw

; and finally instantiate a few phones
;
; [2133](natted-phone,my-codecs)
;     secret = peekaboo
; [2134](natted-phone,ulaw-phone)
;     secret = not_very_secret
; [2136](public-phone,ulaw-phone)
;     secret = not_very_secret_either

```



# Database Support Configuration

Top-level page for information about Database support.

# Realtime Database Configuration

- *Introduction*
  - *Two modes: Static and Realtime*
  - *Realtime SIP friends*
  - *Realtime H.323 friends*
  - *New function in the dial plan: The Realtime Switch*
  - *Capabilities*
  - *Configuration in extconfig.conf*
  - *Limitations*
  - *FreeTDS supported with connection pooling*
  - *Notes on use of the sipregs family*

## Introduction

The Asterisk Realtime Architecture is a new set of drivers and functions implemented in Asterisk.

The benefits of this architecture are many, both from a code management standpoint and from an installation perspective.

The ARA is designed to be independent of storage. Currently, most drivers are based on SQL, but the architecture should be able to handle other storage methods in the future, like LDAP.

The main benefit comes in the database support. In Asterisk v1.0 some functions supported MySQL database, some PostgreSQL and other ODBC. With the ARA, we have a unified database interface internally in Asterisk, so if one function supports database integration, all databases that has a realtime driver will be supported in that function.

Currently there are three realtime database drivers:

1. ODBC: Support for UnixODBC, integrated into Asterisk The UnixODBC subsystem supports many different databases, please check [www.unixodbc.org](http://www.unixodbc.org) for more information.
2. MySQL: Native support for MySQL, integrated into Asterisk
3. PostgreSQL: Native support for Postgres, integrated into Asterisk

## Two modes: Static and Realtime

The ARA realtime mode is used to dynamically load and update objects. This mode is used in the SIP and IAX2 channels, as well as in the voicemail system. For SIP and IAX2 this is similar to the v1.0 MYSQL\_FRIENDS functionality. With the ARA, we now support many more databases for dynamic configuration of phones.

The ARA static mode is used to load configuration files. For the Asterisk modules that read configurations, there's no difference between a static file in the file system, like extensions.conf, and a configuration loaded from a database.

You just have to always make sure the var\_metric values are properly set and ordered as you expect in your database server if you're using the static mode with ARA (either sequentially or with the same var\_metric value for everybody).

If you have an option that depends on another one in a given configuration file (i.e. 'musiconhold' depending on 'agent' from agents.conf) but their var\_metric are not sequential you'll probably get default values being assigned for those options instead of the desired ones. You can still use the same var\_metric for all entries in your DB, just make sure the entries are recorded in an order that does not break the option dependency.

That doesn't happen when you use a static file in the file system. Although this might be interpreted as a bug or limitation, it is not.



To use static realtime with certain core configuration files (e.g. features.conf, cdr.conf, cel.conf, indications.conf, etc.) the realtime backend you wish to use must be preloaded in modules.conf.

```
[modules]
preload => res_odbc.so
preload => res_config_odbc.so
```

## Realtime SIP friends

The SIP realtime objects are users and peers that are loaded in memory when needed, then deleted. This means that Asterisk currently can't handle voicemail notification and NAT keepalives for these peers. Other than that, most of the functionality works the same way for realtime friends as for the ones in static configuration.

With caching, the device stays in memory for a specified time. More information about this is to be found in the sip.conf sample file.

If you specify a separate family called "sipregs" SIP registration data will be stored in that table and not in the "sippeers" table.

## Realtime H.323 friends

Like SIP realtime friends, H.323 friends also can be configured using dynamic realtime objects.

## New function in the dial plan: The Realtime Switch

The realtime switch is more than a port of functionality in v1.0 to the new architecture, this is a new feature of Asterisk based on the ARA. The realtime

switch lets your Asterisk server do database lookups of extensions in realtime from your dial plan. You can have many Asterisk servers sharing a dynamically updated dial plan in real time with this solution.

Note that this switch does NOT support Caller ID matching, only extension name or pattern matching.

### **Capabilities**

The realtime Architecture lets you store all of your configuration in databases and reload it whenever you want. You can force a reload over the AMI, Asterisk Manager Interface or by calling Asterisk from a shell script with

```
asterisk -rx "reload"
```

You may also dynamically add SIP and IAX devices and extensions and making them available without a reload, by using the realtime objects and the realtime switch.

### **Configuration in extconfig.conf**

You configure the ARA in extconfig.conf (yes, it's a strange name, but it was defined in the early days of the realtime architecture and kind of stuck).

The part of Asterisk that connects to the ARA use a well defined family name to find the proper database driver. The syntax is easy:

```
<family> => <realtime driver>,<res_<driver>.conf class name>[,<table>]
```

The options following the realtime driver identified depends on the driver.

Defined well-known family names are:

- sippeers, sipusers - SIP peers and users
- sipregs - SIP registrations
- iaxpeers, iaxusers - IAX2 peers and users
- voicemail - Voicemail accounts
- extensions - Realtime extensions (switch)
- meetme - MeetMe conference rooms
- queues - Queues
- queue\_members - Queue members
- musiconhold - Music On Hold classes
- queue\_log - Queue logging

Voicemail storage with the support of ODBC described in [ODBC Voicemail Storage](#).

### **Limitations**

Currently, realtime extensions do not support realtime hints. There is a workaround available by using func\_odbc. See the sample func\_odbc.conf for more information.

### **FreeTDS supported with connection pooling**

In order to use a FreeTDS-based database with realtime, you need to turn connection pooling on in res\_odbc.conf. This is due to a limitation within the FreeTDS protocol itself. Please note that this includes databases such as MS SQL Server and Sybase. This support is new in the current release.

You may notice a performance issue under high load using UnixODBC. The UnixODBC driver supports threading but you must specifically enable threading within the UnixODBC configuration file like below for each engine:

```
Threading = 2
```

This will enable the driver to service many requests at a time, rather than serially.

### **Notes on use of the sipregs family**

The community provided some additional recommendations on the JIRA issue [ASTERISK-21315](#):

- It is a good idea to avoid using sipregs altogether by NOT enabling it in extconfig. Using a writable sipusers table should be enough. If you cannot write to your base sipusers table because it is readonly, you could consider making a separate sipusers view that joins the readonly table with a writable sipregs table.

## SIP Realtime, MySQL table structure

Here is the table structure used by MySQL for Realtime SIP friends

```
#
# Table structure for table `sipfriends`
#

CREATE TABLE IF NOT EXISTS `sipfriends` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(10) NOT NULL,
  `ipaddr` varchar(15) DEFAULT NULL,
  `port` int(5) DEFAULT NULL,
  `regseconds` int(11) DEFAULT NULL,
  `defaultuser` varchar(10) DEFAULT NULL,
  `fullcontact` varchar(35) DEFAULT NULL,
  `regserver` varchar(20) DEFAULT NULL,
  `useragent` varchar(20) DEFAULT NULL,
  `lastms` int(11) DEFAULT NULL,
  `host` varchar(40) DEFAULT NULL,
  `type` enum('friend','user','peer') DEFAULT NULL,
  `context` varchar(40) DEFAULT NULL,
  `permit` varchar(40) DEFAULT NULL,
  `deny` varchar(40) DEFAULT NULL,
  `secret` varchar(40) DEFAULT NULL,
  `md5secret` varchar(40) DEFAULT NULL,
  `remotesecret` varchar(40) DEFAULT NULL,
  `transport` enum('udp','tcp','udp,tcp','tcp,udp') DEFAULT NULL,
  `dtmfmode` enum('rfc2833','info','shortinfo','inband','auto') DEFAULT NULL,
  `directmedia` enum('yes','no','nonat','update') DEFAULT NULL,
  `nat` enum('yes','no','never','route') DEFAULT NULL,
  `callgroup` varchar(40) DEFAULT NULL,
  `pickupgroup` varchar(40) DEFAULT NULL,
  `language` varchar(40) DEFAULT NULL,
  `allow` varchar(40) DEFAULT NULL,
  `disallow` varchar(40) DEFAULT NULL,
  `insecure` varchar(40) DEFAULT NULL,
  `trustripid` enum('yes','no') DEFAULT NULL,
  `progressinband` enum('yes','no','never') DEFAULT NULL,
  `promiscredir` enum('yes','no') DEFAULT NULL,
  `useclientcode` enum('yes','no') DEFAULT NULL,
  `accountcode` varchar(40) DEFAULT NULL,
  `setvar` varchar(40) DEFAULT NULL,
  `callerid` varchar(40) DEFAULT NULL,
  `amaflags` varchar(40) DEFAULT NULL,
  `callcounter` enum('yes','no') DEFAULT NULL,
  `busylevel` int(11) DEFAULT NULL,
  `allowoverlap` enum('yes','no') DEFAULT NULL,
  `allowsubscribe` enum('yes','no') DEFAULT NULL,
  `videosupport` enum('yes','no') DEFAULT NULL,
  `maxcallbitrate` int(11) DEFAULT NULL,
  `rfc2833compensate` enum('yes','no') DEFAULT NULL,
  `mailbox` varchar(40) DEFAULT NULL,
  `session-timers` enum('accept','refuse','originate') DEFAULT NULL,
  `session-expires` int(11) DEFAULT NULL,
  `session-minse` int(11) DEFAULT NULL,
  `session-refresher` enum('uac','uas') DEFAULT NULL,
  `t38pt_usertpsource` varchar(40) DEFAULT NULL,
  `regexten` varchar(40) DEFAULT NULL,
  `fromdomain` varchar(40) DEFAULT NULL,
  `fromuser` varchar(40) DEFAULT NULL,
  `qualify` varchar(40) DEFAULT NULL,
  `defaulttip` varchar(40) DEFAULT NULL,
  `rtptimeout` int(11) DEFAULT NULL,
  `rtpholdtimeout` int(11) DEFAULT NULL,
  `sendripid` enum('yes','no') DEFAULT NULL,
  `outboundproxy` varchar(40) DEFAULT NULL,
  `callbackextension` varchar(40) DEFAULT NULL,
  `registertrying` enum('yes','no') DEFAULT NULL,
  `timert1` int(11) DEFAULT NULL,
  `timerb` int(11) DEFAULT NULL,
  `qualifyfreq` int(11) DEFAULT NULL,
  `constantssrc` enum('yes','no') DEFAULT NULL,
  `contactpermit` varchar(40) DEFAULT NULL,
  `contactdeny` varchar(40) DEFAULT NULL,
  `useregphone` enum('yes','no') DEFAULT NULL,
  `textsupport` enum('yes','no') DEFAULT NULL,
  `faxdetect` enum('yes','no') DEFAULT NULL,
  `buggymwi` enum('yes','no') DEFAULT NULL,
  `auth` varchar(40) DEFAULT NULL,
  `fullname` varchar(40) DEFAULT NULL,
  `trunkname` varchar(40) DEFAULT NULL,
  `cid_number` varchar(40) DEFAULT NULL,
  `callingpres`
enum('allowed_not_screened','allowed_passed_screen','allowed_failed_screen','allowed','prohib_not_screened','prohib_passed_screen',
'prohib_failed_screen','prohib') DEFAULT NULL,
  `mohinterpret` varchar(40) DEFAULT NULL,

```

```
`mohsuggest` varchar(40) DEFAULT NULL,  
`parkinglot` varchar(40) DEFAULT NULL,  
`hasvoicemail` enum('yes','no') DEFAULT NULL,  
`subscribemwi` enum('yes','no') DEFAULT NULL,  
`vmexten` varchar(40) DEFAULT NULL,  
`autoframing` enum('yes','no') DEFAULT NULL,  
`rtpkeepalive` int(11) DEFAULT NULL,  
`call-limit` int(11) DEFAULT NULL,  
`g726nonstandard` enum('yes','no') DEFAULT NULL,  
`ignorespdversion` enum('yes','no') DEFAULT NULL,  
`allowtransfer` enum('yes','no') DEFAULT NULL,  
`dynamic` enum('yes','no') DEFAULT NULL,  
PRIMARY KEY (`id`),  
UNIQUE KEY `name` (`name`),  
KEY `ipaddr` (`ipaddr`,`port`),
```

```
KEY `host` (`host`,`port`)  
) ENGINE=MyISAM;
```

## Sorcery



Under Construction



Top-level page for sorcery related configuration information.

## Sorcery Caching

Since Asterisk 12, Asterisk has had a generic data access/storage layer called "sorcery", with pluggable "wizards" that each create, retrieve, update, and delete data from various backends. For instance, there is a sorcery wizard that reads configuration data from .conf files. There is a sorcery wizard that uses the Asterisk Realtime Architecture to interface with databases and other alternative backends. There are also sorcery wizards that use the AstDB and a simple in-memory container.

Starting in Asterisk 13.5.0, a new "memory\_cache" wizard has been created. This allows for a cached copy of an object to be stored locally in cases where retrieval from a remote backend (such as a relational database) might be expensive. Memory caching is a flexible way to provide per object type caching, meaning that you are not forced into an all-or-nothing situation if you decide to cache. Caching also provides configuration options to allow for cached entries to automatically be updated or expired.

## Cachable Objects

Not all configurable objects are managed by sorcery. The following is a list of objects that are managed by the sorcery subsystem in Asterisk.

- PJSIP endpoint
- PJSIP AOR
- PJSIP contact
- PJSIP identify
- PJSIP ACL
- PJSIP resource\_list
- PJSIP phoneprov
- PJSIP registration
- PJSIP subscription\_persistence
- PJSIP inbound-publication
- PJSIP asterisk-publication
- PJSIP system
- PJSIP global
- PJSIP auth
- PJSIP outbound-publish
- PJSIP transport
- External MWI mailboxes

### On this Page

- [Cachable Objects](#)
- [When Should I Use Caching?](#)
- [How do I enable Caching?](#)
- [How does the cache behave?](#)
  - [name](#)
  - [maximum\\_objects](#)
  - [object\\_lifetime\\_maximum](#)
  - [object\\_lifetime\\_stale](#)
  - [expire\\_on\\_reload](#)
- [What AMI and CLI commands does the cache provide?](#)
  - [CLI](#)
  - [AMI](#)
- [What are some caching strategies?](#)
  - [Hands-on or hands-off?](#)
  - [Expire or Stale?](#)
- [An example configuration](#)

## When Should I Use Caching?

First, if you are using default sorcery backends for objects (i.e. you have not altered `sorcery.conf` at all), then caching will likely not have any positive effect on your configuration. However, if you are using the "realtime" sorcery wizard or any other that retrieves data from outside the Asterisk process, then caching could be a good fit for certain object types.

Caching works well for values that are

- Read more often than they are written
- Retrieved one-at-a-time.

For the first point, you will be able to know this better than anyone else. For instance, if you tend to configure PJSIP authentication very infrequently, but there are many calls, subscriptions, and qualifies that require authentication, then caching PJSIP auths is probably a good idea. If you are constantly tweaking PJSIP endpoint configuration for some reason, then you might find that caching isn't necessarily as good a fit for PJSIP endpoints.

For the second point, it may not always be obvious which types of objects are typically looked up one-at-a-time and which ones are typically looked up in multiples. The following object types are likely a bad fit for caching since they tend to be looked up in multiples:

- PJSIP contact
- PJSIP identify



- PJSIP global
- PJSIP system
- PJSIP registrations
- PJSIP ACLs
- PJSIP outbound-publishes
- PJSIP subscription\_persistence

The rest of the objects listed are most typically retrieved one-at-a-time and would be good for caching.

## How do I enable Caching?

If you are familiar with enabling realtime for a sorcery object, then enabling caching should not seem difficult. Here is an example of what it might look like if you have configured PJSIP endpoints to use a cache:

```

sorcery.conf

[res_pjsip]
endpoint/cache=memory_cache
endpoint=realtime,ps_endpoints
```

Let's break this down line-by-line. The first line starts with "endpoint/cache". "endpoint" is the name of the object type. "/cache" is a cue to sorcery that the wizard being specified on this line is a cache. And "memory\_cache" is the name of the caching wizard that has been added in Asterisk 14.0.0. The second line is the familiar line that specifies that endpoints can be retrieved from realtime by following the "ps\_endpoints" configuration line in `extconfig.conf`.

The order of the lines is important. You will want to specify the memory\_cache wizard before the realtime wizard so that the memory\_cache is looked in before realtime when retrieving an item.

## How does the cache behave?

By default, the cache will simply store objects in memory. There will be no limits to the number of objects stored in the cache, and the items in the cache will never be updated or expire, no matter whether the backend has been updated to have new configuration values. The cache entry in `sorcery.conf` is configurable, though, so you can modify the behavior to suit your setup. Options for the memory cache are comma-separated on the line in `sorcery.conf` that defines the cache. For instance, you might have something like the following:

```

sorcery.conf

[res_pjsip]
endpoint/cache = memory_cache,maximum_objects=150,expire_on_reload=yes,object_lifetime_maximum=3600
endpoint = realtime,ps_endpoints
```

The following configuration options are recognized by the memory cache:

### name

The name of a cache is used when referring to a specific cache when running an AMI or CLI command. If no name is provided for a cache, then the default is `<configuration section>/<object type>`. PJSIP endpoints, for instance, have a default cache name of "res\_pjsip/endpoint".

### maximum\_objects

This option specifies the maximum number of objects that can be in the cache at a given time. If the cache is full and a new item is to be added, then the oldest item in the cache is removed to make room for the new item. If this option is not set or if its value is set to 0, then there is no limit on the number of objects in the cache.

### object\_lifetime\_maximum

This option specifies the number of seconds an object may occupy the cache before it is automatically removed. This time is measured from when the object is initially added to the cache, not the time when the object was last accessed. If this option is not set or if its value is set to 0, then objects will stay in the cache forever.

### object\_lifetime\_stale

This option specifies the number of seconds an object may occupy the cache until it is considered stale. When a stale object is retrieved from the cache, the stale object is given to the requestor, and a background task is initiated to update the object in the cache by querying whatever backend stores are configured. If a new object is retrieved from the backend, then the stale cached object is replaced with the new object. If the backend no longer has an object with the same ID as the one that has become stale, then the stale object is removed from the cache. If this option is not set or if its value is 0, then objects in the cache will never be marked stale.

### expire\_on\_reload

This option specifies whether a reload of a module should automatically remove all of its objects from the cache. For instance, if this option is enabled, and you are caching PJSIP endpoints, then a module reload of `res_pjsip.so` would clear all PJSIP endpoints from the cache. By default this option is not

enabled.

## What AMI and CLI commands does the cache provide?

### CLI

#### ***sorcery memory cache show <cache name>***

This CLI command displays the configuration for the given cache and tells the number of items currently in the cache.

#### ***sorcery memory cache dump <cache name>***

This CLI command displays all objects in the given cache. In addition to the name of the object, the command also displays the number of seconds until the object becomes stale and the number of seconds until the object will be removed from the cache.

#### ***sorcery memory cache expire <cache name> [object name]***

This CLI command is used to remove objects from a given cache. If no object name is specified, then all objects in the cache are removed. If an object name is specified, then only the specified object is removed.

#### ***sorcery memory cache stale <cache name> [object\_name]***

This CLI command is used to mark an item in the cache as stale. If no object name is specified, then all objects in the cache are marked stale. If an object name is specified, then only the specified object is marked stale. For information on what it means for an object to be stale, see [here](#)

### AMI



Since AMI commands are XML-documented in the source, there should be a dedicated wiki page with this information.

#### ***SorceryMemoryCacheExpireObject***

This command has the following syntax:

```
Action: SorceryMemoryCacheExpireObject
Cache: <cache name>
Object: <object name>
```

Issuing this command will cause the specified object in the specified cache to be removed. Like all AMI commands, an optional ActionID may be specified.

#### ***SorceryMemoryCacheExpire***

This command has the following syntax:

```
Action: SorceryMemoryCacheExpire
Cache: <cache name>
```

Issuing this command will cause all objects in the specified cache to be removed. Like all AMI commands, an optional ActionID may be specified.

#### ***SorceryMemoryCacheStaleObject***

This command has the following syntax:

```
Action: SorceryMemoryCacheStaleObject
Cache: <cache name>
Object: <object name>
```

Issuing this command will cause the specified object in the specified cache to be marked as stale. For more information on what it means for an object to be stale, see [here](#). Like all AMI commands, an optional ActionID may be specified.

#### ***SorceryMemoryCacheStale***

This command has the following syntax:

```
Action: SorceryMemoryCacheStale
Cache: <cache name>
```

Issuing this command will cause all objects in the specified cache to be marked as stale. For more information on what it means for an object to be stale, see [here](#). Like all AMI commands, an optional ActionID may be specified.

## What are some caching strategies?

### Hands-on or hands-off?

The hands-on approach to caching is that you set your cache to have no maximum number of objects, and objects never expire or become stale on their own. Instead, whenever you make changes to the backend store, you issue an API or CLI command to remove objects or mark them stale. The hands-off approach to caching is to fine-tune the maximum number of objects, stale timeout, and expire timeout such that you never have to think about the cache again after you set it up the first time.

The hands-on approach is a good fit either for installations where configuration rarely changes, or where there is some automation involved when configuration changes are made. For instance, if you are setting up a PBX for a small office where you are likely to make configuration changes a few times a year, then the hands-on approach may be a good fit. If your configuration is managed through a GUI that fires off a script when the "submit" button is pressed, then the hands-on approach may be a good fit since your scripts can be modified to manually expire objects or mark them stale. The main disadvantage to the hands-on approach is that if you forget to manually expire a cached object or if you make a mistake in your tooling, you're likely to have some big problems since configuration changes will seemingly not have any effect.

The hands-off approach is a good fit for configurations that change frequently or for deployments with inconsistent usage among users. If configuration is changing frequently, then it makes sense for objects in the cache to become stale and automatically get refreshed. If you have some users on the system that maybe use the system once a week, it makes sense for them to get removed from the cache as more frequent users occupy it. The biggest disadvantage to the hands-off approach is the potential for churn if your settings are overzealous. For instance, if you allow a maximum of 15 objects in a cache but it's common for 20 to be used, then the cache may constantly be shuffling which objects are stored in it. Similarly, if you set a stale object timeout low, then it is possible that objects in the cache will frequently be replacing themselves with identical copies.

There is also a hybrid approach. In the hybrid approach, you're mostly hands-off, but you can be hands-on for "emergency" changes. For instance, if there is a misconfiguration that is resulting in calls not being able to be sent to a user, then you may want to get that configuration updated and immediately remove the cached object so that the new configuration can be added to the cache instead.

### Expire or Stale?

One question that may enter your mind is whether to have objects expire or whether they should become stale.

Letting objects expire has the advantage that they no longer are occupying cache space. For objects that are infrequently accessed, this can be a good thing since they otherwise will be taking up space and being useless. For objects that are accessed frequently, expiration is likely a bad choice. This is because if the object has been removed from the cache, then attempting to retrieve the object will require a cache miss, followed by a backend hit to retrieve the object. If the object configuration has not been altered, then this equates to a waste of cycles.

Letting objects become stale has the advantage that retrievals will always be quick. This is because even if the object is stale, the stale cached object is returned. It's left up to a background task to update the cached object with new data from the backend. The main disadvantage to objects being stale is that infrequently accessed objects will remain in the cache long after their useful lifetime.

One approach to take is a hybrid approach. You can set objects to become stale after an amount of time, and then later, the object will become expired. This way, objects that are retrieved frequently will stay up to date as they become stale, and objects that are rarely accessed will expire after a while.

## An example configuration

Below is a sample `sorcery.conf` file that uses realtime as the backend store for some PJSIP objects.

```
sorcery.conf  
  
[res_pjsip]  
endpoint/cache = memory_cache,object_lifetime_stale=600,object_lifetime_maximum=1800,expire_on_reload=yes  
endpoint = realtime,ps_endpoints  
auth/cache=memory_cache,expire_on_reload=yes  
auth = realtime,ps_auths  
aor/cache = memory_cache,object_lifetime_stale=1500,object_lifetime_maximum=1800,expire_on_reload=yes  
aor = realtime,ps_aors
```

In this particular setup, the administrator has set different options for different object caches.

- For endpoints, the administrator decided that cached endpoint configuration may occasionally need updating. Endpoints therefore will be marked stale after 10 minutes. If an endpoint happens to make it 30 minutes without being retrieved, then the endpoint will be ejected from the cache entirely.
- For auths, the administrator realized that auth so rarely changes that there is no reason to set any sort of extra parameters. On those odd occasions where auth is updated, the admin will just manually expire the old auth.
- AORs, like endpoints, may require refreshing after a while, but because the AOR configurations are changed much more infrequently, it takes 25 minutes for the object to become stale.
- All objects expire on a reload since a reload likely means that there was some large-scale change and everything should start from scratch.

This is just an example. It is not necessarily going to be a good fit for everyone's needs.

# Asterisk Internal Database

Asterisk comes with a database that is used internally and made available for Asterisk programmers and administrators to use as they see fit.

Asterisk versions up to 1.8 used the Berkeley DB, and **in version 10** the project moved to the **SQLite3 database**. You can read about database migration between those major versions in the section [SQLite3 astdb back-end](#).

## Purpose of the internal database

The database really has two purposes:

1. Asterisk uses it to store information that needs to persist between reloads/restarts. Various modules use it for this purpose automatically.
2. Users can use it to store arbitrary data. This is done using a variety of dialplan applications and functions such as:
  - Functions:
    - [DB](#)
    - [DB\\_DELETE](#)
    - [DB\\_EXISTS](#)
    - [DB\\_KEYS](#)
  - Application: [DBdeltree](#)

The functions and applications for Asterisk 11 are linked above, but you should look at the documentation for the version you have deployed.

## Database commands on the CLI

Sub-commands under the command "database" allow a variety of functions to be performed on or with the database.

```
*CLI> core show help database
database del           -- Removes database key/value
database deltree      -- Removes database keytree/values
database get          -- Gets database value
database put          -- Adds/updates database value
database query        -- Run a user-specified query on the astdb
database show         -- Shows database contents
database showkey      -- Shows database contents
```

## SQLite3 astdb back-end



Starting with **Asterisk 10**, Asterisk uses **SQLite3** for its internal database instead of the Berkeley DB database used by Asterisk 1.8 and previous versions.

Every effort has been made to make this transition as automatic and painless for users as possible. This page will describe the upgrade process, any potential problems, and the appropriate solutions to those problems.

### The upgrade process

Asterisk 10 will attempt to upgrade any existing old-style Berkeley DB internal database to the new SQLite 3 database format. This conversion process is accomplished at run-time with the `astdb2sqlite3` utility which builds by default in Asterisk 10. The `astdb2sqlite3` utility will also be forcibly built even if deselected using `menuselect` if the build process determines that there is an old-style Berkeley DB and no new SQLite3 DB exists.

When Asterisk 10 is run, as part of the initialization process it checks for the existence of the SQLite3 database. If it doesn't exist and an old-style Berkeley DB does exist, it will attempt to convert the Berkeley DB to the SQLite3 format. If no existing database exists, a new SQLite 3 database will be created. If the conversion fails, a warning will be displayed with instructions describing possible fixes and Asterisk will exit.



It is important that you perform the upgrade process at the same permission level that you expect Asterisk to run at. For example, if you upgrade as root, but run Asterisk as a user with lower permissions, the SQLite3 database created as part of the upgrade will not be able to be accessed by Asterisk.

### Troubleshooting an upgrade

#### Symptoms

***./configure displays the warning: \*\*\* Please install the SQLite3 development package.***

#### Cause

To build Asterisk 10, the SQLite 3 development libraries must be installed.

#### Solution

On Debian-based distros including Ubuntu, these libraries may be installed by running `'sudo apt-get install libsqlite3-dev'`. For Red Hat-based distros including Fedora and Centos these libraries may be installed by running (as root) `'yum install sqlite3-devel'`.

***Asterisk exits displaying the warning: \*\*\* Database conversion failed!***

**Cause**

Asterisk 10 could not find the `astdb2sqlite3` utility to convert the old Berkeley DB to SQLite 3.

**Solution**

Make sure that `astdb2sqlite3` is selected for build in the Utilities section when running 'make menuselect'. Be sure to re-run 'make' and 'make install' after selecting `astdb2sqlite3` for build.

**Cause**

Asterisk is unable to write to the directory specified in `asterisk.conf` as the 'astdbdir'

**Solution**

SQLite 3 creates a journal file in the 'astdbdir' specified in `asterisk.conf`. It is important that this directory is writable by the user Asterisk runs as. This involves either modifying the permissions of the 'astdbdir' directory listed in `asterisk.conf`, or changing the 'astdbdir' option to a directory for which the user running Asterisk already has write permission. This is generally only a problem if Asterisk is run as a non-root user.

**Cause**

If Asterisk 10 was installed via a distro-specific package, it is possible that the distro forgot to package the `astdb2sqlite3` utility.

**Solution**

Run 'which `astdb2sqlite3`' from a terminal. If no filenames are displayed, then `astdb2sqlite3` has not been installed. Check if the distro includes it in another asterisk related package, or download the Asterisk 10 source from [the Asterisk.org website](http://the Asterisk.org website) and follow the normal build instructions. Instead of running 'make install', manually run 'utils/`astdb2sqlite3` /var/lib/asterisk/astdb' from the Asterisk source directory, replacing '/var/lib/asterisk' with the 'astdbdir' directory listed in `asterisk.conf`. After the conversion, the distro-supplied Asterisk should successfully run.

## Migrating back from Asterisk 10 to Asterisk 1.8

If migrating back to Asterisk 1.8 from Asterisk 10, it is possible to convert the SQLite 3 internal database back to the Berkeley DB format that Asterisk 1.8 uses by using the `astdb2bdb` utility found in the `utils/` directory of the Asterisk 10 source. To build, make sure that `astdb2bdb` is selected in the Utilities section when running 'make menuselect'. Running 'utils/`astdb2bdb` /var/lib/asterisk/astdb.sqlite3' (replacing '/var/lib/asterisk' with the 'astdbdir' directory listed in `asterisk.conf`) will produce a file named 'astdb' in the current directory. Back up any existing `astdb` file in the `astdbdir` directory and replace it with the newly created `astdb` file.

# Key Concepts



Under Construction



Top-level page for a section dealing with concepts of the key moving pieces in Asterisk that an **administrator** needs to understand. Channels, Bridges, Frames, etc.

# Bridges

## Overview

In Asterisk, a bridge is the construct that shares media among [Channels](#). While a channel represents the path of communication between Asterisk and some device, a bridge is how that path of communication is shared. While channels are in a bridge, their media is exchanged in a manner dictated by the bridge's type. While we generally think of media being directed among channels, media can also be directed from Asterisk to the channels in a bridge. This can be the case in some conferences, where Music on Hold (MoH) or announcements are played for waiting channels.

### On this Page

## Creation

Generally, a bridge is created when Asterisk knows that two or more channels want to communicate. A variety of applications and API calls can cause a bridge to be created. Some of these include:

- [Dial](#) - a bridge is created for the two channels when the outbound channel answers. Both the inbound channel and the outbound channel are placed into the bridge.
  - DTMF feature invocations available from [Dial\(\)](#) can create, modify, or destroy bridges.
- [Bridge](#) - this directly creates a new bridge and places two channels into the bridge. Unlike [Dial](#), both channels have to already exist.
- [BridgeWait](#) (Asterisk 12+) - creates a special holding bridge and places a channel into it. Any number of channels may join the holding bridge, which can entertain them in a variety of ways.
- [MeetMe/ConfBridge](#) - both of these applications are used for conferencing, and can support multiple channels together in the same bridge.
- [Page](#) - a conferencing bridge (similar to [MeetMe/ConfBridge](#)) is used to direct the audio from the announcer to the many dialed channels.
- [Parking](#) (Asterisk 12+) - a special holding bridge is used for [Parking](#), which entertains the waiting channel with hold music.



### Asterisk 12+: Bridging Changed

In Asterisk 12, the bridging framework that [ConfBridge](#) was built on top of was extended to all bridges that Asterisk creates (with the exception of [MeetMe](#)). There are some new capabilities that this afforded Asterisk users; where applicable, this page will call out features that only apply to Asterisk 12 and later versions.

## Destruction

Channels typically leave a bridge when the application that created the bridge is terminated (such as a conference leader ending a [ConfBridge](#) conference) or when the other side hangs up (such as in a two-party bridge created by [Dial](#)). When channels leave a bridge they can continue doing what they were doing prior to entering the bridge, continue executing dialplan, or be hung up.

## Types

There are many types of bridges in Asterisk, each of which determine how the media is mixed between the participants of the bridge. In general, there are two categories of bridge types within Asterisk: two party and multiparty. Two party bridge variants include core bridges, local native bridges, and remote native bridges. Multiparty bridge variants include mixing and holding.



### Asterisk 12+: Bridges are Smart

In Asterisk 12, the bridging framework is smart! It will automatically choose the best mixing technology available based on the channels in the bridge and - if needed - it will dynamically change the mixing type of the bridge based on conditions that occur. For example, a two-party core bridge may turn into a multiparty bridge if an attended transfer converges into a three-way bridge via the `atxferthreeway` DTMF option.

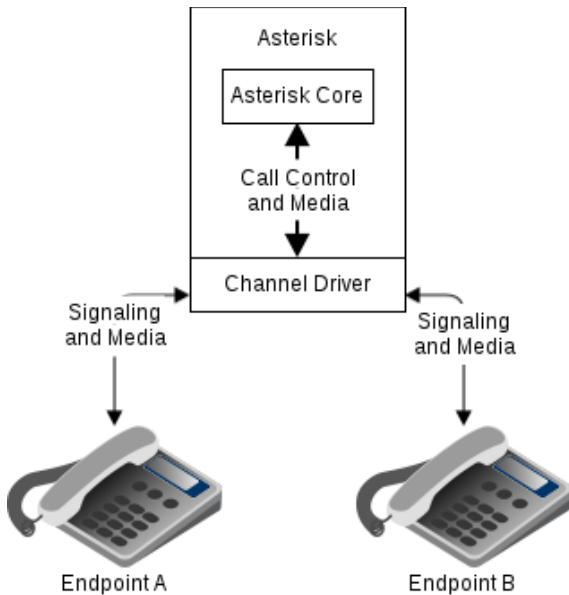
## Two-Party

A two-party bridge shares media between two channels. Because there are only two participants in the bridge, certain optimizations can take place, depending on the type of channels in the bridge. As such, there are "sub-types" of two-party bridges that Asterisk can attempt to use to improve performance.

### Core

A core bridge is the basic two-party bridge in Asterisk. Any channel of any type can communicate with any channel of any other type. A core bridge can perform media transcoding, media manipulation, call recording, DTMF feature execution, talk detection, and additional functionality because Asterisk has direct access to the media flowing between channels. Core bridges are the fallback when other types of bridging are not possible due to limiting network factors, configuration, or functionality requirements.





## Native

A native bridge occurs when both participants in a two-party bridge have similar channel technologies. When this occurs, Asterisk defers the transfer of media to the channel drivers/protocol stacks themselves, and simply monitors for the channels leaving the bridge (either due to hangup, time-out, or some other condition). Since media is handled in the channel drivers/protocol stacks, no transcoding, media manipulation, recording, DTMF, or other features depending on media interpretation can be done by Asterisk. The primary advantage to native bridging is higher performance.

The following channel technologies support native bridging:

- [RTP capable channel drivers](#) (such as SIP channels)
- [DAHDI channels](#)
- [IAX2 channels](#) (Asterisk 11-)

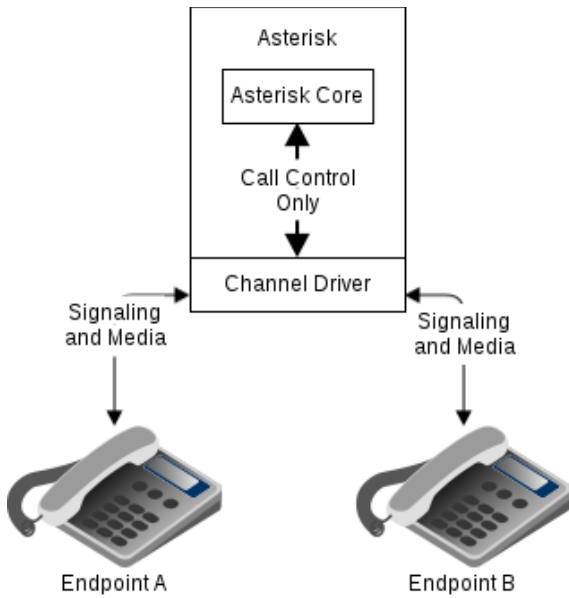


### **Asterisk 12+ IAX2 Native Bridging is Gone**

As it turned out, IAX2 native bridging was not much more efficient than a standard core bridge. In an IAX2 native bridge, the media must still be handled a good bit, i.e., placed into internal Asterisk frames. As such, when the bridging in Asterisk was converted to the new smart bridging framework, the IAX2 native bridge did not survive the transition.

## Local

A local native bridge occurs when the media between two channels is handled by the channel drivers/protocol stacks themselves, but the media is still sent from each device to Asterisk. In this case, Asterisk is merely proxying the media back and forth between the two devices. Most types of native bridging in Asterisk are local.

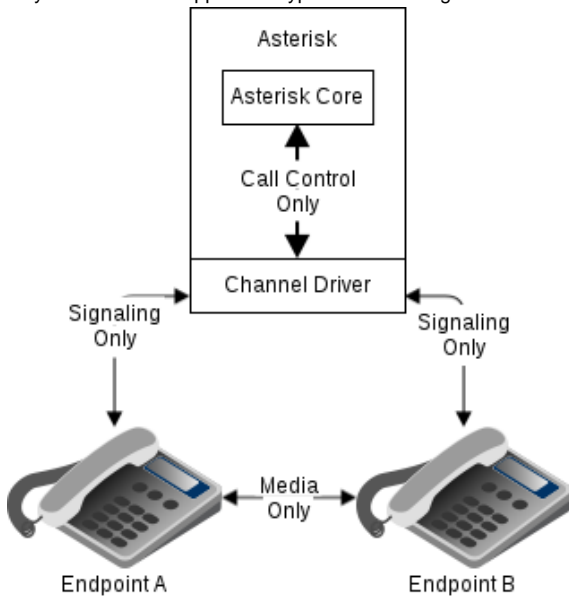


### Remote

A remote native bridge occurs when the media between two channels is redirected by Asterisk to flow directly between the two devices the channels talk to. When this occurs, the media is completely outside of Asterisk. With SIP channels, this is often called "direct media". Not surprisingly, since the media is flowing outside of Asterisk, this bridge has the best performance in Asterisk. However, it can only be used in certain circumstances:

- Both channels in the native bridge must support direct media.
- The devices communicating with Asterisk cannot be behind a NAT (or otherwise obscured with a private IP address that the other device cannot resolve).

Only SIP channels support this type of native bridge.



### Multiparty

Multiparty bridges interact with one or more channels and may route media among them. This can be thought of as an extension to two-party core bridging where media from multiple channels is merged or selected to be forwarded to the channels participating in the bridge. These bridges can have some, all, or none of the extended features of two-party core bridges depending on their intended use.

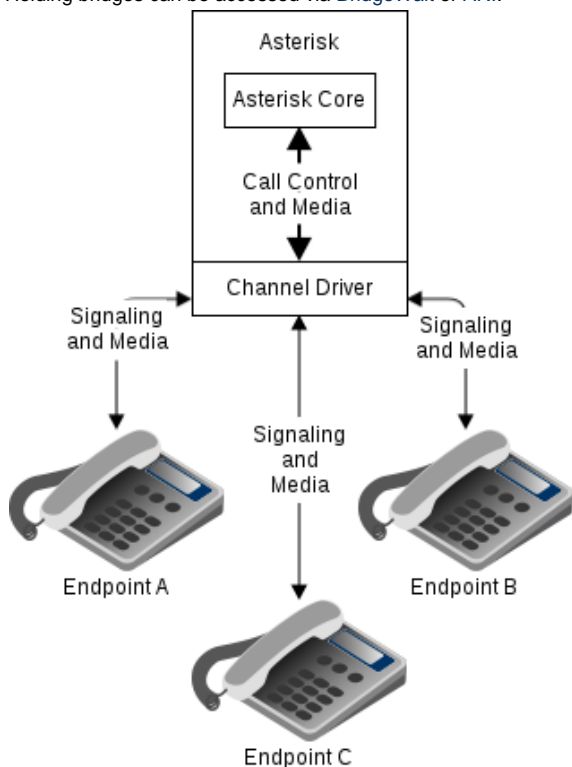
### Mixing

There are several ways to access mixing multiparty bridges:

- [MeetMe](#) - This is a legacy conference bridge application and relies on DAHDI. This type of conference is limited to narrow band audio.
- [ConfBridge](#) (Asterisk 11+) - This is a conference bridge application based that supports wide band mixing.
- Ad-hoc Multiparty Bridges (Asterisk 12+) - Some DTMF features like 3-way attended transfers can create multiparty bridges as necessary.

## Holding

Holding bridges are only available in Asterisk 12+ and provide a waiting area for channels which you may not yet be prepared to process or connect to other channels. This type of bridge prevents participants from exchanging media, can provide entertainment for all participants, and provides the ability for an announcer to interrupt entertainment with special messages as necessary. Entertainment for waiting channels can be MoH, silence, ringing, hold, etc.. Holding bridges can be accessed via [BridgeWait](#) or [ARI](#).



# Channels

## Asterisk Channels

Almost nothing happens in Asterisk without a channel being involved. A channel is an entity inside Asterisk that acts as a channel of communication between Asterisk and another device. That is, a phone, a PBX, another Asterisk system, or even Asterisk itself (in the case of a [local channel](#)).

Our documentation and many Asterisk users speak about channels in terms of "calls". A **call** can be one or more channels creating a path of communication or activity through the Asterisk system.

To give you an idea about what channels do, here are a few facts about them:

- [Channel Drivers](#) provide channels in Asterisk.
- Channels can interface with each other through [bridges](#).
- [Applications](#) and [functions](#) can affect the state or attributes of a channel or its media stream.
- Channels are commonly passing audio between communication endpoints, but can pass other data, such as video or text messages.
- Channels execute instructions with [dialplan](#), but can be controlled by other APIs (AGI,AMI,ARI) or interfaces (CLI).

## Common Asterisk Channels

One of the many benefits of Asterisk is the capability to interface with as many different technologies as you have channel drivers! However, most administrators will only make use of a few types at a time.

Here are a few commonly used [channel types](#):

- A SIP channel driver such as [chan\\_sip](#) or [chan\\_pjsip](#).
- DAHDI channels provided by [chan\\_dahdi](#).
- Local channels provided by [chan\\_local](#). (This was moved into the core in Asterisk 12)

**SIP channels** are used to interface with SIP capable VOIP devices, such as phones, channel banks, other PBXs or Internet Telephony Service Providers.

**DAHDI channels** are used to interface with DAHDI drivers and PRI libraries. In this case [chan\\_dahdi](#) allows you to use any DAHDI capable devices, such as Digium's line of T1/E1/J1 interface cards.

**Local channels** are used for dialing inward to the system itself, allowing any Asterisk component that can dial to call directly into [dialplan](#). This provides a sort of "fake" call that still executes real instructions.

- [Asterisk Channels](#)
- [Configuring Channels](#)
- [Using, Controlling and Routing Channels](#)
- [Inbound and Outbound Channels](#)
- [Channel Variable Inheritance](#)

## Configuring Channels

### Text File Configuration

Most channel drivers have an associated configuration file. Some channels may require the configuration of dependent resources for optimal operation. For example, SIP channels, configured in [sip.conf](#) or [pjsip.conf](#) use RTP resources which can be configured in [rtp.conf](#).

The [Channel Drivers](#) configuration section contains information on configuring documented channel drivers. In other cases the configuration file itself contains configuration documentation.

### Database Configuration

Flat text configuration isn't the only option. A few channel drivers provide support for the ARA ([Asterisk Realtime Architecture](#)) and can therefore pull configuration from a local or remote database. Use of the ARA requires configuration of additional resources and dependencies outside the channel drivers themselves.

## Using, Controlling and Routing Channels

Once you have a channel driver configured, how does it get used? When do channels get created?

Here are a few scenarios where a channel could get created:

- A device configured in the channel driver communicates to Asterisk (e.g. over a network) that it wants to make a call.
- A user executes a command (such as [Originate](#)) to create a new channel.
- An existing channel executes [dialplan](#) that calls an application (such as [Dial](#)) to create a new channel.
- Asterisk receives [API](#) calls that create a new channel or channels.

Once a channel is established, the events that occur are channel technology-dependent. That is, whether audio, video or other data communication begins over the channel will depend on signaling that occurs over SIP, ISDN, H.323 or other protocols implemented via the channel driver.

When Asterisk has established a channel, Asterisk will use a combination of channel driver configuration and dialplan instruction to determine how the channel behaves. On top of that Asterisk can communicate with external programs synchronously or asynchronously to receive API calls for channel inspection, direction or manipulation.

Once channels are established and communicating between devices and Asterisk; where that data flows to depends on the channel type itself, its overall configuration, device specific configuration, signaling sent by the originating mechanism (a device, a command, an API call) and associated [bridges](#). One or more channels could be talking to one or more channels over various bridges. What specifically Asterisk talks to on each channel is limited only by the technology implemented by the channel driver.

## Inbound and Outbound Channels

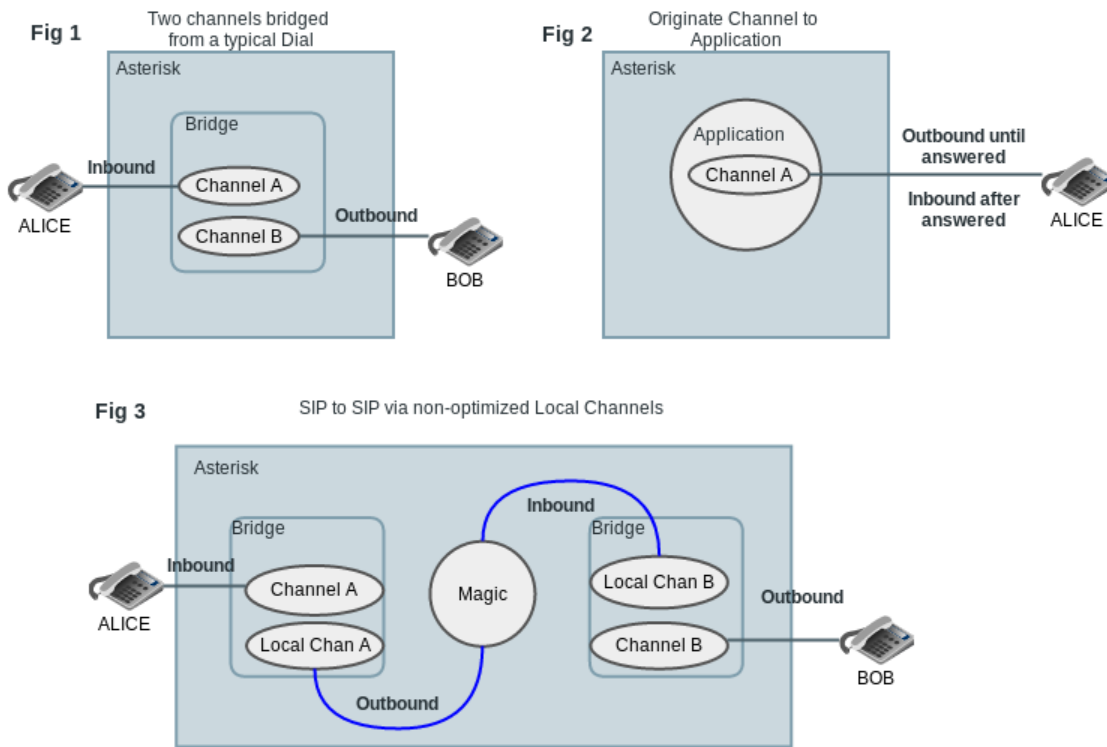
Often in our documentation, troubleshooting and development discussions you'll see mention of inbound or outbound channels. It'll be helpful to define what that means here.

**Inbound channels** are channels created when things outside of Asterisk call **into** Asterisk. This is typically the channel executing [Dialplan](#).

**Outbound channels** are channels created when Asterisk is calling **out** to something outside Asterisk.

The primary exception is with [Local Channels](#). In the case of local channels, you'll typically have two local channel legs, one that is treated as outbound and the other as inbound. In this case both are really inside Asterisk, but one is executing dialplan and the other is not. The leg executing dialplan is the one treated as inbound.

Below we'll diagram a few examples for clarity.



The figures have been kept somewhat generic and should apply to most channel types. Each figure shows the final state of the call, rather than a sequence of events.

Below are explanations of the various figures.

### Fig 1

One phone dials another phone; about as simple as it gets.

The **inbound** channel is created from Alice's phone calling Asterisk. Asterisk then calls the extension dialed by Alice by creating an **outbound** channel to talk to Bob. Once the call is established the two channels are put into a [bridge](#).

### Fig 2

A user runs the [originate](#) command from [AMI](#), or maybe something like "channel originate SIP/Alice application playback demo-congrats" from the [CLI](#).

Asterisk creates an **outbound** channel to call the device specified (SIP/Alice). When answered, Asterisk begins treating the channel like an **inbound** channel and connects it to the specified dialplan application.

### Fig 3

Perhaps a user runs originate again - but this time "channel originate SIP/Alice extension dialbob@internal" from the CLI. Where dialbob@internal contains dialplan telling Asterisk to dial outbound to SIP/Bob. At first, the created **outbound** channel would look like Fig 2 where it begins to be treated as **inbound** after the device answers the call. At that point, a number of things happen:

- Asterisk creates an outbound local channel into Asterisk and bridges it with the now inbound channel to Alice.
- Asterisk creates another leg of local channel as "inbound" into Asterisk to execute the dialplan at the extension specified with the originate. This local channel is essentially bridged with some magic to the other local channel.
- In our case the dialplan executes something like Dial(SIP/Bob), so the new SIP channel is created to communicate with SIP/Bob and is then bridged with the inbound local channel. Now communication flows across the whole path.

For this example demonstrating relationships between channels and other elements we used non-optimized local channels. If the local channels are optimized, then they will optimize themselves out of this mix and Alice and Bob's channels would be bridged together directly.

## Channel Variable Inheritance

When working with channels you'll almost certainly be touching channel variables. It is useful to note that upon setting a channel variable the level of inheritance between channels can be defined. This is discussed in the [Channel Variables](#) sub-section [Variable Inheritance](#).

## Frames



Under Construction



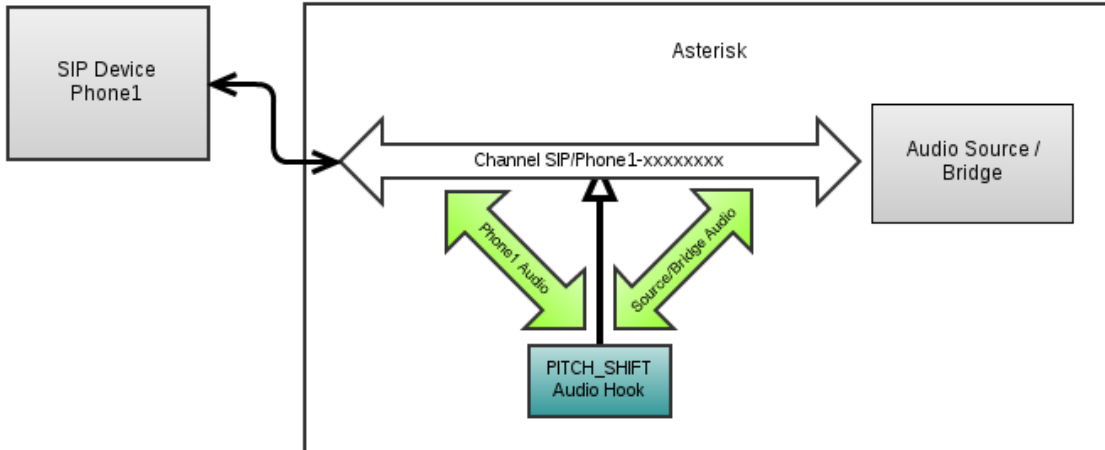
Top-level page for talking about frames, frame-hooks/audio-hooks.

# Audiohooks

## Overview

Certain applications and functions are capable of attaching what is known as an audiohook to a channel. In order to understand what this means and how to handle these applications and functions, it is useful to understand a little of the architecture involved with attaching them.

## Introduction - A Simple Audiohook



In this simple example, a SIP phone has dialed into Asterisk and its channel has invoked a function (`pitch_shift`) which has been set to cause all audio sent and received to have its pitch shifted higher (i.e. if the audio is voice, the voices will sound squeaky sort of like obnoxious cartoon chipmunks). The following dialplan provides a more concrete usage:

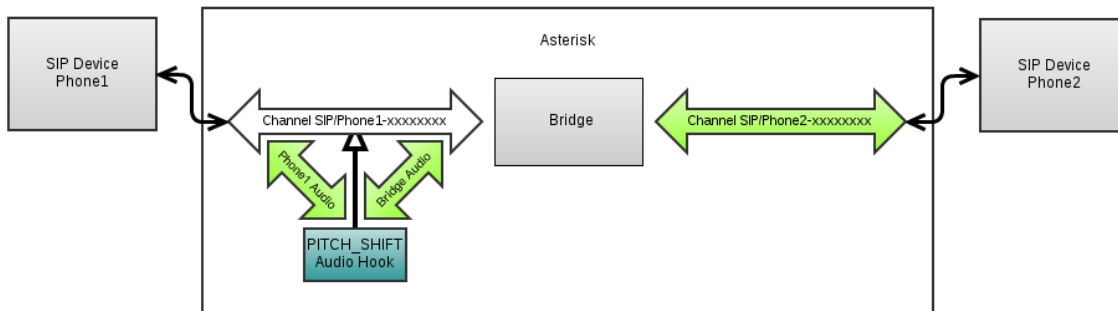
```
exten => 1,1,Answer()  
exten => 1,n,Set(PITCH_SHIFT(both)=higher)  
exten => 1,n,Voicemail(501)
```

When a phone calls this extension, it will be greeted by a higher pitched version of the voicemail prompt and then the speaker will leave a message for 501. The sound going from the phone to voicemail will also be higher pitched than what was actually said by the person who left the message.

Right now a serious minded Asterisk user reading this example might think something along the lines of "So what, I don't have any use for making people using my phone system sound like squirrels." However, audiohooks provide a great deal of the functionality for other applications within Asterisk including some features that are very business minded (listening in on channels, recording phone calls, and even less spy-guy type things like adjusting volume on the fly)

It's important to note that audiohooks are bound to the channel that they were invoked on. They don't apply to a call (a call is actually a somewhat nebulous concept in general anyway) and so one shouldn't expect audiohooks to follow other channels around just because audio that those channels are involved with touches the hook. If the channel that created the audiohook ceases to be involved with an audio stream, the audiohook will also no longer be involved with that audio stream.

## Attended Transfers and AUDIOHOOK\_INHERIT





```
exten => 1,1,Answer( )
exten => 1,n,MixMonitor(training_recording.wav)
exten => 1,n,Queue(techsupport)
```

Imagine the following scenario. An outside line calls into an Asterisk system to enter a tech support queue. When the call starts this user hears something along the lines of "Thank you for calling, all calls will be recorded for training purposes", so naturally MixMonitor will be used to record the call. The first available agent answers the call and can't quite seem to provide a working solution to the customer's problem, so he attempts to perform an attended transfer to someone with more expertise on the issue. The user gets transferred, and the rest of the call goes smoothly, but... ah nuts. The recording stopped for some reason when the agent transferred the customer to the other user. And why didn't this happen when he blind transferred a customer the other day?

The reason MixMonitor stopped is because the channel that owned it died. An Asterisk admin might think something like "That's not true, the mixmonitor was put on the customer channel and it's still there, I can still see it's name is the same and everything." and it's true that it seems that way, but attended transfers in particular cause what's known as a channel masquerade. Yes, its name and everything else about it seems like the same channel, but in reality the customer's channel has been swapped for the agent's channel and died since the agent hung up. The audiohook went with it. Under normal circumstances, administrators don't need to think about masquerades at all, but this is one of the rare instances where it gets in the way of desired behavior. This doesn't affect blind transfers because they don't start the new dialog by having the person who initiated the transfer bridging to the end recipient.

Working around this problem is pretty easy though. Audiohooks are not swapped by default when a masquerade occurs, unlike most of the relevant data on the channel. This can be changed on a case by case basis though with the AUDIOHOOK\_INHERIT dialplan function.

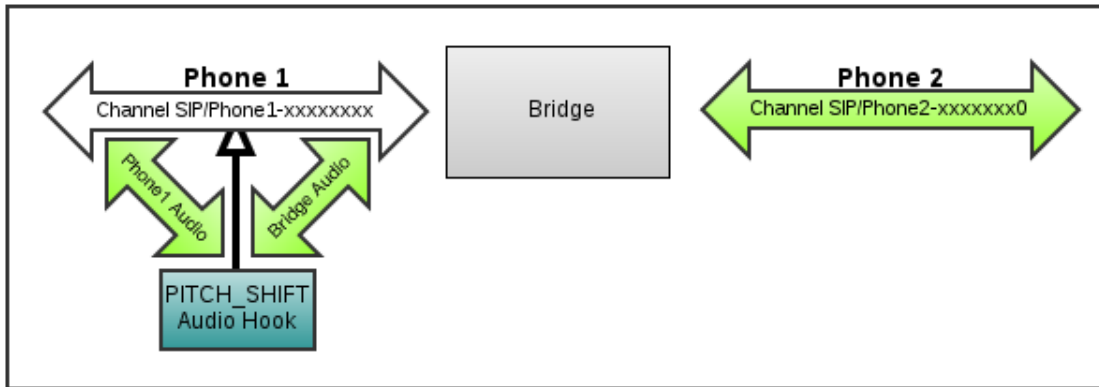
Using AUDIOHOOK\_INHERIT only requires that AUDIOHOOK\_INHERIT(source)=yes is set where source is the name given for the source of the audiohook. For more information on the sources available, see the description of the source argument in the documentation for AUDIOHOOK\_INHERIT.

So to fix the above example so that mixmonitor continues to record after the attended transfer, only one extra line is needed.

```
exten => 1,1,Answer( )
exten => 1,n,MixMonitor(training_recording.wav)
exten => 1,n,Set(AUDIOHOOK_INHERIT(MixMonitor)=yes)
exten => 1,n,Queue(techsupport)
```

Below is an illustrated example of how the masquerade process impacts an audiohook (in the case of the example, PITCH\_SHIFT)

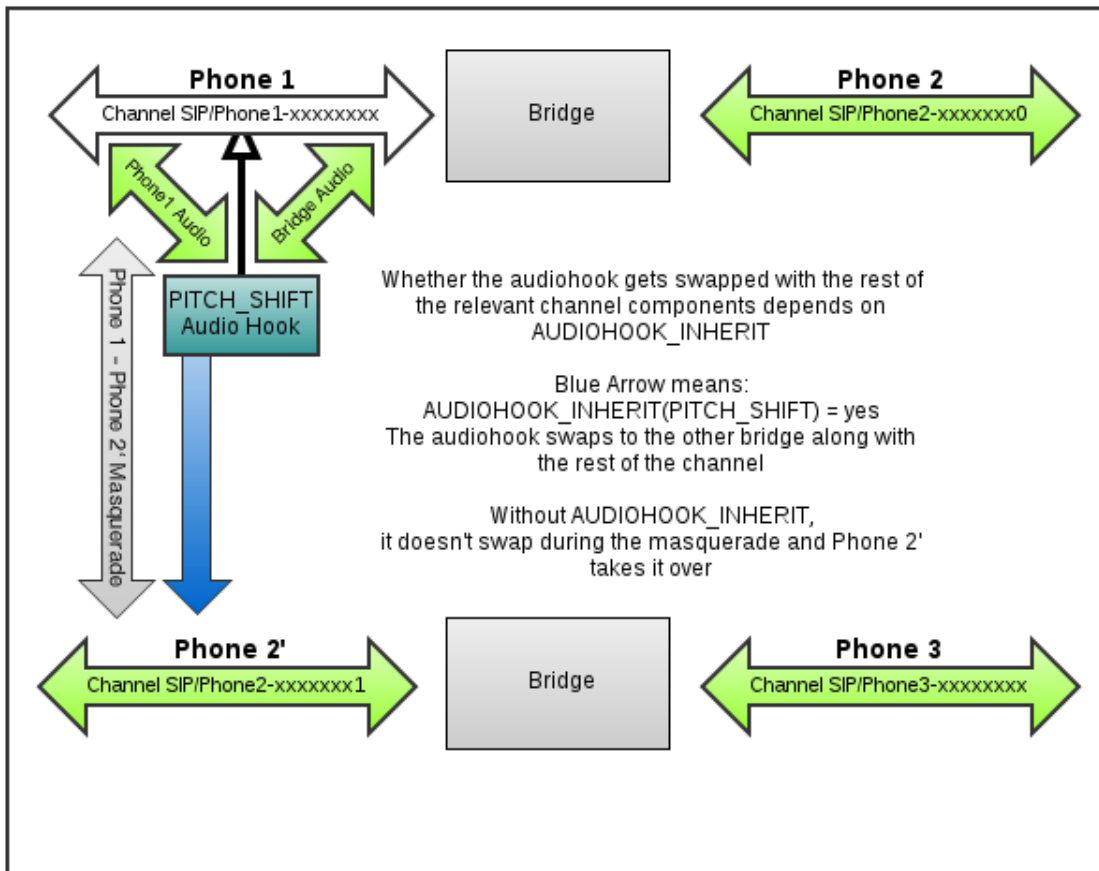
**Initial Call Setup**



**SIP/Phone2 attempts starts to attended transfer SIP/Phone1 to SIP/Phone3**

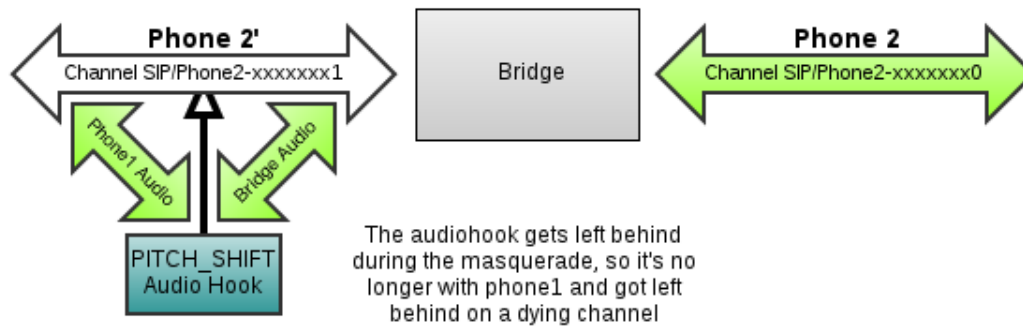


**Phone 2 hangs up on Phone 3, initiating the transfer. This requires a masquerade.**



**Bridges after Transfer: Without AUDIOHOOK\_INHERIT(PITCH\_SHIFT)**

After the masquerade, the bridge consists of Phone2's two channels talking to each other. Phone 2 has already hung up, so this dialog will be ending nearly immediately.

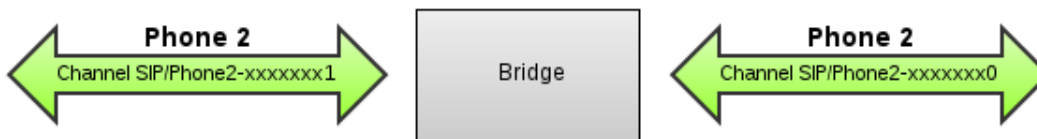


Phone 1 lost the audio hook because it didn't get swapped in the masquerade

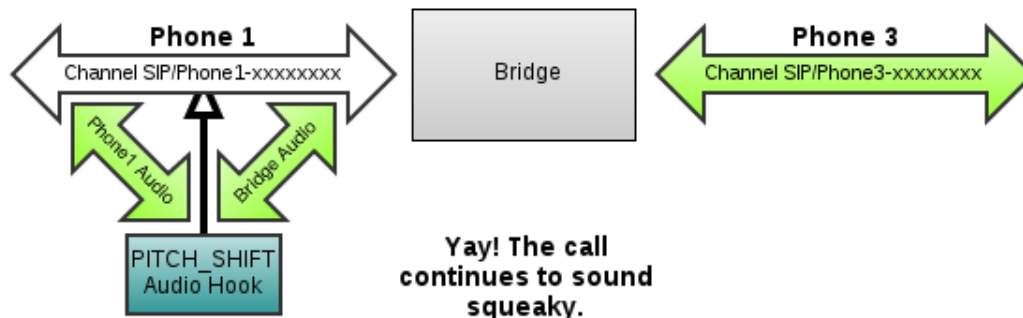


#### Bridges after Transfer: With AUDIOHOOK\_INHERIT(PITCH\_SHIFT)

Again, this bridge is still just phone2 talking to itself now. Phone 2 already hung up and this bridge is in the process of ending



Since AUDIOHOOK\_INHERIT was enabled, the audiohook came along with Phone1's channel.



Inheritance of audiohooks can be turned off in the same way by setting AUDIOHOOK\_INHERIT(source)=no.

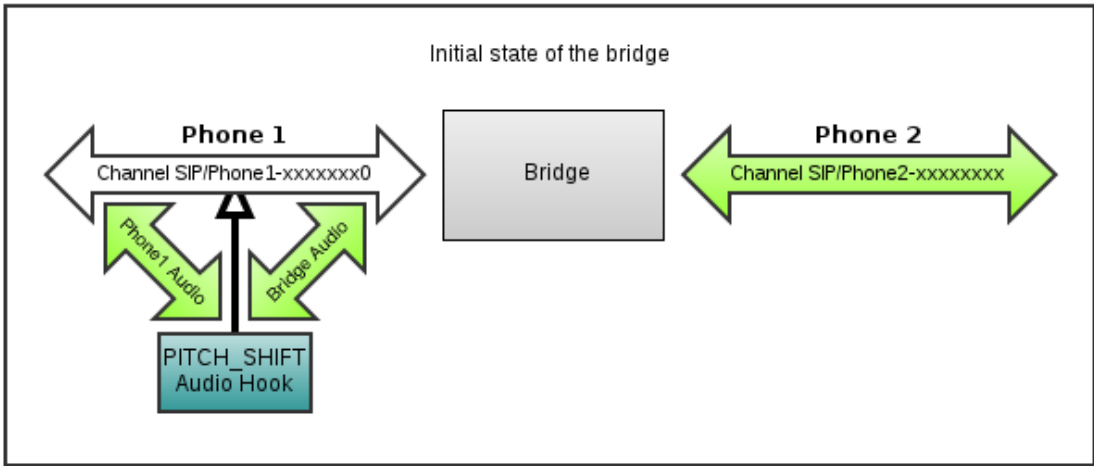
## Audiohook Sources

Audiohooks have a source name and can come from a number of sources. An up to date list of possible sources should always be available from the documentation for AUDIOHOOK\_INHERIT.

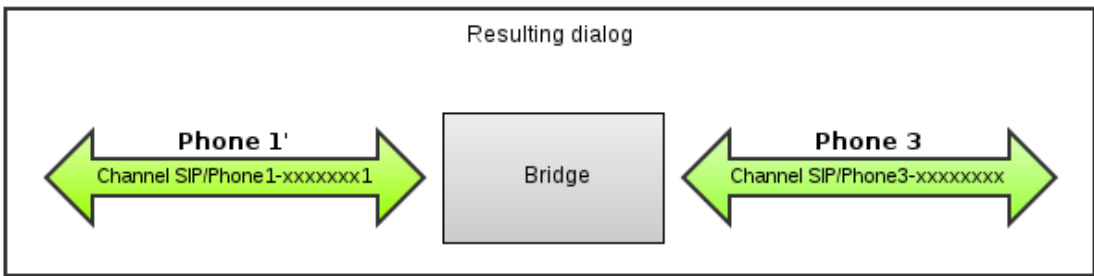
- Chanspy - from app\_chanspy
- MixMonitor - app\_mixmonitor.c
- Volume - func\_volume.c
- Mute - res\_mutestream.c
- Speex - func\_speex.c
- pitch\_shift - func\_pitchshift.c
- JACK\_HOOK - app\_jack.c

## Limitations for transferring Audiohooks

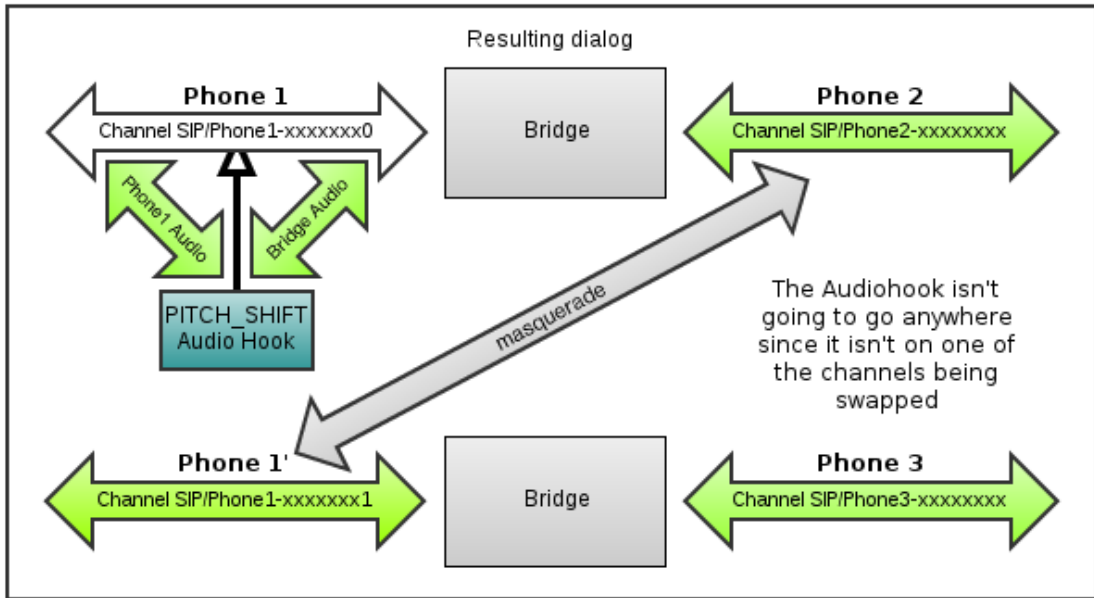
Even with audiohook inheritance set, the MixMonitor is still bound to the channel that invoked it. The only difference in this case is that with this option set, the audiohook won't be left on the discarded channel through the masquerade. This option doesn't enable a channel running mixmonitor to transfer the MixMonitor to another channel or anything like that. The dialog below illustrates why.



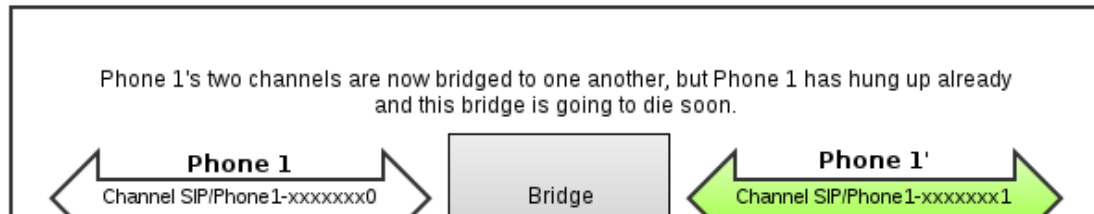
Phone 1 starts an attended transfer to Phone 3

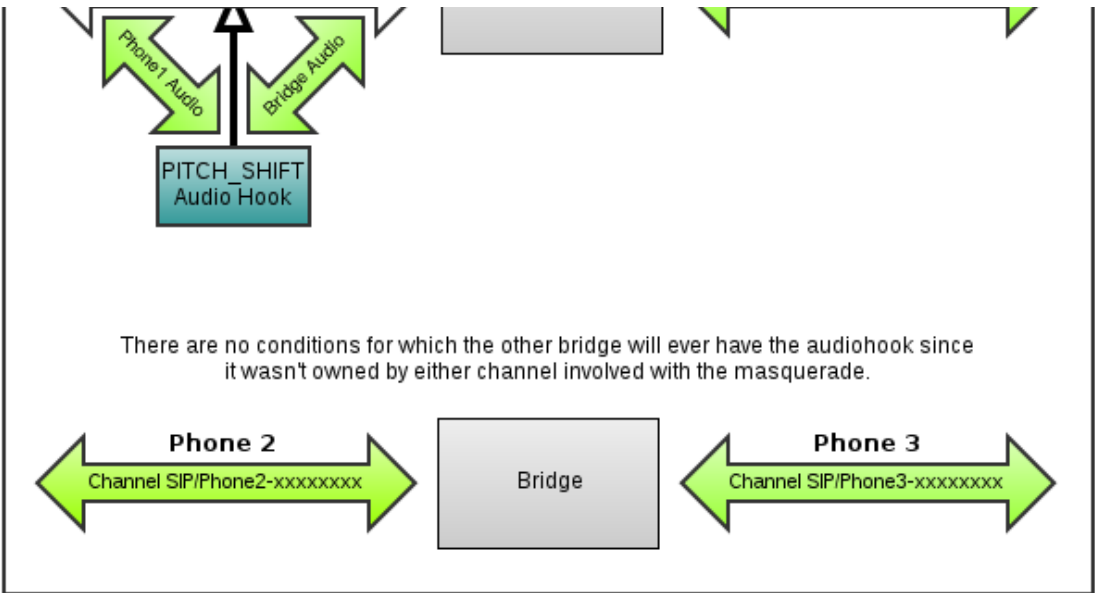


Phone 1 hangs up on Phone 3 initiating the masquerade



Final status of the bridges





# States and Presence

Asterisk includes the concepts of [Device State](#) , [Extension State](#) and [Presence State](#) which together allow Asterisk applications and interfaces to receive information about the state of devices, extensions and the users of the devices.

As an example, channel drivers like `chan_sip` or `res_pjsip/chan_pjsip` may both provide devices with device state, plus allow devices to [subscribe to hints](#) to receive notifications of state change. Other examples would be `app_queue` which takes into consideration the device state of queue members to influence queue logic or the Asterisk Manager Interface which provides actions for [querying extension state](#) and [presence state](#).

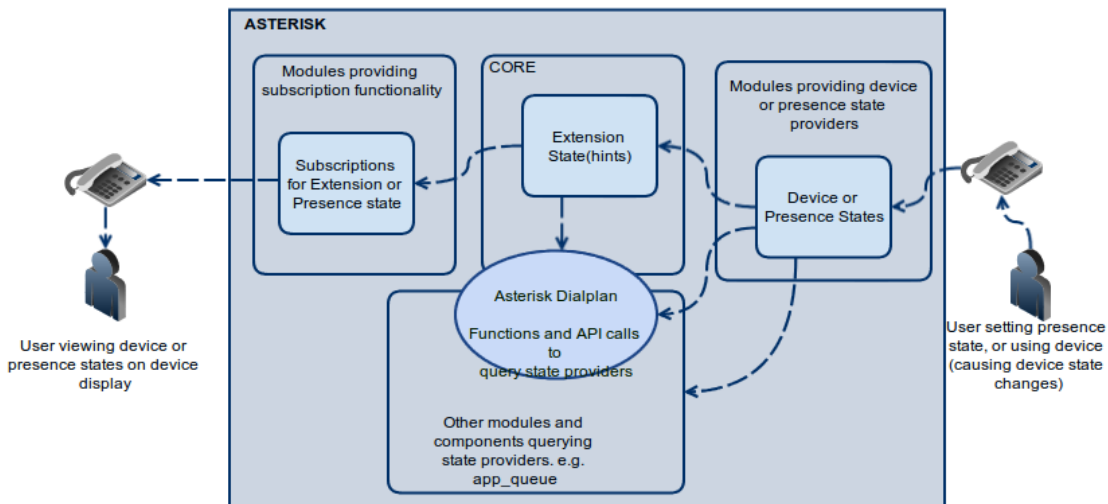
In This Section
<ul style="list-style-type: none"><li>• <a href="#">Device State</a></li><li>• <a href="#">Extension State and Hints</a></li><li>• <a href="#">Presence State</a></li><li>• <a href="#">Querying and Manipulating State</a></li></ul>

See Also
<a href="#">Distributed Device State</a>
<a href="#">Publishing Extension State</a>
<a href="#">Exchanging Device and Mailbox State Using PJSIP</a>

Additionally, modules exist for [Corosync](#) and [XMPP PubSub](#) support to allow device state to be [shared and distributed](#) across multiple systems.

The sub-sections here describe these concepts, point to related module specific configuration sections and discuss [Querying and Manipulating State](#) in formation.

The figure below may help you get an idea of the overall use of states and presence with the Asterisk system. It has been simplified to focus on the flow and usage of state and presence. In reality, the architecture can be a bit more confusing. For example a module could both provide subscription functionality for a subscriber and be the same module providing the devices and device state on the other end.



# Device State

## Devices

Devices are discrete components of functionality within Asterisk that serve a particular task. A device may be a channel technology resource, such as SIP/<name> in the case of chan\_sip.so or a feature resource of another module such as app\_confbridge.so which provides devices like confbridge:<name>.

### State information

Asterisk devices make state information available to the Asterisk user, such that a user might make use of the information to affect call flow or behavior of the Asterisk system. The device state identifier for a particular device is typically very similar to the device name. For example the device state identifier for SIP/6001 would be SIP/6001, for confbridge 7777 it would be confbridge:7777. Device states have a one-to-one mapping to the device they represent. That is opposed to other state providers in Asterisk which may have one-to-many relationships, such as Extension State.

The [Querying and Manipulating State](#) section covers how to access or manipulate device state as well as other states.

## Common Device State Providers

*Device state providers* are the components of Asterisk that provide some state information for their resources. The device state providers available in Asterisk will depend on the version of Asterisk you are using, what modules you have installed and how those modules are configured. Here is a list of the common *device state identifiers* you will see and what Asterisk component provides the resources and state.

On this Page
<ul style="list-style-type: none"><li>• <a href="#">Devices</a><ul style="list-style-type: none"><li>• <a href="#">State information</a></li></ul></li><li>• <a href="#">Common Device State Providers</a><ul style="list-style-type: none"><li>• <a href="#">Custom Device States</a></li></ul></li><li>• <a href="#">Possible Device States</a></li><li>• <a href="#">Module Specific Device State</a></li></ul>

Device State Identifier	Device State Provider
PJSIP/<resource>	PJSIP SIP stack, res_pjsip.so, chan_pjsip.so.
SIP/<resource>	The older SIP channel driver, chan_sip.so.
DAHDI/<resource>	The popular telephony hardware interface driver, chan_dahdi.so.
IAX2/<resource>	Inter-Asterisk Exchange protocol! chan_iax2.so.
ConfBridge:<resource>	The conference bridge application, app_confbridge.so.
MeetMe:<resource>	The older conference bridging app, app_meetme.so.
Park:<resource>	The Asterisk core in versions up to 11. res_parking.so in versions 12 or greater.
Calendar:<resource>	res_calendar.so and related calendaring modules.
Custom:<resource>	Custom device state provided by the asterisk core.

Note that we are not differentiating any device state providers based on what is on the far end. Depending on device state provider, the far end of signaling for state could be a physical device, or just a discrete feature resource inside of Asterisk. In terms of understanding device state for use in Asterisk, it doesn't really matter. The device state represents the state of the Asterisk device as long as it is able to provide it regardless of what is on the far end of the communication path.

### Custom Device States

The Asterisk core provides a *Custom* device state provider (custom:<resource>) that allows you to define arbitrary device state resources. See the [Querying and Manipulating State](#) section for more on using custom device state.

## Possible Device States

Here are the possible states that a device state may have.

- UNKNOWN
- NOT\_INUSE
- INUSE
- BUSY
- INVALID



- UNAVAILABLE
- RINGING
- RINGINUSE
- ONHOLD

Though the label for each state carries a certain connotation, the actual meaning of each state is really up to the device state provider. That is, any particular state may mean something different across device state providers.

## Module Specific Device State

There is module specific configuration that you must be aware of to get optimal behavior with certain state providers.

For **chan\_sip** see the [chan\\_sip State and Presence Options](#) section.

For **res\_pjsip** see the [Configuring res\\_pjsip for Presence Subscriptions](#) section.

# Extension State and Hints

## Overview

Extension state is the state of an Asterisk extension, as opposed to the direct state of a device or a user. It is the aggregate of [Device state](#) from devices mapped to the extension through a **hint** directive. See the [States and Presence](#) section for a diagram showing the relationship of all the various states.

Asterisk's SIP channel drivers provide facilities to allow SIP presence subscriptions ([RFC3856](#)) to extensions with a defined hint. With an active subscription, devices can receive notification of state changes for the subscribed to extension. That notification will take the form of a SIP NOTIFY with PIDF content ([RFC3863](#)) containing the presence/state information.

## Defining Hints

For Asterisk to store and provide state for an extension, you must first define a **hint** for that extension. Hints are defined in the [Asterisk dialplan](#), i.e. `extensions.conf`.

When Asterisk loads the configuration file it will create hints in memory for each hint defined in the dialplan. Those hints can then be [queried](#) or [manipulated](#) by functions and CLI commands. The state of each hint will regularly be updated based on state changes for any devices mapped to a hint.

The full syntax for a hint is

```
exten = <extension>,<hint>,<device state id>[& <more dev state id>,<presence state id>
```

Here is what you might see for a few configured hints.

```
[internal]

exten = 6001,hint,SIP/Alice&SIP/Alice-mobile
exten = 6002,hint,SIP/Bob
exten = 6003,hint,SIP/Charlie&DAHDI/3
exten = 6004,hint,SIP/Diane,CustomPresence:Diane
exten = 6005,hint,,CustomPresence:Ellen
```

Things of note:

- You may notice that the syntax for a hint is similar to a regular extension, except you use the **hint** keyword in place of the priority. Remember these special hint directives are used at load-time and not during run-time, so there is no need for a priority.
- Multiple devices can be mapped to an extension by providing an ampersand delimited list.
- A presence state ID is set after the device state IDs. If set with only a presence state provider you must be sure to include a blank field after the hint as in the example for extension 6005.
- Hints can be anywhere in the dialplan. Though, remember that dialplan referencing the extension and devices subscribing to it will need use the extension number/name and context. The hints shown above would be 6001@internal, 6002@internal, etc, just like regular extensions.

## Querying Extension State

The [Querying and Manipulating State](#) section covers accessing and affecting the various types of state.

For a quick CLI example, once you have defined some hints, you can easily check from the CLI to verify they get loaded correctly.

```
*CLI> core show hints
-- Registered Asterisk Dial Plan Hints --
      6003@internal      : SIP/Charlie&DAHDI/3   State:Unavailable      Watchers  0
      6002@internal      : SIP/Bob                State:Unavailable      Watchers  0
      6001@internal      : SIP/Alice&SIP/Alice-   State:Unavailable      Watchers  0
      6005@internal      : ,CustomPresence:Elle   State:Unavailable      Watchers  0
      6004@internal      : SIP/Diane,CustomPres   State:Unavailable      Watchers  0
-----
- 5 hints registered
```

In this example I was lazy, so they don't have real providers mapped otherwise you would see various states represented.

## SIP Subscription to Asterisk hints

Once a hint is configured, Asterisk's SIP drivers can be configured to allow SIP User Agents to subscribe to the hints. A subscription will result in state change notifications being sent to the subscriber.

Configuration for **chan\_sip** is discussed in [Configuring chan\\_sip for Presence Subscriptions](#)

Configuration for **res\_pjsip** is discussed in [Configuring res\\_pjsip for Presence Subscriptions](#)

# Presence State

## Overview

Asterisk 11 has been outfitted with support for presence states. An easy way to understand this is to compare presence state support to the device state support Asterisk has always had. Like with device state support, Asterisk has a core API so that modules can register themselves as presence state providers, alert others to changes in presence state, and query the presence state of others. The difference between the device and presence state concepts is made clear by understanding the subject of state for each concept.

- Device state reflects the current **state of a physical device** connected to Asterisk
- Presence state reflects the current **state of the user** of the device

For example, a **device** may currently be `not in use` but the **person** is `away`. This can be a critical detail when determining the availability of the **person**.

While the architectures of presence state and device state support in Asterisk are similar, there are some key differences between the two.

- Asterisk cannot infer presence state changes the same way it can device state changes. For instance, when a SIP endpoint is on a call, Asterisk can infer that the device is being used and report the device state as `in use`. Asterisk cannot infer whether a user of such a device does not wish to be disturbed or would rather chat, though. Thus, all presence state changes have to be manually enacted.
- Asterisk does not take presence into consideration when determining availability of a device. For instance, members of a queue whose device state is `busy` will not be called; however, if that member's device is `not in use` but his presence is `away` then Asterisk will still attempt to call the queue member.
- Asterisk cannot aggregate multiple presence states into a single combined state. Multiple device states can be listed in an extension's hint priority to have a combined state reported. Presence state support in Asterisk lacks this concept.

### On this Page

- [Overview](#)
- [Presence States](#)
- [Subtype and Message](#)
- [func\\_presencestate And The CustomPresence Provider](#)
- [Configuring Presence Subscription with Hints](#)
  - [Example Presence Notification](#)
- [Phone Support for Presence State via SIP presence notifications](#)
  - [Digium Phones](#)

## Presence States

- `not_set`: No presence state has been set for this entity.
- `unavailable`: This entity is present but currently not available for communications.
- `available`: This entity is available for communication.
- `away`: This entity is not present and is unable to communicate.
- `xa`: This entity is not present and is not expected to return for a while.
- `chat`: This entity is available to communicate but would rather use instant messaging than speak.
- `dnd`: This entity does not wish to be disturbed.

## Subtype and Message

In addition to the basic presence states provided, presence also has the concept of a **subtype** and a **message**.

The subtype is a brief method of describing the nature of the state. For instance, a subtype for the `away` status might be "at home".

The message is a longer explanation of the current presence state. Using the same `away` example from before, the message may be "Sick with the flu. Out until the 18th".

## func\_presencestate And The CustomPresence Provider

The only provider of presence state in Asterisk 11 is the `CustomPresence` provider. This provider is supplied by the `func_presencestate.so` module, which grants access to the `PRESENCE_STATE` dialplan function. The documentation for `PRESENCE_STATE` can be found [here](#). `CustomPresence` is device-agnostic within the core and can be a handy way to set and query presence from `dialplan`, or APIs such as the `AMI`.

A simple use case for `CustomPresence` in dialplan is demonstrated below.

```
[default]
exten => 2000,1,Answer()
same => n,Set(CURRENT_PRESENCE=${PRESENCE_STATE(CustomPresence:Bob,value)})
same => n,GotoIf(${CURRENT_PRESENCE}=available)?set_unavailable:set_available
same => n(set_unavailable),Set(PRESENCE_STATE(CustomPresence:Bob)=available,,)
same => n,Goto(finished)
same => n(set_unavailable),Set(PRESENCE_STATE(CustomPresence:Bob)=unavailable,,)
same => n(finished),Playback(queue-thankyou)
same => n,Hangup

exten => 2001,1,GotoIf(${PRESENCE_STATE(CustomPresence:Bob,value)}!=available)?voicemail
same => n,Dial(SIP/Bob)
same => n(voicemail)VoiceMail(Bob@default)
```

With this dialplan, a user can dial 2000@default to toggle Bob's presence between available and unavailable. When a user attempts to call Bob using 2001@default, if Bob's presence is currently not available then the call will go directly to voicemail.



One thing to keep in mind with the PRESENCE\_STATE dialplan function is that, like with DEVICE\_STATE, state may be queried from any presence provider, but PRESENCE\_STATE is only capable of setting presence state for the CustomPresence presence state provider.

## Configuring Presence Subscription with Hints

As is mentioned in the phone support section, at the time of writing this will only work with a Digium phone.

Like with device state, presence state is associated to a dialplan extension with a hint. Presence state hints come after device state in the hint extension and are separated by a comma (,). As an example:

```
[default]
exten => 2000, hint, SIP/2000, CustomPresence:2000
exten => 2000,1,Dial(SIP/2000)
same => n, Hangup()
```

Or alternatively, you could define the presence state provider without a device.

```
exten => 2000, hint, , CustomPresence:2000
```

The **first** example would allow for someone subscribing to the extension state of 2000@default to be notified of device state changes for device SIP/2000 as well as presence state changes for the presence provider CustomPresence:2000.

The **second** example would allow for the subscriber to receive notification of state changes for only the presence provider CustomPresence:2000.

The CustomPresence presence state provider will be discussed further on this page.

Also like with device state, there is an [Asterisk Manager Interface](#) command for querying presence state. Documentation for the AMI PresenceState command can be found [here](#).

### Example Presence Notification

When a SIP device is subscribed to a hint you have configured in Asterisk and that hint references a presence state provider, then upon change of that state Asterisk will generate a notification. That notification will take the form of a SIP NOTIFY including XML content. In the expanding panel below I've included an example of a presence notification sent to a Digium phone. This particular presence notification happened when we changed presence state for CustomPresence:6002 via the CLI command 'presencestate change'.

↕ [Click here to see the NOTIFY example](#)

```

myserver*CLI> presencestate change CustomPresence:6002 UNAVAILABLE
Changing 6002 to UNAVAILABLE
set_destination: Parsing <sip:6002@10.24.18.138:5060;ob> for address/port to send to
set_destination: set destination to 10.24.18.138:5060
Reliably Transmitting (no NAT) to 10.24.18.138:5060:
NOTIFY sip:6002@10.24.18.138:5060;ob SIP/2.0
Via: SIP/2.0/UDP 10.24.18.124:5060;branch=z9hG4bK68008251;rport
Max-Forwards: 70
From: sip:6002@10.24.18.124;tag=as722c69ec
To: "Bob" <sip:6002@10.24.18.124>;tag=4DpRzFRilaKU9iQcAME2APx85TgFOEN7
Contact: <sip:6002@10.24.18.124:5060>
Call-ID: JVoQfeZelcWTdPI5aTWkRpdqkjs8zmME
CSeq: 104 NOTIFY
User-Agent: Asterisk PBX SVN-branch-12-r413487
Subscription-State: active
Event: presence
Content-Type: application/pidf+xml
Content-Length: 602

<?xml version="1.0" encoding="ISO-8859-1"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
xmlns:pp="urn:ietf:params:xml:ns:pidf:person"
xmlns:es="urn:ietf:params:xml:ns:pidf:rpidd:status:rpidd-status"
xmlns:ep="urn:ietf:params:xml:ns:pidf:rpidd:status:rpidd-person"
entity="sip:6002@10.24.18.124">
  <pp:person><status>
  </status></pp:person>
  <note>Ready</note>
  <tuple id="6002">
    <contact priority="1">sip:6002@10.24.18.124</contact>
    <status><basic>open</basic></status>
  </tuple>
  <tuple id="digium-presence">
    <status>
    <digium_presence type="unavailable" subtype=""></digium_presence>
    </status>
  </tuple>
</presence>

---
  == Extension Changed 6002[from-internal] new state Idle for Notify User 6002

<--- SIP read from UDP:10.24.18.138:5060 --->
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.24.18.124:5060;rport=5060;received=10.24.18.124;branch=z9hG4bK68008251
Call-ID: JVoQfeZelcWTdPI5aTWkRpdqkjs8zmME
From: <sip:6002@10.24.18.124>;tag=as722c69ec
To: "Bob" <sip:6002@10.24.18.124>;tag=4DpRzFRilaKU9iQcAME2APx85TgFOEN7
CSeq: 104 NOTIFY
Contact: "Bob" <sip:6002@10.24.18.138:5060;ob>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Content-Length: 0

<----->

```

## Phone Support for Presence State via SIP presence notifications

At the time of writing, only Digium phones have built-in support for interpreting Asterisk's Presence State notifications (as opposed to SIP presence notifications for extension/device state). The CustomPresence provider itself is device-agnostic and support for other devices could be added in. Or devices themselves (soft-phone or hardphone) could be modified to interpret the XML send out in the Presence State notification.

### Digium Phones

This [Video](#) provides more insight on how presence can be set and viewed on Digium phones.

When using Digium phones with the [Digium Phone Module for Asterisk](#), you can set hints in Asterisk so that when one Digium phone's presence is updated, other Digium phones can be notified of the presence change. The DPMA automatically creates provisions such that when a Digium Phone updates its presence, CustomPresence:<line name> is updated, where <line name> is the value set for the line= option in a type=phone category. Using the example dialplan from the Overview section, Digium phones that are subscribed to 2000@default will automatically be updated about line 2000's presence whenever line 2000's presence changes.



Digium phones support only the available, away, dnd, xa, and chat states. The unavailable and not\_set states are not supported.

# Querying and Manipulating State

## Overview

This section will enumerate and briefly describe the ways in which you can query or manipulate the various Asterisk state resources. [Device State](#), [Extension State](#) and [Presence State](#). Where mentioned, the various functions and commands will be linked to further available documentation.

## Device State

The **DEVICE\_STATE** function will return the Device State for a specified device state identifier and allow you to set Custom device states.

On the command line, the **devstate** command will allow you to list or modify Custom device states specifically.

```
devstate change      -- Change a custom device state
devstate list       -- List currently known custom device states
```

### On this Page

- [Overview](#)
- [Device State](#)
- [Extension State](#)
- [Presence State](#)
- [Asterisk Manager Interface actions](#)

## Extension State

The **EXTENSION\_STATE** function will return the [Extension State](#) for any specified extension that has a defined hint.

The CLI command **core show hints** will show extension state for all defined hints, as well as display a truncated list of the mapped Device State or Presence State identifiers.

```
myserver*CLI> core show hints
-- Registered Asterisk Dial Plan Hints --
          6002@from-internal      : SIP/6002           State:Unavailable   Watchers  0
          7777@from-internal      : SIP/6003,CustomPre  State:Unavailable   Watchers  0
-----
- 2 hints registered
```

## Presence State

Added in Asterisk 11, the **PRESENCE\_STATE** function will return [Presence State](#) for any specified Presence State identifier, or set the Presence State for specifically for a CustomPresence identifier.

The **presencestate** CLI command will list or modify any currently defined Presence State resources provided by the CustomPresence provider.

```
myserver*CLI> core show help presencestate
presencestate change  -- Change a custom presence state
presencestate list    -- List currently know custom presence states
```

## Asterisk Manager Interface actions

Any of the previously mentioned functions could be called via [AMI](#) with the [Setvar](#) and [Getvar](#) actions.

Then there are two more specific actions called [ExtensionState](#) and [PresenceState](#). See the linked documentation for more info.

# The Stasis Message Bus

## Overview



### Asterisk 12 and Later

This content only applies to Asterisk 12 and later.

In Asterisk 12, a new core component was added to Asterisk: the Stasis Message Bus. As the name suggests, Stasis is an internal publish/subscribe message bus that lets the real-time core of Asterisk inform other modules or components – who subscribe for specific information topic – about events that occurred that they were interested in.

While the Stasis Message Bus is mostly of interest to those developing Asterisk, its existence is a useful piece of information in understanding how the Asterisk architecture works.

## On This Page

- Overview
- Key Concepts
  - Publisher
  - Topic
  - Message
  - Subscriber
  - Cache
- Benefits

## Key Concepts

The Stasis Message Bus has many concepts that work in concert together. Some of the most important are:

### Publisher

A Publisher is some core component that wants to inform other components in Asterisk about some event that took place. More rarely, this can be a dynamically loadable module; most publishers however are real-time components in the Asterisk core (such as the Channel Core or the Bridging Framework).

### Topic

A Topic is a high level, abstract concept that provides a way to group events together. For example, a topic may be all changes to a single channel, or all changes to all bridges in Asterisk.

### Message

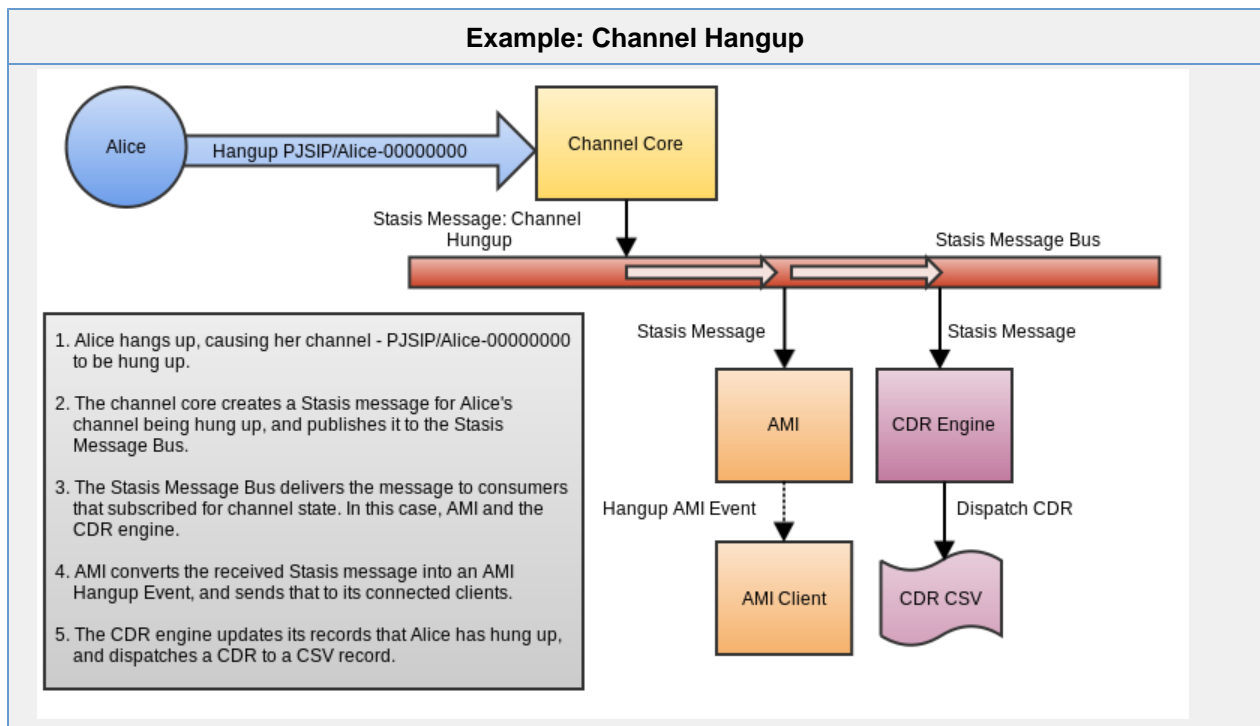
A Message contains the information about the event that just occurred. A Publisher publishes a Message under a specific Topic to the Stasis Message Bus.

### Subscriber

A Subscriber subscribes to a particular topic, and chooses which messages it is interested in. When the Stasis Message Bus receives a Message from a Publisher, it delivers the Message to each subscribed Subscriber.

### Cache

Some Messages - particularly those that affect core communications primitives in Asterisk (such as channels or bridges) are stored in a special cache in Stasis. Subscribers have the option to query the cache for the last known state of those primitives.



## Benefits

Prior to Asterisk 12, various parts of the real-time core of Asterisk itself would have been responsible for updating AMI, the CDR Engine, and other modules/components during key operations. By decoupling the consumers of state (such as AMI or the CDR Engine) from the producer (such as the Channel Core), we have the following benefits:

- **Improved Modularity:** the logic for AMI, CDRs, and other consumers of state is no longer tightly coupled with the real-time components. This simplifies both the producers and the consumers.
- **Insulation:** because the APIs are now based on the Stasis Message Bus, changes to other parts of the Asterisk core do not immediately affect the APIs. The APIs have the ability to transform, buffer, or even discard messages from the message bus, and can choose how to represent Asterisk to their consumers. This provides increased stability for Asterisk users.
- **Extensibility:** because real-time state is now readily available over the message bus, adding additional consumers of state becomes much easier. New interfaces and APIs can be added to Asterisk without modifying the Asterisk core.



# Configuration

This section contains many sub-sections on configuring every aspect of Asterisk. Other than what is covered under [Core Configuration](#), most features and functionality are [provided by modules](#) that you may or may not have installed in your Asterisk system. Built-in configuration documentation for each module (that has documentation) can be accessed through the [Asterisk CLI](#). The [CLI Syntax and Help Commands](#) section has more information on accessing the module configuration help.

## Topics

- [Core Configuration](#)
- [Channel Drivers](#)
- [Dialplan](#)
- [Features](#)
- [Applications](#)
- [Functions](#)
- [Reporting](#)
- [Interfaces](#)

# Core Configuration

The sub-pages here cover any possible configuration of Asterisk's core. That is, functionality which is not separated out into modules.

If you are unfamiliar with the core and modules concepts, take a look at the [Asterisk Architecture](#) section.

## Topics

- [Asterisk Main Configuration File](#)
- [Timing Interfaces](#)
- [Asterisk Builtin mini-HTTP Server](#)
- [Logging Configuration](#)
- [Asterisk CLI Configuration](#)
- [Configuring the Asterisk Module Loader](#)
- [Configuring Localized Tone Indications](#)
- [Video Telephony](#)
- [Video Console](#)
- [Named ACLs](#)

# Asterisk Main Configuration File

## The asterisk.conf file

asterisk.conf is used to configure the locations of directories and files used by Asterisk, as well as options relevant to the core of Asterisk.

[Link to the asterisk.conf.sample](#) file in the Asterisk trunk subversion repo. The information below could become out of date, so always check the relevant sample file in our version control system.

asterisk.conf has two primary contexts, shown below with some descriptions about their content.

## A Note on Includes

**Includes** in this file will only work with absolute paths, as the configuration in this file is setting the relative paths that would be used in includes set in other files.

## Directories Context

```
[directories](!)
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
astsbindir => /usr/sbin
```

The directories listed above are explained in detail in the [Directory and File Structure](#) page.

## Options Context

Some additional annotation for each configuration option is included inline.



TODO: Match this up with what is current in the sample, and update both.

```
[options]
;Under "options" you can enter configuration options
;that you also can set with command line options
; Verbosity level for logging (-v) verbose = 0
; Debug: "No" or value (1-4)
debug = 3

; Background execution disabled (-f)
nofork=yes | no

; Always background, even with -v or -d (-F)
alwaysfork=yes | no

; Console mode (-c)
console= yes | no

; Execute with high priority (-p)
highpriority = yes | no

; Initialize crypto at startup (-i)
initcrypto = yes | no

; Disable ANSI colors (-n)
nocolor = yes | no

; Dump core on failure (-g)
dumpcore = yes | no

; Run quietly (-q)
quiet = yes | no

; Force timestamping in CLI verbose output (-T)
timestamp = yes | no
```

```

; User to run asterisk as (-U) NOTE: will require changes to
; directory and device permissions
runuser = asterisk

; Group to run asterisk as (-G)
rungroup = asterisk

; Enable internal timing support (-I)
internal_timing = yes | no

; Language Options
documentation_language = en | es | ru

; These options have no command line equivalent

; Cache record() files in another directory until completion
cache_record_files = yes | no
record_cache_dir = <dir>

; Build transcode paths via SLINEAR
transcode_via_sln = yes | no

; send SLINEAR silence while channel is being recorded
transmit_silence_during_record = yes | no

; The maximum load average we accept calls for
maxload = 1.0

; The maximum number of concurrent calls you want to allow
maxcalls = 255

; Stop accepting calls when free memory falls below this amount specified in MB
minmemfree = 256

; Allow #exec entries in configuration files
execincludes = yes | no

; Don't over-inform the Asterisk sysadm, he's a guru
dontwarn = yes | no

; System name. Used to prefix CDR uniqueid and to fill \${SYSTEMNAME}
systemname = <a_string>

; Should language code be last component of sound file name or first?
; when off, sound files are searched as <path>/<lang>/<file>
; when on, sound files are search as <lang>/<path>/<file>
; (only affects relative paths for sound files)
languageprefix = yes | no

; Locking mode for voicemail
; - lockfile: default, for normal use
; - flock: for where the lockfile locking method doesn't work
; eh. on SMB/CIFS mounts
lockmode = lockfile | flock

; Entity ID. This is in the form of a MAC address. It should be universally
; unique. It must be unique between servers communicating with a protocol
; that uses this value. The only thing that uses this currently is DUNDi,
; but other things will use it in the future.
; entityid=00:11:22:33:44:55

[files]
; Changing the following lines may compromise your security
; Asterisk.ctl is the pipe that is used to connect the remote CLI
; (asterisk -r) to Asterisk. Changing these settings change the
; permissions and ownership of this file.
; The file is created when Asterisk starts, in the "astrundir" above.
;astctlpermissions = 0660

```

```
;astctlowner = root  
;astctlgroup = asterisk  
;astctl = asterisk.ctl
```

# Timing Interfaces

## Asterisk Timing Interfaces

In the past, if internal timing were desired for an Asterisk system, then the only source acceptable was from DAHDI. Beginning with Asterisk 1.6.1, a new timing API was introduced which allows for various timing modules to be used. Asterisk includes the following timing modules:

- `res_timing_pthread.so`
- `res_timing_dahdi.so`
- `res_timing_timerfd.so` – as of Asterisk 1.6.2
- `res_timing_kqueue.so` – as of Asterisk 11

`res_timing_pthread` uses the POSIX pthreads library in order to provide timing. Since the code uses a commonly-implemented set of functions, `res_timing_pthread` is portable to many types of systems. In fact, this is the only timing source currently usable on a non-Linux system. Due to the fact that a single userspace thread is used to provide timing for all users of the timer, `res_timing_pthread` is also the least efficient of the timing sources and has been known to lose its effectiveness in a heavily-loaded environment.

`res_timing_dahdi` uses timing mechanisms provided by DAHDI. This method of timing was previously the only means by which Asterisk could receive timing. It has the benefit of being efficient, and if a system is already going to use DAHDI hardware, then it makes good sense to use this timing source. If, however, there is no need for DAHDI other than as a timing source, this timing source may seem unattractive. For users who are upgrading from Asterisk 1.4 and are used to the `ztdummy` timing interface, `res_timing_dahdi` provides the interface to DAHDI via the `dahdi` kernel module.



### Historical Note

At the time of Asterisk 1.4's release, Zaptel (now DAHDI) was used to provide timing to Asterisk, either by utilizing telephony hardware installed in the computer or via `ztdummy` (a kernel module) when no hardware was available.

When DAHDI was first released, the `ztdummy` kernel module was renamed to `dahdi_dummy`. As of DAHDI Linux 2.3.0 the `dahdi_dummy` module has been removed and its functionality moved into the main `dahdi` kernel module. As long as the `dahdi` module is loaded, it will provide timing to Asterisk either through installed telephony hardware or utilizing the kernel timing facilities when separate hardware is not available.

`res_timing_timerfd` uses a timing mechanism provided directly by the Linux kernel. This timing interface is only available on Linux systems using a kernel version at least 2.6.25 and a glibc version at least 2.8. This interface has the benefit of being very efficient, but at the time this is being written, it is a relatively new feature on Linux, meaning that its availability is not widespread.

`res_timing_kqueue` uses the `Kqueue` event notification system introduced with FreeBSD 4.1. It can be used on operating systems that support `Kqueue`, such as OpenBSD and Mac OS X. Because `Kqueue` is not available on Linux, this module will not compile or be available there.

## What Asterisk does with the Timing Interfaces

By default, Asterisk will build and load all of the timing interfaces. These timing interfaces are "ordered" based on a hard-coded priority number defined in each of the modules. As of the time of this writing, the preferences for the modules is the following: `res_timing_timerfd.so`, `res_timing_kqueue.so` (where available), `res_timing_dahdi.so`, `res_timing_pthread.so`.

The only functionality that requires internal timing is IAX2 trunking. It may also be used when generating audio for playback, such as from a file. Even though internal timing is not a requirement for most Asterisk functionality, it may be advantageous to use it since the alternative is to use timing based on incoming frames of audio. If there are no incoming frames or if the incoming frames of audio are from an unreliable or jittery source, then the corresponding outgoing audio will also be unreliable, or even worse, nonexistent. Using internal timing prevents such unreliability.

## Customizations/Troubleshooting

Now that you know Asterisk's default preferences for timing modules, you may decide that you have a different preference. Maybe you're on a `timerfd`-capable system but you would prefer to get your timing from DAHDI since you already are using DAHDI to drive your hardware.

Alternatively, you may have been directed to this document due to an error you are currently experiencing with Asterisk. If you receive an error message regarding timing not working correctly, then you can use one of the following suggestions to disable a faulty timing module.

1. Don't build the timing modules you know you will not use. You can disable the compilation of any of the timing modules using `menuselect`. The modules are listed in the "Resource Modules" section. Note that if you have already built Asterisk and have received an error about a timing module not working properly, it is not sufficient to disable it from being built. You will need to remove the module from your modules directory (by default, `/usr/lib/asterisk/modules`) to make sure that it does not get loaded again.
2. Build, but do not load the timing modules you know you will not use. You can edit `modules.conf` using `noload` directives to disable the loading of specific timing modules by default. Based on the note in the section above, you may realize that your Asterisk setup does not require internal timing at all. If this is the case, you can safely `noload` all timing modules.



Some confusion has arisen regarding the fact that non-DAHDI timing interfaces are available now. One common misconception which has arisen is that since timing can be provided elsewhere, DAHDI is no longer required for using the MeetMe application. Unfortunately, this is not the case. In addition to providing timing, DAHDI also provides a conferencing engine which the MeetMe application requires.

Starting with Asterisk 1.6.2, however, there is a new application, `ConfBridge`, which is capable of conference bridging without the use of DAHDI's

built-in mixing engine.

# Asterisk Builtin mini-HTTP Server

## Overview

The core of Asterisk provides a basic HTTP/HTTPS server.

Certain Asterisk modules may make use of the HTTP service, such as the [Asterisk Manager Interface](#) over HTTP, the [Asterisk Restful Interface](#) or WebSocket transports for modules that support that, like `chan_sip` or `chan_pjsip`.

## Configuration

The configuration sample file is by default located at `/etc/asterisk/http.conf`

A very basic configuration of `http.conf` could be as follows:

```
[general]
enabled=yes
bindaddr=0.0.0.0
bindport=8088
```

That configuration would enable the HTTP server and have it bind to all available network interfaces on port 8088.

## Configuration Options

See the sample file in your version of Asterisk for detail on the various configuration options, as this information is not yet automatically pushed to the wiki.



# Logging Configuration

## Asterisk Log File Configuration

General purpose logging facilities in Asterisk can be configured in the *logger.conf* file. Within this file one is able to configure Asterisk to log messages to files and/or a syslog and even to the Asterisk console. Note, the sections and descriptions listed below are meant to be informational and act as a guide (a "how to") when configuring logging in Asterisk. Options with stated defaults don't have to be explicitly set as they will simply default to a designated value.

### General Section:

```
[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. Note that there is
; also a fractional second parameter which may be used in this field. Use
; %lq for tenths, %2q for hundredths, etc.
;
dateformat = %F %T.%3q ; ISO 8601 date format with milliseconds

; Write callids to log messages (defaults to yes)
use_callids = yes

; Append the hostname to the name of the log files (defaults to no)
appendhostname = no

; Log queue events to a file (defaults to yes)
queue_log = yes

; Always log queue events to a file, even when a realtime backend is
; present (defaults to no).
queue_log_to_file = no

; Set the queue_log filename (defaults to queue_log)
queue_log_name = queue_log

; When using realtime for the queue log, use GMT for the timestamp
; instead of localtime. (defaults to no)
queue_log_realtime_use_gmt = no

; Log rotation strategy (defaults to sequential):
; none: Do not perform any log rotation at all. You should make
;       very sure to set up some external log rotate mechanism
;       as the asterisk logs can get very large, very quickly.
; sequential: Rename archived logs in order, such that the newest
;             has the highest sequence number. When
;             exec_after_rotate is set, ${filename} will specify
;             the new archived logfile.
; rotate: Rotate all the old files, such that the oldest has the
;         highest sequence number (this is the expected behavior
;         for Unix administrators). When exec_after_rotate is
;         set, ${filename} will specify the original root filename.
; timestamp: Rename the logfiles using a timestamp instead of a
;            sequence number when "logger rotate" is executed.
;            When exec_after_rotate is set, ${filename} will
;            specify the new archived logfile.
rotatestrategy = rotate

; Run a system command after rotating the files. This is mainly
; useful for rotatestrategy=rotate. The example allows the last
; two archive files to remain uncompressed, but after that point,
; they are compressed on disk.
exec_after_rotate=gzip -9 ${filename}.2
```

### Log Files Section:

```

[logfiles]
; File names can either be relative to the standard Asterisk log directory (see "astlogdir" in
; asterisk.conf), or absolute paths that begin with '/'.
;
; A few file names have been reserved and are considered special, thus cannot be used and will
; not be considered as a regular file name. These include the following:
;
;   syslog - logs to syslog facility
;   console - logs messages to the Asterisk root console.
;
; For each file name given a comma separated list of logging "level" types should be specified
; and include at least one of the following (in no particular order):
;   debug
;   notice
;   warning
;   error
;   dtmf
;   fax
;   security
;   verbose(<level>)
;
; The "verbose" value can take an optional integer argument that indicates the maximum level
; of verbosity to log at. Verbose messages with higher levels than the indicated level will
; not be logged to the file. If a verbose level is not given, verbose messages are logged
; based upon the current level set for the root console.
;
; The special character "*" can also be specified and represents all levels, even dynamic
; levels registered by modules after the logger has been initialized. This means that loading
; and unloading modules that create and remove dynamic logging levels will result in these
; levels being included on filenames that have a level name of "*", without any need to
; perform a "logger reload" or similar operation.
;
; Note, there is no value in specifying both "*" and specific level types for a file name.
; The "*" level means ALL levels. The only exception is if you need to specify a specific
; verbose level. e.g, "verbose(3)*".
;
; It is highly recommended that you DO NOT turn on debug mode when running a production system
; unless you are in the process of debugging a specific issue. Debug mode outputs a LOT of
; extra messages and information that can and do fill up log files quickly. Most of these
; messages are hard to interpret 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.

; output notices, warnings and errors to the console
console => notice,warning,error

; output security messages to the file named "security"
security => security

; output notices, warnings and errors to the the file named "messages"
messages => notice,warning,error

; output notices, warnings, errors, verbose, dtmf, and fax to file name "full"
full => notice,warning,error,verbose,dtmf,fax

; output notices, warning, and errors to the syslog facility
syslog.local0 => notice,warning,error

```

# Asterisk CLI Configuration

With the exception of the functionality provided by the `res_clialias` module, [Asterisk's Command Line Interface](#) is provided by the core. There are a few configuration files relevant to the CLI that you'll see in a default Asterisk installation. All of these should be found in the typical `/etc/asterisk/` directory in a default install. The configuration of these files is trivial and examples exist in the sample files included in the source and tarballs.

## cli.conf

This file allows a listing of CLI commands to be automatically executed upon startup of Asterisk.

## cli\_permissions.conf

Allows you to configure specific restrictions or allowances on commands for users connecting to an Asterisk console. Read through the sample file carefully before making use of it, as you could create security issues.

## cli\_aliases.conf

This file allows configuration of aliases for existing commands. For example, the 'help' command is really an alias to 'core show help'. This functionality is provided by the `res_clialias` module.

## CLI related commands

There are a few commands relevant to the CLI configuration itself.

- **cli check permissions** - allows you to try running a command through the permissions of a specified user
- **cli reload permissions** - reloads the `cli_permissions.conf` file
- **cli show permissions** - shows configured CLI permissions
- **cli show aliases** - shows configured CLI command aliases

## Changing the CLI Prompt

The CLI prompt is set with the `ASTERISK_PROMPT` UNIX environment variable that you set from the Unix shell before starting Asterisk

You may include the following variables, that will be replaced by the current value by Asterisk:

- `%d` - Date (year-month-date)
- `%s` - Asterisk system name (from `asterisk.conf`)
- `%h` - Full hostname
- `%H` - Short hostname
  
- `%t` - Time
- `%u` - Username
- `%g` - Groupname
- `%%` - Percent sign
  
- `%#` - '#' if Asterisk is run in console mode, " if running as remote console
- `%Cn[:n]` - Change terminal foreground (and optional background) color to specified A full list of colors may be found in `include/asterisk/term.h`

On systems which implement `getloadavg(3)`, you may also use:

- `%l1` - Load average over past minute
- `%l2` - Load average over past 5 minutes
- `%l3` - Load average over past 15 minutes

# Configuring the Asterisk Module Loader

## Overview

As you may have learned from the [Asterisk Architecture](#) section, the majority of Asterisk's features and functionality are separated outside of the core into various **modules**. Each module has distinct functionality, but sometimes relies on another module or modules.

Asterisk provides capability to automatically and manually load modules. Module load order can be configured before load-time, or modules may be loaded and unloaded during run-time.

## Configuration

The configuration file for Asterisk's module loader is **modules.conf**. It is read from the typical [Asterisk configuration directory](#). You can also view the sample of modules.conf file in your source directory at [configs/modules.conf.sample](#) or on [SVN](#) at [this link](#).

The configuration consist of one large section called "modules" with possible directives configured within.

There are several directives that can be used.

- **autoload** - When enabled, Asterisk will automatically load any modules found in the [Asterisk modules directory](#).
- **preload** - Used to specify individual modules to load before the Asterisk core has been initialized. Often used for [realtime](#) modules so that config files can be pushed to a backend before the dependent modules are loaded.
- **require** - Set a required module. If a required module does not load, then Asterisk exits with status code 2.
- **preload-require** - A combination of preload and require.
- **noload** - Do not load the specified module.
- **load** - Load the specified module. Typically used when autoload is set to 'no'.

Let's show a few arbitrary examples below.

```
[modules]
;autoload = yes
;preload = res_odbc.so
;preload = res_config_odbc.so
;preload-require = res_odbc.so
;require = res_pjsip.so
;noload = pbx_gtkconsole.so
;load = res_musiconhold.so
```

## CLI Commands

Asterisk provides a few commands for managing modules at run-time. Be sure to check the current usage using the CLI help with "core show help <command>".

- **module show**

```
Usage: module show [like keyword]
Shows Asterisk modules currently in use, and usage statistics.
```

- **module load**

```
Usage: module load <module name>
Loads the specified module into Asterisk.
```

- **module unload**

```
Usage: module unload [-f|-h] <module_1> [<module_2> ... ]
Unloads the specified module from Asterisk. The -f
option causes the module to be unloaded even if it is
in use (may cause a crash) and the -h module causes the
module to be unloaded even if the module says it cannot,
which almost always will cause a crash.
```

- **module reload**

```
Usage: module reload [module ...]
Reloads configuration files for all listed modules which support
reloading, or for all supported modules if none are listed.
```

# Configuring Localized Tone Indications

## Overview

In certain cases Asterisk will generate tones to be used in call signaling. It may be during the use of a specific application, or with certain channel drivers. The tones used are configurable and may be defined by location.

Note that the tones configured here are only used when Asterisk is directly generating the tones.

## Configuration

The configuration file for location specific tone indications is **indications.conf**. It is read from the typical Asterisk configuration directory. You can also view the sample of indications.conf file in your source directory at [configs/modules.conf.sample](#) or on [SVN at this link](#).

The configuration itself consists of a 'general' section and then one or more country specific sections. (e.g. '[au]' for Australia)

Within the general section, only the **country** option can be set. This option sets the default location tone set to be used.

```
[general]
country=us
```

As an example, the above set the default country to the tone set for the USA.

Within any location specific configuration, several tone types may be configured.

- **description = string** ; The full name of your country, in English.
- **ringcadence = num[,num]\*** ; List of durations the physical bell rings.
- **dial = tonelist** ; Set of tones to be played when one picks up the hook.
- **busy = tonelist** ; Set of tones played when the receiving end is busy.
- **congestion = tonelist** ; Set of tones played when there is some congestion (on the network?)
- **callwaiting = tonelist** ; Set of tones played when there is a call waiting in the background.
- **dialrecall = tonelist** ; Not well defined; many phone systems play a recall dial tone after hook flash
- **record = tonelist** ; Set of tones played when call recording is in progress.
- **info = tonelist** ; Set of tones played with special information messages (e.g., "number is out of service")
- **'name' = tonelist** ; Every other variable will be available as a shortcut for the "PlayList" command but will not be used automatically by Asterisk.

## Explanation of the 'tonelist' usage:

```
; The tonelist itself is defined by a comma-separated sequence of elements.
; Each element consist of a frequency (f) with an optional duration (in ms)
; attached to it (f/duration). The frequency component may be a mixture of two
; frequencies (f1+f2) or a frequency modulated by another frequency (f1*f2).
; The implicit modulation depth is fixed at 90%, though.
; If the list element starts with a !, that element is NOT repeated,
; therefore, only if all elements start with !, the tonelist is time-limited,
; all others will repeat indefinitely.
;
; concisely:
; element = [!]freq[+]*freq2[/duration]
; tonelist = element[,element]*
```

## Example of a location specific tone configuration:

```
[br]
description = Brazil
ringcadence = 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
; Dialrecall not used in Brazil standard (using UK standard)
dialrecall = 350+440
; Record tone is not used in Brazil, use busy tone
record = 425/250,0/250
; Info not used in Brazil standard (using UK standard)
info = 950/330,1400/330,1800/330
stutter = 350+440
```

# Video Telephony

## Asterisk and Video telephony

Asterisk supports video telephony in the core infrastructure. Internally, it's one audio stream and one video stream in the same call. Some channel drivers and applications has video support, but not all.

### Codecs and formats

Asterisk supports the following video codecs and file formats. There's no video transcoding so you have to make sure that both ends support the same video format.

Codec	Format	Notes
H.263	read/write	
H.264	read/write	
H.261	-	Passthrough only

Note that the file produced by Asterisk video format drivers is in no generic video format. Gstreamer has support for producing these files and converting from various video files to Asterisk video+audio files.

Note that H.264 is not enabled by default. You need to add that in the channel configuration file.

### Channel Driver Support

Channel Driver	Module	Notes
SIP	chan_sip.so	The SIP channel driver (chan_sip.so) has support for video
IAX2	chan_iax2.so	Supports video calls (over trunks too)
Local	chan_local.so	Forwards video calls as a proxy channel
Agent	chan_agent.so	Forwards video calls as a proxy channel
oss	chan_oss.so	Has support for video display/decoding, see video_console.txt

### Applications

This is not yet a complete list. These dialplan applications are known to handle video:

- Voicemail - Video voicemail storage (does not attach video to e-mail)
- Record - Records audio and video files (give audio format as argument)
- Playback - Plays a video while being instructed to play audio
- Echo - Echos audio and video back to the user

There is a development group working on enhancing video support for Asterisk.

If you want to participate, join the asterisk-video mailing list on <http://lists.digium.com>

Updates to this file are more than welcome!

# Video Console

## Video Console Support in Asterisk

Some console drivers (at the moment chan\_oss.so) can be built with support for sending and receiving video. In order to have this working you need to perform the following steps:

### Enable building the video\_console support

The simplest way to do it is add this one line to channels/Makefile:

```
chan_oss.so: _ASTCFLAGS+=-DHAVE_VIDEO_CONSOLE
```

### Install prerequisite packages

The video\_console support relies on the presence of SDL, SDL\_image and ffmpeg libraries, and of course on the availability of X11

On Linux, these are supplied by

- libncurses-dev
- libsdl1.2-dev
- libsdl-image1.2-dev
- libavcodec-dev
- libswscale-dev

On FreeBSD, you need the following ports:

- multimedia/ffmpeg (2007.10.04)
- devel/sdl12 graphics/sdl\_image

### Build and install asterisk with all the above

Make sure you do a 'make clean' and run configure again after you have installed the required packages, to make sure that the required pieces are found.

Check that chan\_oss.so is generated and correctly installed.

### Update configuration files

Video support requires explicit configuration as described below:

#### oss.conf

You need to set various parameters for video console, the easiest way is to uncomment the following line in oss.conf by removing the leading ';'.

```
:[general](+,my_video,skin2)
```

You also need to manually copy the two files

- images/kpad2.jpg
- images/font.png

into the places specified in oss.conf, which in the sample are set to

```
keypad = /tmp/kpad2.jpg  
keypad_font = /tmp/font.png
```

other configuration parameters are described in oss.conf.sample

#### sip.conf

To actually run a call using SIP (the same probably applies to iax.conf) you need to enable video support as following

```
[general](+)
videosupport=yes
allow=h263      ; this or other video formats
allow=h263p    ; this or other video formats
```

You can add other video formats e.g. h261, h264, mpeg if they are supported by your version of libavcodec.

### Run the Program

Run asterisk in console mode e.g. asterisk -vdc

If video console support has been successfully compiled in, then you will see the "console startgui" command available on the CLI interface. Run the command, and you should see a window like this [http://info.iet.unipi.it/~luigi/asterisk\\_video\\_console.jpg](http://info.iet.unipi.it/~luigi/asterisk_video_console.jpg)

To exit from this window, in the console run "console stopgui".

If you want to start a video call, you need to configure your dialplan so that you can reach (or be reachable) by a peer who can support video. Once done, a video call is the same as an ordinary call:

"console dial ...", "console answer", "console hangup" all work the same.

To use the GUI, and also configure video sources, see the next section.

### Video Sources

Video sources are declared with the "videodevice=..." lines in oss.conf where the ... is the name of a device (e.g. /dev/video0 ...) or a string starting with X11 which identifies one instance of an X11 grabber.

You can have up to 9 sources, displayed in thumbnails in the gui, and select which one to transmit, possibly using Picture-in-Picture.

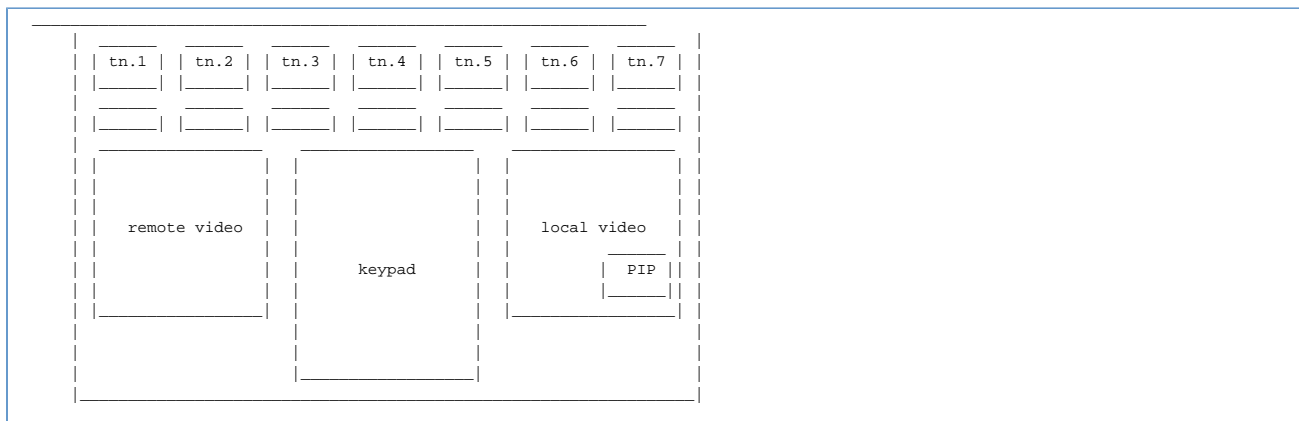
For webcams, the only control you have is the image size and frame rate (which at the moment is the same for all video sources). X11 grabbers capture a region of the X11 screen (it can contain anything, even a live video) and use it as the source. The position of the grab region can be configured using the GUI below independently for each video source.

The actual video sent to the remote side is the device selected as "primary" (with the mouse, see below), possibly with a small 'Picture-in-Picture' of the "secondary" device (all selectable with the mouse).

### GUI Commands and Video Sources

(most of the text below is taken from channels/console\_gui.c)

The GUI is made of 4 areas: remote video on the left, local video on the right, keypad with all controls and text windows in the center, and source device thumbnails on the top. The top row is not displayed if no devices are specified in the config file.



The central section is built using an image (jpg, png, maybe gif too) for the skin and other GUI elements. Comments embedded in the image indicate to what function each area is mapped to.

Another image (png with transparency) is used for the font.

Mouse and keyboard events are detected on the whole surface, and handled differently according to their location:

- Center/right click on the local/remote window are used to resize the corresponding window
- Clicks on the thumbnail start/stop sources and select them as primary or secondary video sources
- Drag on the local video window are used to move the captured area (in the case of X11 grabber) or the picture-in-picture position
- Keystrokes on the keypad are mapped to the corresponding key; keystrokes are used as keypad functions, or as text input



if we are in text-input mode.

- Drag on some keypad areas (sliders etc.) are mapped to the corresponding functions (mute/unmute audio and video, enable/disable Picture-in-Picture, freeze the incoming video, dial numbers, pick up or hang up a call, ...)

# Named ACLs

## Overview

Named ACLs introduce a new way to define Access Control Lists (ACLs) in Asterisk. Unlike traditional ACLs defined in specific module configuration files, Named ACLs can be shared across multiple modules. Named ACLs can also be accessed via the Asterisk Realtime Architecture (ARA), allowing for run-time updates of ACL information that can be retrieved by multiple consumers of ACL information.

## Configuration

### Static Configuration

Named ACLs can be defined statically in *acl.conf*. Each context in *acl.conf* defines a specific Named ACL, where the name of the context is the name of the ACL. The syntax for each context follows the permit/deny nomenclature used in traditional ACLs defined in a consumer module's configuration file.

Option	Value	Description
deny	IP address [/Mask]	An IP address to deny, with an optional subnet mask to apply
permit	IP address [/Mask]	An IP address to allow, with an optional subnet mask to apply

### Examples

```
; within acl.conf  
  
[name_of_acl1]  
deny=0.0.0.0/0.0.0.0  
permit=127.0.0.1
```

Multiple rules can be specified in an ACL as well by chaining deny/permit specifiers.

```
[name_of_acl2]  
deny=10.24.0.0/255.255.0.0  
deny=10.25.0.0/255.255.0.0  
permit=10.24.11.0/255.255.255.0  
permit=10.24.12.0/255.255.255.0
```

Named ACLs support common modifiers like templates and additions within configuration as well.

```
[template_deny_all](!  
deny=0.0.0.0/0.0.0.0  
  
[deny_all_whitelist_these](template_deny_all)  
permit=10.24.20.1  
permit=10.24.20.2  
permit=10.24.20.3
```

### Configuring for IPv6

Named ACLs can use ipv6 addresses just like normal ACLs.

```
[ipv6_example_1]  
deny = ::  
permit = ::1/128  
  
[ipv6_example_2]  
permit = fe80::21d:bad:fad:2323
```

## On This Page

- Overview
  - Configuration
    - Static Configuration
    - Configuring for IPv6
    - ARA Configuration
  - Named ACL Consumers
    - Configuration
    - ACL Rule Application
  - Module Reloads

## ARA Configuration

The ARA supports Named ACLs using the 'acls' keyword in *extconfig.conf*.

### Example Configuration

```
;in extconfig.conf
acls => odbc,asterisk,acltable
```

## Schema

Column Name	Type	Description
name	varchar(80)	Name of the ACL
rule_order	integer	Order to apply the ACL rule. Rules are applied in ascending order. Rule numbers do not have to be sequential
sense	varchar(6)	Either 'permit' or 'deny'
rule	varchar(95)	The IP address/Mask pair to apply

## Examples

### Table Creation Script (PostgreSQL)

```
CREATE TABLE acltable
(
    "name" character varying(80) NOT NULL,
    rule_order integer NOT NULL,
    sense character varying(6) NOT NULL,
    "rule" character varying(95) NOT NULL,
    CONSTRAINT aclrulekey PRIMARY KEY (name, rule_order, rule, sense)
)
WITH (
    OIDS=FALSE
);
ALTER TABLE acltable OWNER TO asterisk;
GRANT ALL ON TABLE acltable TO asterisk;
)
```

### Table Creation Script (SQLite3)

```
BEGIN TRANSACTION;
CREATE TABLE acltable (rule TEXT, sense TEXT, rule_order NUMERIC, name TEXT);
COMMIT;
```



These scripts were generated by pgadmin III and SQLite Database Browser. They might not necessarily apply for your own setup.



Since ACLs are obtained by consumer modules when they are loaded, an ACL updated in an ARA backend will not be propagated automatically to consumers using static configuration. Consumer modules also using ARA for their configuration (such as SIP/IAX2 peers) will similarly be up to date if and only if they have built the peer in question since the changes to the realtime ACL have taken place.

## Named ACL Consumers

Named ACLs are supported by the following Asterisk components:

- Manager (IPv4 and IPv6)
- chan\_sip (IPv4 and IPv6)
- chan\_pjsip (IPv4 and IPv6)
- chan\_iax2 (IPv4 and IPv6)

## Configuration

A consumer of Named ACLs can be configured to use a named ACL using the *acl* option in their ACL access rules. This can be in addition to the ACL rules traditionally defined in those configuration files.

### Example 1: referencing a Named ACL

```
; within sip.conf

[peer1]
;stuff
;deny=0.0.0.0/0.0.0.0
;permit=127.0.0.1
acl=name_of_acl_1 ; an ACL included from acl.conf that matches peer1's commented out permits/denies
```

Multiple named ACLs can be referenced as well by specifying a comma delineated list of Named ACLs to apply.

### Example 2: multiple Named ACL references

```
; within sip.conf

[peer1]
;stuff
acl=named_acl_1,named_acl_2
```

Similarly, a SIP or IAX2 peer defined in ARA can include an `acl` column and list the Named ACLs to apply in that column.



#### NOTE

Named ACLs can also be defined using multiple instances of the `acl` keyword. This is discouraged, however, as the order in which ACLs are applied can be less obvious than the comma delineated list format.

```
acl=named_acl_1
acl=named_acl_2
```

## ACL Rule Application

Each module consumer of ACL information maintains, for each object that uses the information, a list of the defined ACL rule sets that apply to that object. When an address is evaluated for the particular object, the address is evaluated against each rule. For an address to pass the ACL rules, it must pass each ACL rule set that was defined for that object. Failure of any ACL rule set will result in a rejection of the address.

## Module Reloads

ACL information is static once a consumer module references that information. Hence, changes in ACL information in an ARA backend will not automatically update consumers of that information. In order for consumers to receive updated ACL information, the Named ACL component must be reloaded.

The Named ACL component supports module reloads, in the same way as other Asterisk components. When the Named ACL component is reloaded, it will issue a request to all consumers of Named ACLs. Those consumer modules will also be automatically reloaded.



#### WARNING

This implies that reloading the Named ACL component will force a reload of `manager`, `chan_sip`, etc. Only reload the Named ACL component if you want all consumers of that information to be reloaded as well.

# Channel Drivers

All about Asterisk and its Channel Drivers

## Topics

- SIP
- Inter-Asterisk eXchange protocol, version 2 (IAX2)
- DAHDI
- Local Channel
- Motif
- mISDN
- Mobile Channel
- Unistim
- Skinny
- RTP Packetization

# SIP



Under Construction



Section to hold information on configuring the SIP channel drivers, `chan_sip` and `chan_pjsip`

## Configuring chan\_sip



Under Construction - This is a stub

Currently the documentation resides in the sip.conf.sample file included with the source. We are in the process of updating the wiki!

## chan\_sip State and Presence Options

### Device State

There are a few configuration options for chan\_sip that affect [Device State](#) behavior.

#### **callcounter**

The **callcounter** option in sip.conf **must be enabled** for SIP devices (e.g. SIP/Alice) to provide advanced [device state](#). Without it you may see some state, such as unavailable or idle, but not much more.

The option can be set in the general context, or on a per-peer basis.

Default: no

```
[general]
callcounter=yes
```

#### **busylevel**

The **busylevel** option only works if call counters are enabled via the above option. If call counters are enabled, then busylevel allows you to set a threshold for when to consider this device busy. If busylevel is set to 2, then only at 2 or more calls will the device state report BUSY. The busylevel option can only be set for peers.

Default: 0

```
[6001]
type=friend
busylevel=2
```

#### **notifyhold**

The **notifyhold** option, when enabled, adds the ONHOLD device state to the range of possible device states that chan\_sip will use.

This option can only be set in the general section.

Default: yes

```
[general]
notifyhold=no
```

## Extension State, Hints, Subscriptions

Extension State and subscriptions tend to go hand in hand. That is, if you are using Extension State, you probably have SIP user agents subscribing to those extensions/hints. These options all affect that behavior.

#### **allowsubscribe**

The **allowsubscribe** option enables or disables support for any kind of subscriptions. You can set allowsubscribe per-peer or in the general section.

Default: yes

```
[6001]
type=friend
allowsubscribe=no
```

#### **subscribecontext**

**subscribecontext** sets a specific context to be used for subscriptions. That means, if SIP user agent subscribes to this peer, Asterisk will search for an associated hint mapping in the context specified.

This option can be set per-peer or in the general section.

Default: null (by default Asterisk will use the context specified with the "context" option)

```
[6001]
type=friend
context=internal
subscribecontext=myhints
```



## **notifyingringing**

**notifyingringing** enables or disables notifications for the RINGING state when an extension is already INUSE. Only affects subscriptions using the **dialog-info** event package. Option can be configured in the general section only. It cannot be set per-peer.

Default: yes

```
[general]
notifyingringing=no
```

## **notificycid**

**notificycid** some nuance and may only be relevant to SNOM phones or others that support dialog-info+xml notifications. Below are the notes from the sample sip.conf.

Default: no

```
;notificycid = yes                ; Control whether caller ID information is sent along with
                                   ; dialog-info+xml notifications (supported by snom phones).
                                   ; Note that this feature will only work properly when the
                                   ; incoming call is using the same extension and context that
                                   ; is being used as the hint for the called extension. This means
                                   ; that it won't work when using subscribecontext for your sip
                                   ; user or peer (if subscribecontext is different than context).
                                   ; This is also limited to a single caller, meaning that if an
                                   ; extension is ringing because multiple calls are incoming,
                                   ; only one will be used as the source of caller ID. Specify
                                   ; 'ignore-context' to ignore the called context when looking
                                   ; for the caller's channel. The default value is 'no.' Setting
                                   ; notificycid to 'ignore-context' also causes call-pickups attempted
                                   ; via SNOM's NOTIFY mechanism to set the context for the call pickup
                                   ; to PICKUPMARK.
```

## Configuring chan\_sip for IPv6

### Configuring chan\_sip for IPv6

Mostly you can use IPv6 addresses where you would have otherwise used IPv4 addresses within sip.conf. The sip.conf.sample provides several examples of how to use the various options with IPv6 addresses. We'll provide a few examples here as well.

#### Examples

Binding to a specific IPv6 interface

```
[general]
bindaddr=2001:db8::1
```

Binding to all available IPv6 interfaces (wildcard)

```
[general]
bindaddr>:::
```

You can specify a port number by wrapping the address in square brackets and using a colon delimiter.

```
[general]
bindaddr=[::]:5062
```



You can choose independently for UDP, TCP, and TLS, by specifying different values for "udpbindaddr", "tcpbindaddr", and "tlsbindaddr".

Note that using bindaddr>::: will show only a single IPv6 socket in netstat. IPv4 is supported at the same time using IPv4-mapped IPv6 addresses.)

#### Other Options

Other options such as "outboundproxy" or "permit" can use IPv6 addresses the same as in the above examples.

```
permit=2001:db8::/32
```

# Configuring chan\_sip for Presence Subscriptions

## Overview

This page is a rough guide to get you configuring chan\_sip and Asterisk to accept subscriptions for presence (in this case, [Extension State](#)) and notify the subscribers of state changes.

## Requirements

You should understand the basics of

- [Device State](#) and [Extension State and Hints](#)
- [Configuring SIP peers in sip.conf](#)

## General Process

### Overview

It is best to consider this configuration in the context of a very simplified use case. It should illustrate the overall concept, as well as the ability for Extension State to aggregate Device States.

The case is that our administrator wants the user device of SIP/Alice to display the presence of Bob. Bob has two devices, SIP/Bob-mobile and SIP/Bob-desk. He could be on either device at any one time, so we want to map them both to the same Hint. That way, when Alice subscribes to the Hint, she'll get the aggregated Extension State of Bob's devices. That means if either of Bobs phones are busy, then the extension state will be busy. Then Alice knows that Bob is busy without having to have a separate light for each of Bob's phones.

Figure 1 should illustrate the overall relationships of the different elements involved.

Then following down the page you can find detail on configuring the three major elements, SIP configuration options, hints in dialplan, and configuring a phone to subscribe.

### Configure SIP options

Since this is not a guide on configuring SIP peers, we'll show a very simple **sip.conf** with only enough configuration to point out where you might set specific **chan\_sip State and Presence Options** .

```
[general]
callcounter=yes

[Alice]
type=friend
subscribecontext=default
allowssubscribe=yes

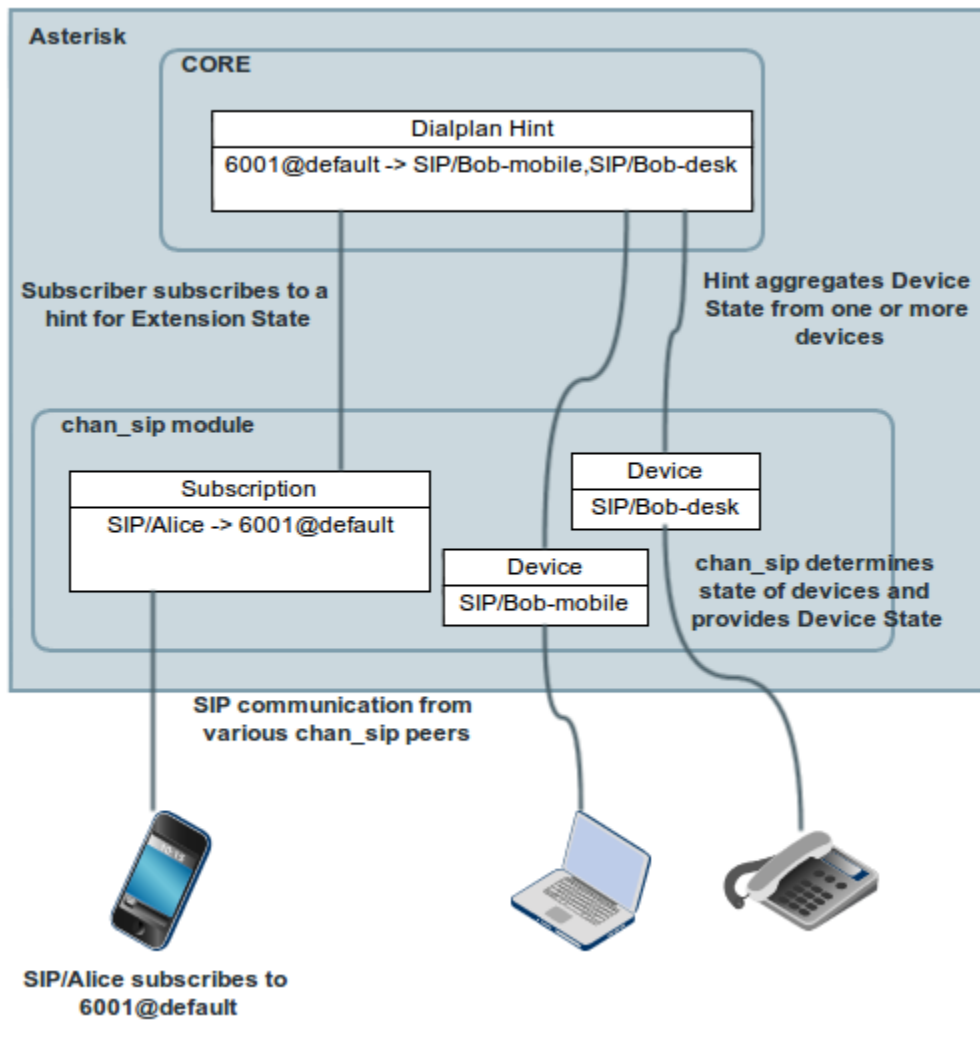
[Bob-mobile]
type=friend
busylevel=1

[Bob-desk]
type=friend
busylevel=1
```

We are setting one option in the general section, and then a few options across the three SIP peers involved.

**callcounter** and **busylevel** are the most essential options. **callcounter** needs to be enabled for chan\_sip to provide accurate device. **busylevel=1** says we want the device states of those peers to show busy if they have at least one call in progress. The **subscribecontext** option tells Asterisk which dialplan context to look for the hint. **allowssubscribe** says that we will allow subscriptions for that peer. It is really set to yes by default, but we are defining it here to demonstrate that you could allow and disallow subscriptions on a per-peer basis if you wanted.

Figure 1



This diagram is purposefully simplified to only show the relationships between the elements involved in this configuration.

### Configure Hints

Hints are configured in Asterisk `dialplan` (`extensions.conf`). This is where you map `Device State` identifiers or `Presence State` identifiers to a hint, which will then be subscribed to by one or more SIP User Agents.

For our example we need to define a hint mapping 6001 to Bob's two devices.

```
[default]
exten = 6001,hint,SIP/Bob-mobile&SIP/Bob-desk
```

Defining the hint is pretty straightforward and follows the syntax discussed in the `Extension State and Hints` section.

Notice that we put it in the context we set in `subscribecontext` in `sip.conf` earlier. Otherwise we would need to make sure it is in the same context that the SIP peer uses (defined with "context").

If you have restarted Asterisk to load the hints, then you can check to make sure they are configured with "core show hints"

```
*CLI> core show hints
-- Registered Asterisk Dial Plan Hints ==
        6001@default          : SIP/Bob-mobile&SIP/B State:Unavailable   Watchers 0
```

You'll see the state changes to `Idle` or something else if you have your `sip.conf` configured properly and the two SIP devices are at least available.

## Configure Subscriber

You should configure your SIP User Agent (soft-phone, hard-phone, another phone application like Asterisk) to subscribe to the hint. In this case that is SIP/Alice and we want her phone to subscribe to 6001.

The process will be different for every phone, and keep in mind that some phones may not support Asterisk's state notification. With most phones it'll be a matter of adding a "contact" to a contact list, buddy list, or address book and then making sure that SIP presence is enabled in the options.

If you want to submit a guide for a specific phone, feel free to comment on this page or submit it to the [Asterisk issue tracker](#).

## Operation

Typically as soon as you add the contact or subscription on the phone then it will attempt to SUBSCRIBE to Asterisk.

If you haven't done so, restart Asterisk and then restart the SIP User Agent client doing the subscribing.

The flow of SIP messaging can differ based on configuration, but typically looks like this for a peer that requires authentication:

SIP/Alice	Asterisk
SUBSCRIBE	--->
	<--- 401 Unauthorized
SUBSCRIBE(w/ Auth)	--->
	<--- 200 OK
	<--- NOTIFY
200 OK	--->

In the expanding frame below is a SIP trace of a successful subscription for reference. You could see this on your own system by running "sip set debug on" and then watching for the subscription. You might have to restart your phone again or re-add a contact to see it.

▼ [Click to see the subscription trace...](#)

```
<--- SIP read from UDP:10.24.17.254:37509 --->
SUBSCRIBE sip:6001@10.24.18.124;transport=UDP SIP/2.0
Via: SIP/2.0/UDP 10.24.17.254:37509;branch=z9hG4bK-d8754z-e5ecfdelf337b690-1---d8754z-
Max-Forwards: 70
Contact: <sip:Alice@10.24.17.254:37509;transport=UDP>
To: <sip:6001@10.24.18.124;transport=UDP>
From: <sip:Alice@10.24.18.124;transport=UDP>;tag=f51e9632
Call-ID: ZjE2ZDAwYThiOTA2MzYxOWEwNTEwMjc1ZGIxNTk3NDU.
CSeq: 1 SUBSCRIBE
Expires: 1800
Accept: application/pidf+xml
Allow: INVITE, ACK, CANCEL, BYE, NOTIFY, REFER, MESSAGE, OPTIONS, INFO, SUBSCRIBE
Supported: replaces, norefersub, extended-refer, timer, X-cisco-serviceuri
User-Agent: Z 3.2.21357 r21103
Event: presence
Allow-Events: presence, kpml
Content-Length: 0

<----->
--- (16 headers 0 lines) ---
Sending to 10.24.17.254:37509 (no NAT)
Creating new subscription
Sending to 10.24.17.254:37509 (no NAT)
list_route: route/path hop: <sip:Alice@10.24.17.254:37509;transport=UDP>
Found peer 'Alice' for 'Alice' from 10.24.17.254:37509

<--- Transmitting (no NAT) to 10.24.17.254:37509 --->
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 10.24.17.254:37509;branch=z9hG4bK-d8754z-e5ecfdelf337b690-1---d8754z-;received=10.24.17.254
From: <sip:Alice@10.24.18.124;transport=UDP>;tag=f51e9632
To: <sip:6001@10.24.18.124;transport=UDP>;tag=as46a6e039
Call-ID: ZjE2ZDAwYThiOTA2MzYxOWEwNTEwMjc1ZGIxNTk3NDU.
CSeq: 1 SUBSCRIBE
Server: Asterisk PBX SVN-branch-12-r413487
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO, PUBLISH, MESSAGE
Supported: replaces, timer
WWW-Authenticate: Digest algorithm=MD5, realm="asterisk", nonce="522456f4"
Content-Length: 0

<----->
Scheduling destruction of SIP dialog 'ZjE2ZDAwYThiOTA2MzYxOWEwNTEwMjc1ZGIxNTk3NDU.' in 32000 ms (Method: SUBSCRIBE)

<--- SIP read from UDP:10.24.17.254:37509 --->
SUBSCRIBE sip:6001@10.24.18.124;transport=UDP SIP/2.0
Via: SIP/2.0/UDP 10.24.17.254:37509;branch=z9hG4bK-d8754z-c6908de6f0126edf-1---d8754z-
Max-Forwards: 70
Contact: <sip:Alice@10.24.17.254:37509;transport=UDP>
To: <sip:6001@10.24.18.124;transport=UDP>
From: <sip:Alice@10.24.18.124;transport=UDP>;tag=f51e9632
Call-ID: ZjE2ZDAwYThiOTA2MzYxOWEwNTEwMjc1ZGIxNTk3NDU.
CSeq: 2 SUBSCRIBE
Expires: 1800
```

```

Accept: application/pidf+xml
Allow: INVITE, ACK, CANCEL, BYE, NOTIFY, REFER, MESSAGE, OPTIONS, INFO, SUBSCRIBE
Supported: replaces, noferesub, extended-refer, timer, X-cisco-serviceuri
User-Agent: Z 3.2.21357 r21103
Authorization: Digest
username="Alice",realm="asterisk",nonce="522456f4",uri="sip:6001@10.24.18.124;transport=UDP",response="6d66dcad8c176aa3ef7baec7680d2445",algorithm=MD5
Event: presence
Allow-Events: presence, kpml
Content-Length: 0

<----->
--- (17 headers 0 lines) ---
Creating new subscription
Sending to 10.24.17.254:37509 (no NAT)
Found peer 'Alice' for 'Alice' from 10.24.17.254:37509
Looking for 6001 in default (domain 10.24.18.124)
Scheduling destruction of SIP dialog 'ZjE2ZDAwYThiOTA2MzYxOWEwNTEwMjc1ZGIxNTk3NDU.' in 1810000 ms (Method: SUBSCRIBE)

<--- Transmitting (no NAT) to 10.24.17.254:37509 --->
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.24.17.254:37509;branch=z9hG4bK-d8754z-c6908de6f0126edf-1---d8754z-;received=10.24.17.254
From: <sip:Alice@10.24.18.124;transport=UDP>;tag=f51e9632
To: <sip:6001@10.24.18.124;transport=UDP>;tag=as46a6e039
Call-ID: ZjE2ZDAwYThiOTA2MzYxOWEwNTEwMjc1ZGIxNTk3NDU.
CSeq: 2 SUBSCRIBE
Server: Asterisk PBX SVN-branch-12-r413487
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO, PUBLISH, MESSAGE
Supported: replaces, timer
Expires: 1800
Contact: <sip:6001@10.24.18.124:5060>;expires=1800
Content-Length: 0

<----->
set_destination: Parsing <sip:Alice@10.24.17.254:37509;transport=UDP> for address/port to send to
set_destination: set destination to 10.24.17.254:37509
Reliably Transmitting (no NAT) to 10.24.17.254:37509:
NOTIFY sip:Alice@10.24.17.254:37509;transport=UDP SIP/2.0
Via: SIP/2.0/UDP 10.24.18.124:5060;branch=z9hG4bK14aacddc
Max-Forwards: 70
From: <sip:6001@10.24.18.124;transport=UDP>;tag=as46a6e039
To: <sip:Alice@10.24.18.124;transport=UDP>;tag=f51e9632
Contact: <sip:6001@10.24.18.124:5060>
Call-ID: ZjE2ZDAwYThiOTA2MzYxOWEwNTEwMjc1ZGIxNTk3NDU.
CSeq: 102 NOTIFY
User-Agent: Asterisk PBX SVN-branch-12-r413487
Subscription-State: active
Event: presence
Content-Type: application/pidf+xml
Content-Length: 530

<?xml version="1.0" encoding="ISO-8859-1"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
xmlns:pp="urn:ietf:params:xml:ns:pidf:person"
xmlns:es="urn:ietf:params:xml:ns:pidf:rpidd:status:rpidd-status"
xmlns:ep="urn:ietf:params:xml:ns:pidf:rpidd:person"
entity="sip:Alice@10.24.18.124">
<pp:person><status>
<ep:activities><ep:away/></ep:activities>
</status></pp:person>
<note>Unavailable</note>
<tuple id="6001">
<contact priority="1">sip:6001@10.24.18.124</contact>
<status><basic>closed</basic></status>
</tuple>
</presence>

---

<--- SIP read from UDP:10.24.17.254:37509 --->
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.24.18.124:5060;branch=z9hG4bK14aacddc
Contact: <sip:Alice@10.24.17.254:37509;transport=UDP>
To: <sip:Alice@10.24.18.124;transport=UDP>;tag=f51e9632
From: <sip:6001@10.24.18.124;transport=UDP>;tag=as46a6e039
Call-ID: ZjE2ZDAwYThiOTA2MzYxOWEwNTEwMjc1ZGIxNTk3NDU.
CSeq: 102 NOTIFY
User-Agent: Z 3.2.21357 r21103
Content-Length: 0

<----->
--- (9 headers 0 lines) ---

```

Once the subscription has taken place, there is a command to list them. "sip show subscriptions"

```
*CLI> sip show subscriptions
Peer      User      Call ID      Extension      Last state      Type      Mailbox      Expiry
10.24.17.254  Alice    ZjE2ZDAwYThiOTA  6001@default    Unavailable     pdf+xml   <none>       001800
1 active SIP subscription
```

From this point onward, Asterisk should send out a SIP NOTIFY to the Alice peer whenever state changes for any of the devices mapped to the hint 6001. Alice's phone should then reflect that state on its display.

# Configuring res\_pjsip

## Overview

This page and its sub-pages are intended to help an administrator configure the new SIP resources and channel driver included with Asterisk 12. The channel driver itself being chan\_pjsip which depends on res\_pjsip and its many associated modules. The **res\_pjsip** module handles configuration, so we'll mostly speak in terms of configuring res\_pjsip.

A variety of reference content is provided in the following sub-pages.

- If you are moving from the old channel driver, then look at [Migrating from chan\\_sip to res\\_pjsip](#).
- For basic config examples look at [res\\_pjsip Configuration Examples](#).
- For detailed explanation of the res\_pjsip config file go to [PJSIP Configuration Sections and Relationships](#).
- You can also find info on [Dialing PJSIP Channels](#).
- Maybe you're migrating to IPv6 and need to learn about [Configuring res\\_pjsip for IPv6](#)

## Before You Configure

This page assumes certain knowledge, or that you have completed a few prerequisites.

- You have [installed pjproject](#), a dependency for res\_pjsip.
- You have Installed Asterisk including the res\_pjsip and chan\_pjsip [modules](#) and their dependencies.
- You understand [basic Asterisk concepts](#). Including the [role of extensions.conf](#) (dialplan) in your overall Asterisk configuration.

## Quick Start

If you like to figure out things as you go; here's a few quick steps to get you started.

- Understand that res\_pjsip is configured through pjsip.conf. This is where you'll be configuring everything related to your inbound or outbound SIP accounts and endpoints.
- Look at the [res\\_pjsip Configuration Examples](#) section. Grab the example most appropriate to your goal and use that to replace your pjsip.conf.
- Reference documentation for all configuration parameters is available on the wiki:
  - [Core res\\_pjsip configuration options](#)
  - [Configuration options for ACLs in res\\_pjsip\\_acl](#)
  - [Configuration options for outbound registration, provided by res\\_pjsip\\_outbound\\_registration](#)
  - [Configuration options for endpoint identification by IP address, provided by res\\_pjsip\\_endpoint\\_identifier\\_ip](#)
- You'll need to tweak details in pjsip.conf and on your SIP device (for example IP addresses and authentication credentials) to get it working with Asterisk.  
Refer back to [the config documentation on the wiki](#) or the sample pjsip.conf if you get confused.



## PJSIP Configuration Sections and Relationships

### Configuration Section Format

pjsip.conf is a flat text file composed of **sections** like most configuration files used with Asterisk. Each **section** defines configuration for a **configuration object** within res\_pjsip or an associated module.

**Sections** are identified by **names in square brackets**. (see [SectionName](#) below)

Each section has one or more **configuration options** that can be assigned a value by using an **equal sign** followed by a value. (see [ConfigOption](#) and [Value](#) below) These options and values are the configuration for a particular component of functionality provided by the configuration object's respective Asterisk modules.

Every section will have a **type** option that defines what kind of section is being configured. You'll see that in every example config section below.

Syntax for res_sip config objects
<pre>[ SectionName ] ConfigOption = Value ConfigOption = Value</pre>
On this Page
<ul style="list-style-type: none"><li>• <a href="#">Configuration Section Format</a></li><li>• <a href="#">Config Section Help and Defaults</a></li><li>• <a href="#">Section Names</a></li><li>• <a href="#">Section Types</a></li><li>• <a href="#">Relationships of Configuration Objects in pjsip.conf</a></li></ul>

### Config Section Help and Defaults

Reference documentation for all configuration parameters is available on the wiki:

- [Core res\\_pjsip configuration options](#)
- [Configuration options for ACLs in res\\_pjsip\\_acl](#)
- [Configuration options for outbound registration, provided by res\\_pjsip\\_outbound\\_registration](#)
- [Configuration options for endpoint identification by IP address, provided by res\\_pjsip\\_endpoint\\_identifier\\_ip](#)

The same documentation is available at the Asterisk CLI as well. You can use "config show help <res\_pjsip module name> <configobject> <configoption>" to get help on a particular option. That help will typically describe the default value for an option as well.



**Defaults:** For many config options, it's very helpful to understand their default behavior. For example, for the endpoint section "transport=" option, if no value is assigned then Asterisk will \*DEFAULT\* to the first configured transport in pjsip.conf which is valid for the URI we are trying to contact.

### Section Names

In most cases, you can name a section whatever makes sense to you. For example you might name a transport [transport-udp-nat] to help you remember how that section is being used.

However, in some cases, (endpoint and aor types) the section name has a relationship to its function. In the case of endpoint and aor their names must match the user portion of the SIP URI in the "To" header for inbound SIP requests. The exception to that rule is if you have an identify section configured for that endpoint. In that case the inbound request would be matched by IP instead of against the user in the "To" header.

### Section Types

Below is a brief description of each section type and an example showing configuration of that section only. The module providing the configuration object related to the section is listed in parentheses next to each section name.

There are dozens of config options for some of the sections, but the examples below are very minimal for the sake of simplicity.

#### ENDPOINT

(provided by module: res\_pjsip)

Endpoint configuration provides numerous options relating to core SIP functionality and ties to other sections such as auth, aor and transport. You can't contact an endpoint without associating one or more AoR sections. An endpoint is essentially a profile for the configuration of a SIP endpoint such as a phone or remote server.

▼ [EXAMPLE BASIC CONFIGURATION](#)

```
[6001]
type=endpoint
context=default
disallow=all
allow=ulaw
transport=simpletrans
auth=auth6001
aors=6001
```

If you want to define the Caller Id this endpoint should use, then add something like the following:

```
trust_id_outbound=yes
callerid=Spaceman Spiff <6001>
```

## TRANSPORT

(provided by module: res\_pjsip)

Configure how res\_pjsip will operate at the transport layer. For example, it supports configuration options for protocols such as TCP, UDP or WebSockets and encryption methods like TLS/SSL.

You can setup multiple transport sections and other sections (such as endpoints) could each use the same transport, or a unique one. However, there are a couple caveats for creating multiple transports:

- They cannot share the same IP+port or IP+protocol combination. That is, each transport that binds to the same IP as another must use a different port or protocol.
- PJSIP does not allow multiple TCP or TLS transports of the same IP version (IPv4 or IPv6).



**Reloading Config:** Configuration for transport type sections can't be reloaded during run-time without a full module unload and load. You'll effectively need to restart Asterisk completely for your transport changes to take effect.

### EXAMPLE BASIC CONFIGURATION

A basic UDP transport bound to all interfaces

```
[simpletrans]
type=transport
protocol=udp
bind=0.0.0.0
```

Or a TLS transport, with many possible options and parameters:

```
[simpletrans]
type=transport
protocol=tls
bind=0.0.0.0
;various TLS specific options below:
cert_file=
priv_key_file=
ca_list_file=
cipher=
method=
```

## AUTH

(provided by module: res\_pjsip)

Authentication sections hold the options and credentials related to inbound or outbound authentication. You'll associate other sections such as endpoints or registrations to this one. Multiple endpoints or registrations can use a single auth config if needed.

### EXAMPLE BASIC CONFIGURATION

An example with username and password authentication

```
[auth6001]
type=auth
auth_type=userpass
password=6001
username=6001
```

And then an example with MD5 authentication

```
[auth6001]
type=auth
auth_type=md5
md5_cred=51e63a3da6425a39aecc045ec45f1ae8
username=6001
```

## AOR

(provided by module: res\_pjsip)

A primary feature of AOR objects (Address of Record) is to tell Asterisk where an endpoint can be contacted. Without an associated AOR section, an endpoint cannot be contacted. AOR objects also store associations to mailboxes for MWI requests and other data that might relate to the whole group of contacts such as expiration and qualify settings.

When Asterisk receives an inbound registration, it'll look to match against available AORs.

**Registrations:** The name of the AOR section must match the user portion of the SIP URI in the "To:" header of the inbound SIP registration. That will usually be the "user name" set in your hard or soft phones configuration.

### ✓ EXAMPLE BASIC CONFIGURATION

First, we have a configuration where you are expecting the SIP User Agent (likely a phone) to register against the AOR. In this case, the contact objects will be created automatically. We limit the maximum contact creation to 1. We could do 10 if we wanted up to 10 SIP User Agents to be able to register against it.

```
[6001]
type=aor
max_contacts=1
```

Second, we have a configuration where you are **not** expecting the SIP User Agent to register against the AOR. In this case, you can assign contacts manually as follows. We don't have to worry about max\_contacts since that option only affects the maximum allowed contacts to be created through external interaction, like registration.

```
[6001]
type=aor
contact=sip:6001@192.0.2.1:5060
```

Third, it's useful to note that you could define only the domain and omit the user portion of the SIP URI if you wanted. Then you could define the **user** portion dynamically in your dialplan when calling the Dial application. You'll likely do this when building an AOR/Endpoint combo to use for dialing out to an ITSP. For example: "Dial(PJSIP/\${EXTEN}@mytrunk)"

```
[mytrunk]
type=aor
contact=sip:203.0.113.1:5060
```

## REGISTRATION

(provided by module: res\_pjsip\_outbound\_registration)

The registration section contains information about an outbound registration. You'll use this when setting up a registration to another system whether it's local or a trunk from your ITSP.

### ✓ EXAMPLE BASIC CONFIGURATION

This example shows you how you might configure registration and outbound authentication against another Asterisk system, where the other system is using the older chan\_sip peer setup.

This example is just the registration itself. You'll of course need the associated transport and auth sections. Plus, if you want to receive calls from the far end (who now knows where to send calls, thanks to your registration!) then you'll need endpoint, AOR and possibly identify sections setup to match inbound calls to a context in your dialplan.

```
[mytrunk]
type=registration
transport=simpletrans
outbound_auth=mytrunk
server_uri=sip:myaccountname@203.0.113.1:5060
client_uri=sip:myaccountname@192.0.2.1:5060
retry_interval=60
```

And an example that may work with a SIP trunking provider

```
[mytrunk]
type=registration
transport=simpletrans
outbound_auth=mytrunk
server_uri=sip:sip.example.com
client_uri=sip:1234567890@sip.example.com
retry_interval=60
```

What if you don't need to authenticate? You can simply omit the **outbound\_auth** option.

## DOMAIN\_ALIAS

(provided by module: res\_pjsip)

Allows you to specify an alias for a domain. If the domain on a session is not found to match an AoR then this object is used to see if we have an alias for the AoR to which the endpoint is binding. This sections name as defined in configuration should be the domain alias and a config option (domain=) is provided to specify the domain to be aliased.

### ▼ EXAMPLE BASIC CONFIGURATION

```
[example2.com]
type=domain_alias
domain=example.com
```

## ACL

(provided by module: res\_pjsip\_acl)

The ACL module used by 'res\_pjsip'. This module is independent of 'endpoints' and operates on all inbound SIP communication using res\_pjsip. Features such as an Access Control List, as defined in the configuration section itself, or as defined in **acl.conf**. ACL's can be defined specifically for source IP addresses, or IP addresses within the contact header of SIP traffic.

### ▼ EXAMPLE BASIC CONFIGURATION

A configuration pulling from the acl.conf file:

```
[acl]
type=acl
acl=example_named_acl1
```

A configuration defined in the object itself:

```
[acl]
type=acl
deny=0.0.0.0/0.0.0.0
permit=209.16.236.0
permit=209.16.236.1
```

A configuration where we are restricting based on contact headers instead of IP addresses.

```
[acl]
type=acl
contactdeny=0.0.0.0/0.0.0.0
contactpermit=209.16.236.0
contactpermit=209.16.236.1
```

All of these configurations can be combined.

## IDENTIFY

(provided by module: res\_pjsip\_endpoint\_identifier\_ip)

Controls how the res\_pjsip\_endpoint\_identifier\_ip module determines what endpoint an incoming packet is from. If you don't have an identify section defined, or else you have res\_pjsip\_endpoint\_\*\*identifier\_ip\*\* loading **after** res\_pjsip\_endpoint\_\*\*identifier\_user\*\*, then res\_pjsip\_endpoint\_\*\*identifier\_user\*\* will identify inbound traffic by pulling the user from the "From:" SIP header in the packet. Basically the module load order, and your configuration will both determine whether you identify by IP or by user.

### ▼ EXAMPLE BASIC CONFIGURATION

Its use is quite straightforward. With this configuration if Asterisk sees inbound traffic from 203.0.113.1 then it will match that to Endpoint 6001.

```
[6001]
type=identify
endpoint=6001
match=203.0.113.1
```

## CONTACT

(provided by module: res\_pjsip)

The contact config object effectively acts as an alias for a SIP URIs and holds information about an inbound registrations. Contact objects can be associated with an individual SIP User Agent and contain a few config options related to the connection. Contacts are created automatically upon registration to an AOR, or can be created manually by using the "contact=" config option in an AOR section. Manually configuring a CONTACT config object itself is outside the scope of this "getting started" style document.

### ***Relationships of Configuration Objects in pjsip.conf***

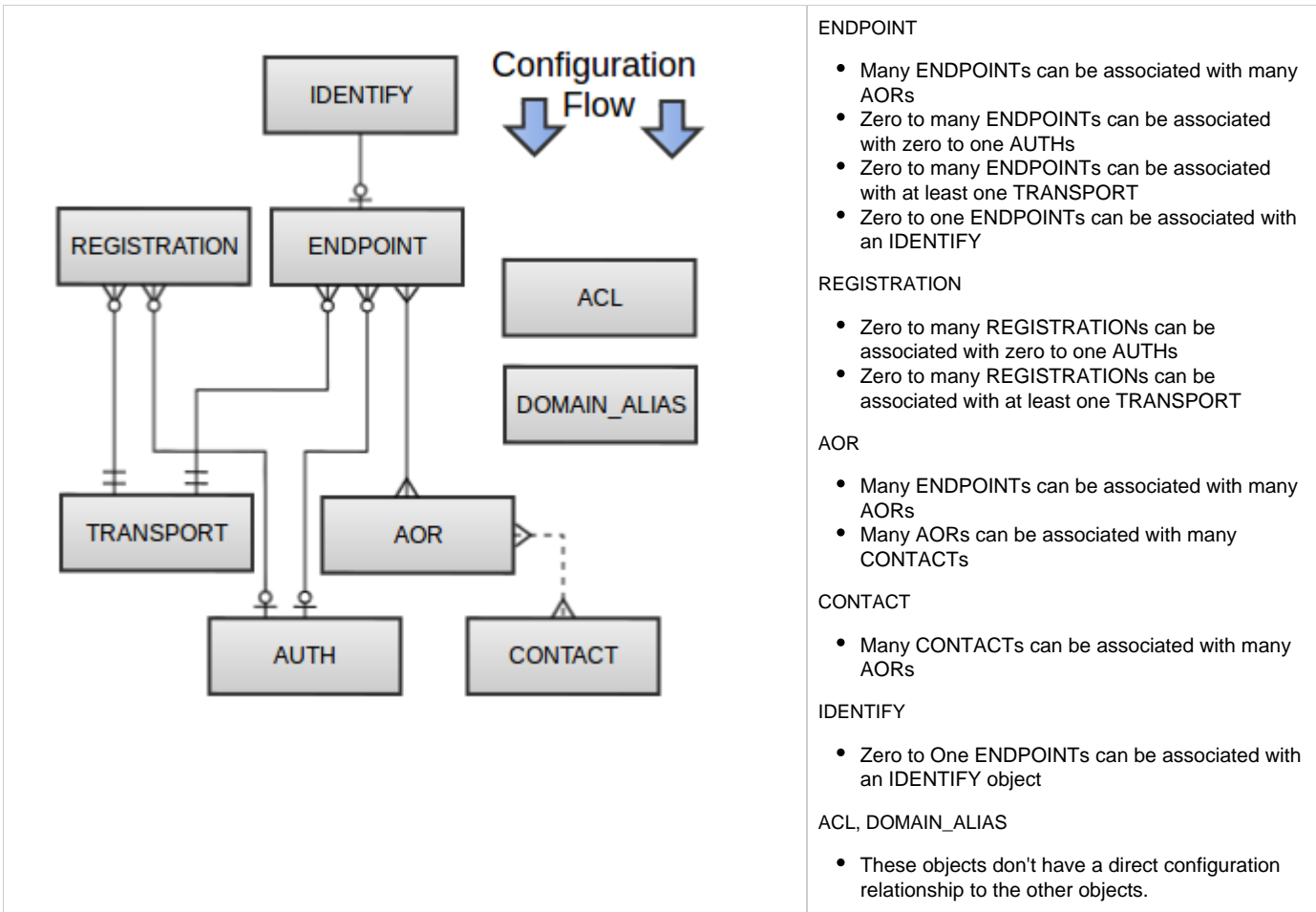
Now that you understand the various configuration sections related to each config object, lets look at how they interrelate.

You'll see that the new SIP implementation within Asterisk is extremely flexible due to its modular design. A diagram will help you to visualize the relationships between the various configuration objects. The following entity relationship diagram covers only the configuration relationships between the objects. For example if an **endpoint** object requires authorization for registration of a SIP device, then you may associate a single **auth** object with the endpoint object. Though many endpoints could use the same or different auth objects.

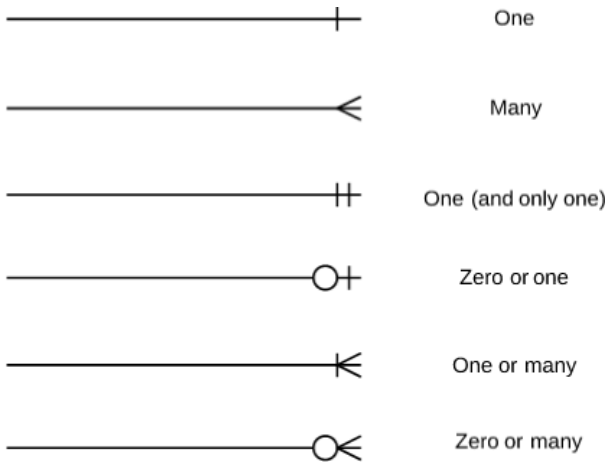
**Configuration Flow:** This lets you know which direction the objects are associated to other objects. e.g. The identify config section has an option "endpoint=" which allows you to associate it with an endpoint object.

**Entity Relationships**

**Relationship Descriptions**



Unfamiliar with ERD? Click here to see a key...



## res\_pjsip Configuration Examples

Below are some sample configurations to demonstrate various scenarios with complete pjsip.conf files. To see examples side by side with old chan\_sip config head to [Migrating from chan\\_sip to res\\_pjsip](#). Explanations of the config sections found in each example can be found in [PJSIP Configuration Sections and Relationships](#).

A tutorial on secure and encrypted calling is located in the [Secure Calling](#) section of the wiki.

### **An endpoint with a single SIP phone with inbound registration to Asterisk**

#### ✓ EXAMPLE CONFIGURATION

```
;=====TRANSPORT

[simpletrns]
type=transport
protocol=udp
bind=0.0.0.0

;=====EXTENSION 6001

[6001]
type=endpoint
context=internal
disallow=all
allow=ulaw
transport=simpletrns
auth=auth6001
aors=6001

[auth6001]
type=auth
auth_type=userpass
password=6001
username=6001

[6001]
type=aor
max_contacts=1
```

- `auth=` is used for the endpoint as opposed to `outbound_auth=` since we want to allow inbound registration for this endpoint
- `max_contacts=` is set to something non-zero as we want to allow contacts to be created through registration

#### On this Page

- [An endpoint with a single SIP phone with inbound registration to Asterisk](#)
- [A SIP trunk to your service provider, including outbound registration](#)
- [Multiple endpoints with phones registering to Asterisk, using templates](#)

### **A SIP trunk to your service provider, including outbound registration**

#### ✓ EXAMPLE CONFIGURATION

Trunks are a little tricky since many providers have unique requirements. Your final configuration may differ from what you see here.

```

;=====TRANSPORTS

[simpletrans]
type=transport
protocol=udp
bind=0.0.0.0

;=====TRUNK

[mytrunk]
type=registration
transport=simpletrans
outbound_auth=mytrunk
server_uri=sip:sip.example.com
client_uri=sip:1234567890@sip.example.com
retry_interval=60

[mytrunk]
type=auth
auth_type=userpass
password=1234567890
username=1234567890

[mytrunk]
type=aor
contact=sip:203.0.113.1:5060

[mytrunk]
type=endpoint
transport=simpletrans
context=from-external
disallow=all
allow=ulaw
outbound_auth=mytrunk
aors=mytrunk

[mytrunk]
type=identify
endpoint=mytrunk
match=203.0.113.1

```

- "contact=sip:203.0.113.1:5060", we don't define the user portion statically since we'll set that dynamically in dialplan when we call the Dial application. See the dialing examples in the section "Dialing using chan\_pjsip" for more.
- "outbound\_auth=mytrunk", we use "outbound\_auth" instead of "auth" since the provider isn't typically going to authenticate with us when calling, but we will probably have to authenticate when calling through them.
- We use an identify object to map all traffic from the provider's IP as traffic to that endpoint since the user portion of their From: header may vary with each call.
- This example assumes that [sip.example.com](http://sip.example.com) resolves to 203.0.113.1

### **Multiple endpoints with phones registering to Asterisk, using templates**

#### ▼ EXAMPLE CONFIGURATION

We want to show here that generally, with a large configuration you'll end up using templates to make configuration easier to handle when scaling. This avoids having redundant code in every similar section that you create.

```

;=====TRANSPORT

[simpletrans]
type=transport

```



```
protocol=udp
bind=0.0.0.0

;=====ENDPOINT TEMPLATES

[endpoint-basic](!)
type=endpoint
transport=simpletrans
context=internal
disallow=all
allow=ulaw

[auth-userpass](!)
type=auth
auth_type=userpass

[aor-single-reg](!)
type=aor
max_contacts=1

;=====EXTENSION 6001

[6001](endpoint-basic)
auth=auth6001
aors=6001

[auth6001](auth-userpass)
password=6001
username=6001

[6001](aor-single-reg)

;=====EXTENSION 6002

[6002](endpoint-basic)
auth=auth6002
aors=6002

[auth6002](auth-userpass)
password=6002
username=6002

[6002](aor-single-reg)

;=====EXTENSION 6003

[6003](endpoint-basic)
auth=auth6003
aors=6003

[auth6003](auth-userpass)
password=6003
```

```
username=6003
```

```
[6003](aor-single-reg)
```

Obviously the larger your configuration is, the more templates will benefit you. Here we just break apart the endpoints with templates, but you could do that with any config section that needs instances with variation, but where each may share common settings with their peers.

## Migrating from chan\_sip to res\_pjsip

### Overview

This page documents any useful tools, tips or examples on moving from the old chan\_sip channel driver to the new chan\_pjsip/res\_pjsip added in Asterisk 12.

### Configuration Conversion Script

There is a [script](#) available to provide a basic conversion of a sip.conf config to a pjsip.conf config. It is not intended to work for every scenario or configuration; for basic configurations it should provide a good example of how to convert it over to pjsip.conf style config.

To insure that the script can read any `#include'd` files, run it from the `/etc/asterisk` directory or in another location with a copy of the sip.conf and any included files. The default input file is sip.conf, and the default output file is pjsip.conf. Any included files will also be converted, and written out with a pjsip\_ prefix, unless changed with the `--prefix=xxx` option.

### Command line usage

```
# /path/to/asterisk/source/contrib/scripts/sip_to_pjsip/sip_to_pjsip.py --help
Usage: sip_to_pjsip.py [options] [input-file [output-file]]
input-file defaults to 'sip.conf'
output-file defaults to 'pjsip.conf'
Options:
-h, --help            show this help message and exit
-p PREFIX, --prefix=PREFIX
                       output prefix for include files
```

### Example of Use

```
# cd /etc/asterisk
# /path/to/asterisk/source/contrib/scripts/sip_to_pjsip/sip_to_pjsip.py
Reading sip.conf
Converting to PJSIP...
Writing pjsip.conf
```

### On this Page

- Overview
  - Configuration Conversion Script
- Side by Side Examples of sip.conf and pjsip.conf Configuration
  - Example Endpoint Configuration
  - Example SIP Trunk Configuration
- Disabling res\_pjsip and chan\_pjsip
- Network Address Translation (NAT)

### Side by Side Examples of sip.conf and pjsip.conf Configuration

These examples contain only the configuration required for sip.conf/pjsip.conf as the configuration for other files should be the same, excepting the Dial statements in your extensions.conf. Dialing with PJSIP is discussed in [Dialing PJSIP Channels](#).



It is important to know that PJSIP syntax and configuration format is stricter than the older chan\_sip driver. When in doubt, try to follow the documentation exactly, avoid extra spaces or strange capitalization. Always check your logs for warnings or errors if you suspect something is wrong.

### Example Endpoint Configuration

This examples shows the configuration required for:

- two SIP phones need to make calls to or through Asterisk, we also want to be able to call them from Asterisk
- for them to be identified as users (in the old chan\_sip) or endpoints (in the new res\_sip/chan\_pjsip)
- both devices need to use username and password authentication
- 6001 is setup to allow registration to Asterisk, and 6002 is setup with a static host/contact

sip.conf

pjsip.conf

```
[general]
udpbindaddr=0.0.0.0

[6001]
type=friend
host=dynamic
disallow=all
allow=ulaw
context=internal
secret=1234

[6002]
type=friend
host=192.0.2.1
disallow=all
allow=ulaw
context=internal
secret=1234
```

```
[simpletrans]
type=transport
protocol=udp
bind=0.0.0.0

[6001]
type = endpoint
transport = simpletrans
context = internal
disallow = all
allow = ulaw
aors = 6001
auth = auth6001

[6001]
type = aor
max_contacts = 1

[auth6001]
type=auth
auth_type=userpass
password=1234
username=6001

[6002]
type = endpoint
transport = simpletrans
context = internal
disallow = all
allow = ulaw
aors = 6002
auth = auth6002

[6002]
type = aor
contact = sip:6002@192.0.2.1:5060

[auth6002]
type=auth
auth_type=userpass
password=1234
username=6001
```

### Example SIP Trunk Configuration

This shows configuration for a SIP trunk as would typically be provided by an ITSP. That is registration to a remote server, authentication to it and a peer/endpoint setup to allow inbound calls from the provider.

- SIP provider requires registration to their server with a username of "myaccountname" and a password of "1234567890"
- SIP provider requires registration to their server at the address of 203.0.113.1:5060
- SIP provider requires outbound calls to their server at the same address of registration, plus using same authentication details.
- SIP provider will call your server with a user name of "mytrunk". Their traffic will only be coming from 203.0.113.1

sip.conf

pjsip.conf

```
[general]
udpbindaddr=0.0.0.0

register => myaccountname:1234567890@203.0.113.1:5060

[mytrunk]
type=friend
secret=1234567890
username=myaccountname
host=203.0.113.1
disallow=all
allow=ulaw
context=from-external
```

```
[simpletrans]
type=transport
protocol=udp
bind=0.0.0.0

[mytrunk]
type=registration
transport=simpletrans
outbound_auth=mytrunk
server_uri=sip:myaccountname@20:
client_uri=sip:myaccountname@19:

[mytrunk]
type=auth
auth_type=userpass
password=1234567890
username=myaccountname

[mytrunk]
type=aor
contact=sip:203.0.113.1:5060

[mytrunk]
type=endpoint
transport=simpletrans
context=from-external
disallow=all
allow=ulaw
outbound_auth=mytrunk
aors=mytrunk

[mytrunk]
type=identify
endpoint=mytrunk
match=203.0.113.1
```

## Disabling res\_pjsip and chan\_pjsip

You may want to keep using chan\_sip for a short time in Asterisk 12+ while you migrate to res\_pjsip. In that case, it is best to disable res\_pjsip unless you understand how to configure them both together.

There are several methods to disable or remove modules in Asterisk. Which method is best depends on your intent.

If you have built Asterisk with the PJSIP modules, but don't intend to use them at this moment, you might consider the following:

1. Edit the file **modules.conf** in your Asterisk configuration directory. (typically `/etc/asterisk/`)

```
noload => res_pjsip.so
noload => res_pjsip_pubsub.so
noload => res_pjsip_session.so
noload => chan_pjsip.so
noload => res_pjsip_exten_state.so
noload => res_pjsip_log_forwarder.so
```

Having a noload for the above modules should (at the moment of writing this) prevent any PJSIP related modules from loading.

2. Restart Asterisk!

Other possibilities would be:

- Remove all PJSIP modules from the modules directory (often, /usr/lib/asterisk/modules)
- Remove the configuration file (pjsip.conf)
- Un-install and re-install Asterisk with no PJSIP related modules.
- If you are wanting to use chan\_pjsip alongside chan\_sip, you could change the port or bind interface of your chan\_pjsip transport in pjsip.conf

## Network Address Translation (NAT)

When configured with **chan\_sip**, peers that are, relative to Asterisk, located behind a NAT are configured using the **nat** parameter. In versions 1.8 and greater of Asterisk, the following nat parameter options are available:

Value	Description
no	Do not perform NAT handling other than RFC 3581.
force_rport	When the rport parameter is not present, send responses to the source IP address and port anyway, as though the rport parameter was present
comedia	Send media to the address and port from which Asterisk received it, regardless of where SDP indicates that it should be sent
auto_force_rport	Automatically enable the sending of responses to the source IP address and port, as though rport were present, if Asterisk detects NAT. Default.
auto_comedia	Automatically send media to the port from which Asterisk received it, regardless of where SDP indicates that it should be sent, if Asterisk detects NAT.

Versions of Asterisk prior to 1.8 had less granularity for the nat parameter:

Value	Description
no	Do not perform NAT handling other than RFC 3581
yes	Send media to the port from which Asterisk received it, regardless of where SDP indicates that it should be sent; send responses to the source IP address and port as though rport were present; and rewrite the SIP Contact to the source address and port of the request so that subsequent requests go to that address and port.
never	Do not perform any NAT handling
route	Send media to the port from which Asterisk received it, regardless of where SDP indicates that it should be sent and rewrite the SIP Contact to the source address and port of the request so that subsequent requests go to that address and port.

In **chan\_pjsip**, the **endpoint** options that control NAT behavior are:

- rtp\_symmetric - Send media to the address and port from which Asterisk receives it, regardless of where SDP indicates that it should be sent
- force\_rport - Send responses to the source IP address and port as though port were present, even if it's not
- rewrite\_contact - Rewrite SIP Contact to the source address and port of the request so that subsequent requests go to that address and port.

Thus, the following are equivalent:

<b>chan_sip (sip.conf)</b>	<b>chan_pjsip (pjsip.conf)</b>
----------------------------	--------------------------------

<pre>[mypeer1] type=peer nat=yes i...</pre>	<pre>[mypeer1] type=endpoint rtp_symmetric=yes force_rport=yes rewrite_contact=yes i...</pre>
<pre>[mypeer2] type=peer nat=no i...</pre>	<pre>[mypeer2] type=endpoint rtp_symmetric=no force_rport=no rewrite_contact=no i...</pre>
<pre>[mypeer3] type=peer nat=never i...</pre>	<pre>[mypeer3] type=endpoint rtp_symmetric=no force_rport=no rewrite_contact=no i...</pre>
<pre>[mypeer4] type=peer nat=route i...</pre>	<pre>[mypeer4] type=endpoint rtp_symmetric=no force_rport=yes rewrite_contact=yes i...</pre>

## Dialing PJSIP Channels

### Dialing from dialplan

We are assuming you already know a little bit about the Dial application here. To see the full help for it, see "core show help application dial" on the Asterisk CLI, or see [Application\\_Dial](#)

Below we'll simply dial an endpoint using the chan\_pjsip channel driver. This is really going to look at the AOR of the same name as the endpoint and start dialing the first contact associated.

```
exten => _6XXX,1,Dial(PJSIP/${EXTEN})
```

To dial all the contacts associated with the endpoint, use the PJSIP\_DIAL\_CONTACTS function. It evaluates to a list of contacts separated by &, which causes the Dial application to call them simultaneously.

```
exten => _6XXX,1,Dial(${PJSIP_DIAL_CONTACTS(${EXTEN})})
```

Here's how you would dial with an explicit SIP URI, user and domain, via an endpoint (in this case dialing out a trunk), but not using its associated AOR/contact objects.

```
exten => _9NXXNXXXXXX,1,Dial(PJSIP/mytrunk/sip:${EXTEN:1}@203.0.113.1:5060)
```

This uses a contact (and its domain) set in the AOR associated with the **mytrunk** endpoint, but still explicitly sets the user portion of the URI in the dial string. For the AOR's contact, you would define it in the AOR config without the user name.

```
exten => _9NXXNXXXXXX,1,Dial(PJSIP/${EXTEN:1}@mytrunk)
```



## Configuring res\_pjsip to work through NAT

Here we can show some examples of working configuration for Asterisk's SIP channel driver when Asterisk is behind NAT (Network Address Translation).

If you are migrating from chan\_sip to chan\_pjsip, then also read the NAT section in [Migrating from chan\\_sip to res\\_pjsip](#) for helpful tips.

### Asterisk and Phones Connecting Through NAT to an ITSP

This example should apply for most simple NAT scenarios that meet the following criteria:

- Asterisk and the phones are on a private network.
- There is a router interfacing the private and public networks. Where the public network is the Internet.
- The router is performing Network Address Translation and Firewall functions.
- The router is configured for port-forwarding, where it is mapping the necessary ranges of SIP and RTP traffic to your internal Asterisk server.

In this example the router is port-forwarding WAN inbound TCP/UDP 5060 and UDP 10000-20000 to LAN 192.0.2.10

This example was based on a configuration for the ITSP [SIP.US](#) and assuming you swap out the addresses and credentials for real ones, it should work for a SIP.US SIP account.

#### Devices Involved in the Example

Using [RFC5737](#) documentation addresses

Device	IP in example
VOIP Phone(6001)	192.0.2.20
PC/Asterisk	192.0.2.10
Router	LAN: 192.0.2.1 WAN: 198.51.100.5
ITSP SIP gateway	203.0.113.1 (gw1.example.com) 203.0.113.2 (gw2.example.com)

For the sake of a complete example and clarity, in this example we use the following fake details:

ITSP Account number: 1112223333

DID number provided by ITSP: 19998887777

#### pjsip.conf Configuration

We are assuming you have already read the [Configuring res\\_pjsip](#) page and have a basic understanding of Asterisk. For this NAT example, the important config options to note are **local\_net**, **external\_media\_address** and **external\_signaling\_address** in the transport type section and **direct\_media** in the endpoint section. The rest of the options may depend on your particular configuration, phone model, network settings, ITSP, etc. The key is to make sure you have those three options set appropriately.

**local\_net**

This is the IP network that we want to consider our local network. For communication to addresses within this range, we won't apply any NAT-related settings, such as the external\* options below.

**external\_media\_address**

This is the external IP address to use in RTP handling. When a request or response is sent out from Asterisk, if the destination of the message is outside the IP network defined in the option 'local\_net', and the media address in the SDP is within the localnet network, then the media address in the SDP will be rewritten to the value defined for 'external\_media\_address'.

**external\_signaling\_address**

This is much like the external\_media\_address setting, but for SIP signaling instead of RTP media. The two external\* options mentioned here should be set to the same address unless you separate your signaling and media to different addresses or servers.

**direct\_media**

Determines whether media may flow directly between endpoints

Together these options make sure the far end knows where to send back SIP and RTP packets, and direct\_media ensures Asterisk stays in the media path. This is important, because our Asterisk system has a private IP address that the ITSP cannot route to. We want to make sure the SIP and RTP traffic comes back to the WAN/Public internet address of our router. The sections prefixed with "sipus" are all configuration needed for inbound and outbound connectivity of the SIP trunk, and the sections named 6001 are all for the VOIP phone.

```
[transport-udp-nat]
type=transport
protocol=udp
bind=0.0.0.0
local_net=192.0.2.0/24
local_net=127.0.0.1/32
external_media_address=198.51.100.5
external_signaling_address=198.51.100.5

[sipus_reg]
type=registration
transport=transport-udp-nat
outbound_auth=sipus_auth
server_uri=sip:gw1.example.com
client_uri=sip:1112223333@gw1.example.com
contact_user=19998887777
retry_interval=60

[sipus_auth]
type=auth
auth_type=userpass
password=*****
username=1112223333
realm=gw1.example.com

[sipus_endpoint]
type=endpoint
transport=transport-udp-nat
context=from-external
disallow=all
allow=ulaw
outbound_auth=sipus_auth
aors=sipus_aor
direct_media=no
from_domain=gw1.example.com

[sipus_aor]
type=aor
contact=sip:gw1.example.com
contact=sip:gw2.example.com

[sipus_identify]
type=identify
endpoint=sipus_endpoint
match=203.0.113.1
match=203.0.113.2

[6001]
type=endpoint
context=from-internal
disallow=all
allow=ulaw
transport=transport-udp-nat
auth=6001
aors=6001
direct_media=no

[6001]
type=auth
```

```
auth_type=userpass  
password=*****  
username=6001
```

```
[6001]
type=aor
max_contacts=2
```

### **For Remote Phones Behind NAT**

In the above example we assumed the phone was on the same local network as Asterisk. Now, perhaps Asterisk is exposed on a public address, and instead your phones are remote and behind NAT, or maybe you have a double NAT scenario?

In these cases you will want to consider the below settings for the remote endpoints.  
media\_address

IP address used in SDP for media handling

At the time of SDP creation, the IP address defined here will be used as the media address for individual streams in the SDP.

NOTE: Be aware that the 'external\_media\_address' option, set in Transport configuration, can also affect the final media address used in the SDP.

rtp\_symmetric

Enforce that RTP must be symmetric. Send RTP back to the same address/port we received it from.

force\_rport

Force RFC3581 compliant behavior even when no rport parameter exists. Basically always send SIP responses back to the same port we received SIP requests from.

direct\_media

Determines whether media may flow directly between endpoints.

### ***Clients Supporting ICE,STUN,TURN***

This is really relevant to media, so look to the [section here](#) for basic information on enabling this support and we'll add relevant examples later.

## Setting up PJSIP Realtime

- Overview
  - Installing Dependencies
  - Creating the MySQL Database
  - Installing and Using Alembic
  - Configuring ODBC
  - Connecting PJSIP Sorcery to the Realtime Database
  - Optionally configuring sorcery for realtime and non-realtime data sources
  - Realtime Configuration
  - Asterisk Startup Configuration
  - Asterisk PJSIP configuration
  - Endpoint Population
  - A Little Dialplan
  - Conclusion

### Overview

This tutorial describes the configuration of Asterisk's PJSIP channel driver with the "realtime" database storage backend. The realtime interface allows storing much of the configuration of PJSIP, such as endpoints, auths, aors and more, in a database, as opposed to the normal flat-file storage of pjsip.conf.

### Installing Dependencies

For the purposes of this tutorial, we will assume a base Ubuntu 12.0.4.3 x86\_64 server installation, with the OpenSSH server and LAMP server options, and that Asterisk will use its ODBC connector to reach a back-end MySQL database.

Beyond the normal packages needed to install Asterisk 12 on such a server (build-essential, libncurses5-dev, uuid-dev, libjansson-dev, libxml2-dev, libsquid3-dev) as well as the [Installation of pjproject](#), you will need to install the following packages:

- unixodbc and unixodbc-dev
  - ODBC and the development packages for building against ODBC
- libmyodbc
  - The ODBC to MySQL interface package
- python-dev and python-pip
  - The Python development package and the pip package to allow installation of Alembic
- python-mysqldb
  - The Python interface to MySQL, which will be used by Alembic to generate the database tables

So, from the CLI, perform:

```
# apt-get install unixodbc unixodbc-dev libmyodbc python-dev python-pip python-mysqldb
```

Once these packages are installed, check your Asterisk installation's **make menuconfig** tool to make sure that the **res\_config\_odbc** and **res\_odbc** resource modules, as well as the **res\_pjsip\_XXX** modules are selected for installation. If they are, then go through the normal Asterisk installation process: **.configure; make; make install**

And, if this is your first installation of Asterisk, be sure to install the sample files: **make samples**

### Creating the MySQL Database

Use the **mysqladmin** tool to create the database that we'll use to store the configuration. From the Linux CLI, perform:

```
# mysqladmin -u root -p create asterisk
```

This will prompt you for your MySQL database password and then create a database named **asterisk** that we'll use to store our PJSIP configuration.

### Installing and Using Alembic

Alembic is a full database migration tool, with support for upgrading the schemas of existing databases, versioning of schemas, creation of new tables and databases, and a whole lot more. A good guide on using Alembic with Asterisk can be found on the [Managing Realtime Databases with Alembic](#) wiki page. A shorter discussion of the steps necessary to prep your database will follow.

First, install Alembic:

```
# pip install alembic
```

Then, move to the Asterisk source directory containing the Alembic scripts:

```
# cd contrib/ast-db-manage/
```

Next, edit the **config.ini.sample** file and change the **sqlalchemy.url** option, e.g.

```
sqlalchemy.url = mysql://root:password@localhost/asterisk
```

such that the URL matches the username and password required to access your database.

Then rename the config.ini.sample file to config.ini

```
# cp config.ini.sample config.ini
```

Finally, use Alembic to setup the database tables:

```
# alembic -c config.ini upgrade head
```

You'll see something similar to:

```
# alembic -c config.ini upgrade head
INFO [alembic.migration] Context impl MySQLImpl.
INFO [alembic.migration] Will assume non-transactional DDL.
INFO [alembic.migration] Running upgrade None -> 4da0c5f79a9c, Create tables
INFO [alembic.migration] Running upgrade 4da0c5f79a9c -> 43956d550a44, Add tables for pjsip
#
```

You can then connect to MySQL to see that the tables were created:

```
# mysql -u root -p -D asterisk

mysql> show tables;
+-----+
| Tables_in_asterisk |
+-----+
| alembic_version    |
| iaxfriends         |
| meetme             |
| musiconhold        |
| ps_aors             |
| ps_auths           |
| ps_contacts        |
| ps_domain_aliases  |
| ps_endpoint_id_ips |
| ps_endpoints       |
| sippeers           |
| voicemail           |
+-----+
12 rows in set (0.00 sec)
mysql> quit
```

## Configuring ODBC

Now that we have our MySQL database created and populated, we'll need to setup ODBC and Asterisk's ODBC resource to access the database. First, we'll tell ODBC how to connect to MySQL. To do this, we'll edit the `/etc/odbcinst.ini` configuration file. Your file should look something like:

### `/etc/odbcinst.ini`

```
[MySQL]
Description = ODBC for MySQL
Driver = /usr/lib/x86_64-linux-gnu/odbc/libmyodbc.so
Setup = /usr/lib/x86_64-linux-gnu/odbc/libodbcmyS.so
UsageCount = 2
```

Next, we'll tell ODBC **which** MySQL database to use. To do this, we'll edit the `/etc/odbc.ini` configuration file and create a database handle called **asterisk**. Your file should look something like:

### **/etc/odbc.ini**

```
[asterisk]
Driver = MySQL
Description = MySQL connection to 'asterisk' database
Server = localhost
Port = 3306
Database = asterisk
UserName = root
Password = password
Socket = /var/run/mysqld/mysqld.sock
```

Take care to use your database access UserName and Password, and not necessarily what's defined in this example.

Now, we need to configure Asterisk's ODBC resource, `res_odbc`, to connect to the ODBC **asterisk** database handle that we just created. `res_odbc` is configured using the `/etc/asterisk/res_odbc.conf` configuration file. There, you'll want:

### **/etc/asterisk/res\_odbc.conf**

```
[asterisk]
enabled => yes
dsn => asterisk
username => root
password => password
pre-connect => yes
```

Again, take care to use the proper username and password.

Now, you can start Asterisk and you can check its connection to your "asterisk" MySQL database using the "asterisk" `res_odbc` connector to ODBC. You can do this by executing "odbc show" from the Asterisk CLI. If everything went well, you'll see:

```
# asterisk -vvvvv
*CLI> odbc show

ODBC DSN Settings
-----
Name: asterisk
DSN: asterisk
Last connection attempt: 1969-12-31 18:00:00
Pooled: No
Connected: Yes
*CLI>
```

## **Connecting PJSIP Sorcery to the Realtime Database**

The PJSIP stack uses a new data abstraction layer in Asterisk called **sorcery**. Sorcery lets a user build a hierarchical layer of data sources for Asterisk to use when it retrieves, updates, creates, or destroys data that it interacts with. This tutorial focuses on getting PJSIP's configuration stored in a realtime back-end; the rest of the details of sorcery are beyond the scope of this page.

PJSIP bases its configuration on types of objects. For more information about these types of objects, please refer to the [Configuring res\\_pjsip](#) wiki page. In this case, we have a total of five objects we need to configure in Sorcery:

- endpoint
- auth
- aor
- domain
- identify

We'll also configure the contact object, though we don't need it for this example.

Sorcery is configured using the `/etc/asterisk/sorcery.conf` configuration file. So, we need to add the following lines to the file:

## **/etc/asterisk/sorcery.conf**

```
[res_pjsip] ; Realtime PJSIP configuration wizard
endpoint=realtime,ps_endpoints
auth=realtime,ps_auths
aor=realtime,ps_aors
domain_alias=realtime,ps_domain_aliases
contact=realtime,ps_contacts

[res_pjsip_endpoint_identifier_ip]
identify=realtime,ps_endpoint_id_ips
```

The items use the following nomenclature:

```
{object_type} = {sorcery_wizard_name},{wizard_arguments}
```

In our case, the `sorcery_wizard_name` is **realtime**, and the **wizard\_arguments** are the name of the database connector ("asterisk") to associate with our object types. Note that the "identify" object is separated from the rest of the configuration objects. This is because this object type is provided by an optional module (`res_pjsip_endpoint_identifier_ip.so`) and not the main PJSIP module (`res_pjsip.so`).

### **Optionally configuring sorcery for realtime and non-realtime data sources**

If you want to configure **both realtime and static configuration** file lookups for PJSIP then you need to add additional lines to the sorcery config.

For example if you want to read **endpoints** from both realtime and static configuration:

```
endpoint=realtime,ps_endpoints
endpoint=config,pjsip.conf,criteria=type=endpoint
```

You can swap the order to control which data source is read first.

### **Realtime Configuration**

Since we've associated the PJSIP objects with database connector types, we now need to tell Asterisk to use a database backend with the object types, and not just the flat `pjsip.conf` file. To do this, we modify the **/etc/asterisk/extconfig.conf** configuration file to provide these connections.

Open `extconfig.conf` (`/etc/asterisk/extconfig.conf`) and add the following lines to the 'settings' configuration section

## **/etc/asterisk/extconfig.conf**

```
[settings]
ps_endpoints => odbc,asterisk
ps_auths => odbc,asterisk
ps_aors => odbc,asterisk
ps_domain_aliases => odbc,asterisk
ps_endpoint_id_ips => odbc,asterisk
ps_contacts => odbc,asterisk
```



Other tables allowed but not demonstrated in this tutorial: `ps_systems`, `ps_globals`, `ps_transports`, and `ps_registrations`.

At this point, Asterisk is nearly ready to use the tables created by alembic with PJSIP to configure endpoints, authorization, AORs, domain aliases, and endpoint identifiers.



#### **A warning for adventurous types:**

Sorcery.conf allows you to try to configure other PJSIP objects such as transport using realtime and it currently won't stop you from doing so. However, some of these object types should not be used with realtime and this can lead to errant behavior.

### **Asterisk Startup Configuration**



Now, we need to configure Asterisk to load its ODBC driver at an early stage of startup, so that it's available when any other modules might need to take advantage of it. Also, we're going to prevent the old `chan_sip` channel driver from loading, since we're only worried about PJSIP.

To do this, edit the `/etc/asterisk/modules.conf` configuration file. In the `[modules]` section, add the following lines:

```
/etc/asterisk/modules.conf

preload => res_odbc.so
preload => res_config_odbc.so
noload => chan_sip.so
```

### Asterisk PJSIP configuration

Next, we need to configure a transport in `/etc/asterisk/pjsip.conf`. PJSIP transport object types are not stored in realtime as unexpected results can occur. So, edit it and add the following lines:

```
/etc/asterisk/pjsip.conf

[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0
```

Here, we created a transport called `transport-udp` that we'll reference in the next section.

### Endpoint Population

Now, we need to create our endpoints inside of the database. For this example, we'll create two peers, 101 and 102, that register using the totally insecure passwords "101" and "102" respectively. Here, we'll be populating data directly into the database using the MySQL interactive tool.

```
# mysql -u root -p -D asterisk;
mysql> insert into ps_aors (id, max_contacts) values (101, 1);
mysql> insert into ps_aors (id, max_contacts) values (102, 1);
mysql> insert into ps_auths (id, auth_type, password, username) values (101, 'userpass', 101, 101);
mysql> insert into ps_auths (id, auth_type, password, username) values (102, 'userpass', 102, 102);
mysql> insert into ps_endpoints (id, transport, aors, auth, context, disallow, allow, direct_media) values (101, 'transport-udp',
'101', '101', 'testing', 'all', 'g722', 'no');
mysql> insert into ps_endpoints (id, transport, aors, auth, context, disallow, allow, direct_media) values (102, 'transport-udp',
'102', '102', 'testing', 'all', 'g722', 'no');
mysql> quit;
```

In this example, we first created an `aor` for each peer, one called `101` and the other `102`.

Next, we created an `auth` for each peer with a `userpass` of `101` and `102`, respectively.

Then, we created two endpoints, `101` and `102`, each referencing the appropriate `auth` and `aor`, we selected the G.722 codec and we forced media to route inside of Asterisk (not the default behavior of Asterisk).

Now, you can start Asterisk and you can check to see if it's finding your PJSIP endpoints in the database. You can do this by executing "pjsip show endpoints" from the Asterisk CLI. If everything went well, you'll see:

```
# asterisk -vvvvc
*CLI> pjsip show endpoints
Endpoints:
101
102
*CLI>
```

### A Little Dialplan

Now that we have our PJSIP endpoints stored in our MySQL database, let's add a little dialplan so that they can call each other. To do this, edit Asterisk's `/etc/asterisk/extensions.conf` file and add the following lines to the end:

```
/etc/asterisk/extensions.conf

[testing]
exten => _1XX,1,NoOp()
same => n,Dial(PJSIP/${EXTEN})
```

Or to dial multiple AOR contacts at the same time, use the `PJSIP_DIAL_CONTACTS` function:

### **`/etc/asterisk/extensions.conf`**

```
[testing]
exten => _1XX,1,NoOp()
same => n,Dial(${PJSIP_DIAL_CONTACTS(${EXTEN}}))
```

### **Conclusion**

Now, start Asterisk back up, or reload it using **core reload** from the Asterisk CLI, register your two SIP phones using the 101/101 and 102/102 credentials, and make a call.

## Exchanging Device and Mailbox State Using PJSIP

### Background

Asterisk has permitted the exchange of device and mailbox state for many versions. This has normally been accomplished using the `res_xmpp` module for instances across networks or using `res_corosync` for instances on the same network. This has required, in some cases, an extreme amount of work to setup. In the case of `res_xmpp` this also adds another point of failure for the exchange in the form of the XMPP server itself. The `res_pjsip_publish_asterisk` module on the other hand does not suffer from this.

### Operation

The `res_pjsip_publish_asterisk` module establishes an optionally bidirectional or unidirectional relationship between Asterisk instances. When the device or mailbox state on one Asterisk changes it is sent to the other Asterisk instance using a PUBLISH message containing an Asterisk specific body. This body is comprised of JSON and contains the information required to reflect the remote state change. For situations where you may not want to expose all states or you may not want to allow all states to be received you can optionally filter using a regular expression. This limits the scope of traffic.

### Configuration

Configuring things to exchange state requires a few different objects: `endpoint`, `publish`, `asterisk-publication`, and optionally `auth`. These all configure a specific part in the exchange. An `endpoint` must be configured as a fundamental part of PJSIP is that **all** incoming requests are associated with an `endpoint`. A `publish` object tells the `res_pjsip_outbound_publish` where to send the PUBLISH and what type of PUBLISH message to send. An `asterisk-publication` object configures handling of PUBLISH messages, including whether they are permitted and from whom. Last you can optionally use authentication so that PUBLISH messages are challenged for credentials.

### Example Configuration

The below configuration is for two Asterisk instances sharing all device and mailbox state between them.

#### Instance #1 (IP Address: 172.16.10.1):

```
[instance2]
type=endpoint

[instance2-devicestate]
type=outbound-publish
server_uri=sip:instance1@172.16.10.2
event=asterisk-devicestate

[instance2-mwi]
type=outbound-publish
server_uri=sip:instance1@172.16.10.2
event=asterisk-mwi

[instance2]
type=inbound-publication
event_asterisk-devicestate=instance2
event_asterisk-mwi=instance2

[instance2]
type=asterisk-publication
devicestate_publish=instance2-devicestate
mailboxstate_publish=instance2-mwi
device_state=yes
mailbox_state=yes
```

This configures the first instance to publish device and mailbox state to 'instance 2' located at 172.16.10.2 using a resource name of 'instance1' without authentication. As no filters exist all state will be published. It also configures the first instance to accept all device and mailbox state messages published to a resource named 'instance2' from 'instance2'.

#### Instance #2 (IP Address: 172.16.10.2):

```

[instance1]
type=endpoint

[instance1-devicestate]
type=outbound-publish
server_uri=sip:instance2@172.16.10.1
event=asterisk-devicestate

[instance1-mwi]
type=outbound-publish
server_uri=sip:instance2@172.16.10.1
event=asterisk-mwi

[instance1]
type=inbound-publication
event_asterisk-devicestate=instance1
event_asterisk-mwi=instance1

[instance1]
type=asterisk-publication
devicestate_publish=instance1-devicestate
mailboxstate_publish=instance1-mwi
device_state=yes
mailbox_state=yes

```

This configures the second instance to publish device and mailbox state to 'instance 1' located at 172.16.10.1 using a resource name of 'instance2' without authentication. As no filters exist all state will be published. It also configures the second instance to accept all device and mailbox state messages published to a resource named 'instance1' from 'instance1'.

### ***Filtering***

As previously mentioned state events can be filtered by the device or mailbox they relate to using a regular expression. This is configured on 'publish' types using '@device\_state\_filter' and '@mailbox\_state\_filter' and on 'asterisk-publication' types using 'device\_state\_filter' and 'mailbox\_state\_filter'. As each event is sent or received the device or mailbox is given to the regular expression and if it does not match the event is stopped.

### ***Example***

```

[instance1]
type=endpoint

[instance1-devicestate]
type=outbound-publish
server_uri=sip:instance2@172.16.10.1
event=asterisk-devicestate

[instance1-mwi]
type=outbound-publish
server_uri=sip:instance2@172.16.10.1
event=asterisk-mwi

[instance1]
type=inbound-publication
event_asterisk-devicestate=instance1
event_asterisk-mwi=instance1

[instance1]
type=asterisk-publication
devicestate_publish=instance1-devicestate
mailboxstate_publish=instance1-mwi
device_state=yes
device_state_filter=^PJSIP/
mailbox_state=yes
mailbox_state_filter=^1000

```

This builds upon the initial configuration for instance #2 but adds filtering of received events. Only device state events relating to PJSIP endpoints will be accepted. As well only mailbox state events for mailboxes starting with 1000 will be accepted.



This configuration is not ideal as the publishing instance (instance #1) will still send state changes for devices and mailboxes that instance #2 does not care about, thus wasting bandwidth.

### ***Fresh Startup***

When the `res_pjsip_publish_asterisk` module is loaded it will send its own current states for all applicable devices and mailboxes to all configured 'publish' types. Instances may optionally be configured to send a refresh request to 'publish' types as well by setting the 'devicestate\_publish' and/or 'mailboxstate\_publish' option in the 'asterisk-publication' type. This refresh request causes the remote instances to send current states for all applicable devices and mailboxes back, bringing the potentially newly started Asterisk up to date with its peers.

## Configuring res\_pjsip for Presence Subscriptions



Under Construction - This page is a stub!

### Capabilities

Asterisk's PJSIP channel driver provides the same presence subscription capabilities as `chan_sip` does. This means that RFC 3856 presence and RFC 4235 dialog info are supported. Presence subscriptions support RFC 3863 PIDF+XML bodies as well as XPIDF+XML. Beyond that, Asterisk also supports subscribing to RFC 4662 lists of presence resources.

### Configuration

If you are familiar with configuring subscriptions in `chan_sip` then this should be familiar to you. Configuration of presence is performed using the "hint" priority for an extension in `extensions.conf`.

#### On this Page

- [Capabilities](#)
- [Configuration](#)
- [Presence Customisations](#)
  - [Digium Presence](#)
  - [Rich Presence \(limited\)](#)

#### extensions.conf

```
[default]
exten => 1000,hint,PJSIP/alice
```

The line shown here is similar to any normal line in a dialplan, except that instead of a priority number or label, the word "hint" is specified. The hint is used to associate the state of individual devices with the state of a dialplan extension. An English translation of the dialplan line would be "Use the state of device PJSIP/alice as the basis for the state of extension 1000". When PJSIP endpoints subscribe to presence, they are subscribing to the state of an extension in the dialplan. By providing the dialplan hint, you are creating the necessary association in order to know which device (or devices) are relevant. For the example given above, this means that if someone subscribes to the state of extension 1000, then they will be told the state of PJSIP/alice. For more information about device state, see [this page](#).

There are two endpoint options that affect presence subscriptions in `pjsip.conf`. The `allow_subscribe` option determines whether SUBSCRIBE requests from the endpoint are permitted to be received by Asterisk. By default, `allow_subscribe` is enabled. The other setting that affects presence subscriptions is the `context` option. This is used to determine the dialplan context in which the extension being subscribed to should be searched for. Given the dialplan snippet above, if the intent of an endpoint that subscribes to extension 1000 is to subscribe to the hint at 1000@default, then the context of the subscribing endpoint would need to be set to "default". Note that if the `context` option is set to something other than "default", then Asterisk will search that context for the hint instead.

In order for presence subscriptions to work properly, some modules need to be loaded. Here is a list of the required modules:

- `res_pjsip.so`: Core of PJSIP code in Asterisk.
- `res_pjsip_pubsub.so`: The code that implements SUBSCRIBE/NOTIFY logic, on which individual event handlers are built.
- `res_pjsip_exten_state.so`: Handles the "presence" and "dialog" events.
- `res_pjsip_pidf_body_generator.so`: This module generates application/pidf+xml message bodies. Required for most subscriptions to the "presence" event.
- `res_pjsip_xpidf_body_generator.so`: This module generates application/xpidf+xml message bodies. Required for some subscriptions to the "presence" event.
- `res_pjsip_dialog_info_body_generator.so`: Required for subscriptions to the "dialog" event. This module generates application/dialog-info message bodies.

If you are unsure of what event or what body type your device uses for presence subscriptions, consult the device manufacturer's manual for more information.

### Presence Customisations

#### Digium Presence

Digium phones are outfitted with a custom supplement to the base PIDF+XML presence format that allows for XMPP-like presence to be understood. To add this, the hint can be modified to include an additional presence state, like so:

#### extensions.conf

```
[default]
exten => 1000,hint,PJSIP/alice,CustomPresence:alice
```

This means that updates to the presence state of CustomPresence:alice will also be conveyed to subscribers to extension 1000. For more information on presence state in Asterisk, see [this page](#).

The `res_pjsip_pidf_digium_body_supplement.so` module must be loaded in order for additional presence details to be reported.

### ***Rich Presence (limited)***

Some rich presence supplements that were in `chan_sip` have been migrated to the PJSIP channel driver as well. This is an extremely limited implementation of the "activities" element of a person. The `res_pjsip_pidf_eyebeam_body_supplement.so` module is required to add this functionality.

## Resource List Subscriptions (RLS)

### Overview

Beginning in Asterisk 13, Asterisk supports RFC 4662 resource list subscriptions in its PJSIP-based SIP implementation.

In a PBX environment, it is common for SIP devices to subscribe to many resources offered by the PBX. This holds especially true for presence resources. Consider a small office where an Asterisk server acts as a PBX for 20 people, each with a SIP desk phone. Each of those 20 phones subscribes to the state of the others in the office. In this case, each of the phones would create 19 subscriptions (since the phone does not subscribe to its own state). When totalled, the Asterisk server would maintain  $20 * 19 = 380$  subscriptions. For an office with 30 people, the total number of subscriptions becomes  $30 * 29 = 870$  subscriptions. For an office with 40 people, the total number of subscriptions becomes  $40 * 39 = 1560$ . That is about four times the number of subscriptions for the 20-person office, despite only having twice the number of people. The number of subscriptions follows a geometric progression, leading to a situation commonly called an *N-squared* problem. In other words, the amount of traffic generated and amount of server resources required are proportional to the square of the number of users (N) on the system. The N-squared problem with subscriptions can be a limiting factor for PBX deployers for several reasons:

- In a situation where all phones boot up simultaneously, each of the phones will be sending out their SIP SUBSCRIBEs nearly simultaneously, placing a larger-than-average burden on the Asterisk server's CPU.
- In the SIP stack, N-squared long-term SIP dialogs have to be maintained, tying up more system resources (e.g. memory).

#### On this Page

- [Overview](#)
- [Configuring Resource List Subscriptions](#)
- [Batching Notifications](#)
- [Corner Cases](#)
  - [Non-existent List Items](#)
  - [Loops](#)
  - [Ambiguity](#)
- [Limitations](#)
  - [List size](#)
  - [Lack of dynamism](#)

These limitations can drastically limit the number of devices a PBX administrator can use with an Asterisk system. Even if the hardware is capable of handling the mean traffic of, say, 200 users, it may be required to limit the number of users to 50 or fewer because of the N-squared subscriptions generating so much simultaneous traffic.

Resource list subscriptions provide relief for this problem by allowing for resources to be grouped into lists. A single subscription to a list will result in multiple back-end subscriptions to the resources in that list. Notifications of state changes can also be batched so that multiple state changes may be conveyed in a single message. This can help to significantly decrease the amount of subscription-related traffic and processing being performed.

### Configuring Resource List Subscriptions

RLS is configured in `pjsip.conf` using a special configuration section type called "resource\_list". Here is an example of a simple resource list:

#### pjsip.conf

```
[sales]
type = resource_list
event = presence
list_item = alice
list_item = bob
list_item = carol
```

It should be simple to glean the intent of this list. We have created a list called "sales" that provides the presence of the sales team of alice, bob, and carol. Let's go over each of the options in more detail.

- `type`: This must be set to "resource\_list" so that the configuration parser knows that it is looking at a resource list.
- `event`: The SIP event package provided by this resource list. Asterisk, as provided by Digium, provides support for the following event packages:
  - `presence`: Provides ability to determine when devices are in use or not. Commonly known as BLF.
  - `dialog`: An alternate method of providing BLF. Used by certain SIP equipment instead of the presence event package.
  - `message-summary`: Provides the ability to determine the number of voice mail messages that a mailbox contains. Commonly known as MWI.
- `list_item`: This is the name of a resource that belongs to the list. The formatting of list items is dependent on the event package provided by the list.
  - `presence`: This is the name of an extension in the dialplan. In the example, the extensions "alice", "bob", and "carol" exist in `extensions.conf`.
  - `dialog`: The same as the presence event package.
  - `message-summary`: This is the name of a mailbox. If you are using `app_voicemail`, then mailboxes will be in the form of "mailbox@context". If you are using an external voicemail system, then the name of the mailbox will be in whatever format the external voicemail system uses for mailbox names.

The list items in the example were placed on separate lines, but it is also valid to place multiple list items on a single line: `list_item =`



alice,bob,carol. Note also that list items can also be other resource lists of the same event type.

There is one further option that is not listed here, but deserves some mention: `full_state`. RFC 4662 defines "full" and "partial" state notifications. When the states of a subset of resources on a resource list changes, a server has the option of sending a notification that only contains the resources whose states have changed. This is a partial state notification. A full state notification would include the states of all resources in the list, even if only some of the resources' states have changed. The `full_state` option allows for full state notifications to be transmitted unconditionally. By default, `full_state` is disabled on resource list subscriptions in order to keep the size of notifications small. It is **highly recommended** that you use the default value for this option unless you are using a client that does not understand partial state notifications.

## Batching Notifications

In addition to the basic options listed above, there is another option, `notification_batch_interval` that can be used to change Asterisk's behavior when sending notifications of resource state changes on a list. By default, whenever the state of any resource on a list changes, Asterisk will immediately send out a notification of the state change. If, however, a `notification_batch_interval` is specified, then when a resource state changes, Asterisk will start a timer for the specified interval. While the timer is running, any further state changes of resources in the list are batched along with the original state change that started the timer. When the timer expires, then all batched state changes are sent in a single NOTIFY.

Let's modify the previous configuration to use a batching interval:

```
pjsip.conf

[sales]
type = resource_list
event = presence
list_item = alice
list_item = bob
list_item = carol
notification_batch_interval = 2000
```

The units for the `notification_batch_interval` are milliseconds. With this configuration, Asterisk will collect resource state changes for 2000 milliseconds before sending notifications on this resource list.

The biggest advantage of notification batching is that it can decrease the number of NOTIFY requests that Asterisk sends. If two SIP phones on a PBX are having a conversation with one another, when a call completes, both phones are likely to change states to being not in use. By having a batching interval configured, it would allow for a single NOTIFY to indicate both devices' state changes instead of having to send two separate NOTIFY requests.

The biggest disadvantage of notification batching is that it becomes possible for transient states for a device to be missed. If you have a batching interval of 3000 milliseconds, and a phone only rings for one second before it is answered, it means that the ringing state of the phone never got transmitted to interested listeners.

## Corner Cases

### Non-existent List Items

Let's say you have the following list configured in `pjsip.conf`:

```
pjsip.conf

[sales]
type = resource_list
event = presence
list_item = alice
list_item = bob
list_item = carol
```

And you have the following in `extensions.conf`

```
pjsip.conf

[default]
exten => alice,hint,PJSIP/alice
exten => bob,hint,PJSIP/bob
```

Notice that there is no "carol" extension in `extensions.conf`. What happens when a user attempts to subscribe to the sales list?

When the subscription arrives, Asterisk recognizes the subscription as being for the list. Asterisk then acts as if it is establishing individual subscriptions to each of the list items the same way it would if a subscription had arrived directly for the list item. In this case, the subscriptions to alice and bob succeed. However, the presence subscription handler complains that it cannot subscribe to carol since the resource does not exist.

The policy currently used is that if subscription to at least one list resource succeeds, then the subscription to the entire list has succeeded. Only the list items that were successfully subscribed to will be reflected in the list subscription. If subscription to **all** list items fails, then the subscription to the list also fails.

## Loops

Let's say you have the following pjsip.conf file:

```
                                pjsip.conf
[sales]
type = resource_list
event = presence
list_item = tech_support

[tech_support]
type = resource_list
event = presence
list_item = sales
```

Notice that the sales list contains the tech\_support list, and the tech\_support list contains the sales list. We have a loop here. How is that handled?

Asterisk's policy with loops is to try to resolve the issue while being as graceful as possible. The way it does this is that when it detects a loop, it essentially considers the looped subscription to be a failed list item subscription. The process would go something like this:

1. A subscription arrives for the sales list.
2. We attempt to subscribe to the tech\_support list item in the sales list.
3. Inside the tech\_support list, we see the sales list as a list item.
4. We notice that we've already visited the sales list, so we fail the subscription to the sales list list item.
5. Since subscriptions to all list items in the tech support list failed, the subscription to the tech support list failed.
6. Since the tech support list was the only list item in the sales list, and that subscription failed, the subscription to the sales list fails as well.

What if the configured lists were modified slightly:

```
                                pjsip.conf
[sales]
type = resource_list
event = presence
list_item = tech_support

[tech_support]
type = resource_list
event = presence
list_item = sales
list_item = alice
```

Notice that the tech\_support list now also has alice as a list\_item. How does the process change on a subscription attempt to sales?

1. A subscription arrives for the sales list
2. We attempt to subscribe to the tech\_support list item in the sales list.
3. Inside the tech\_support list, we see the sales list as a list item.
4. We notice that we've already visited the sales list, so we fail the subscription to the sales list list item.
5. We move on to the next list\_item in tech\_support, alice.
6. We attempt a subscription to alice, and it succeeds.
7. Since at least one subscription to a list item in tech\_support succeeded, the subscription to tech\_support succeeds.
8. Since the subscription to the only list item in sales succeeded, the subscription to sales succeeds.

So in this case, even though the configuration contains a loop, Asterisk is able to successfully create a subscription while trimming the loops out.

## Ambiguity

### Duplicated List Names

If a list name is duplicated, then the configuration framework of Asterisk will not allow for the two to exist as separate entities. It is expected that the most recent list in the configuration file will overwrite the earlier ones.

While this may seem like an obvious thing, users may be tempted to configure lists that have the same name but that exist for different SIP event packages. While this may seem like a legitimate configuration, it will not work as intended.

### List and Resources with Same Name

One flaw that RLS has is that there is no way to know whether a subscription is intended to be for a list or for an individual resource. Let's say you have the following pjsip configuration:

### pjsip.conf

```
[sales]
type = resource_list
event = presence
list_item = alice
list_item = bob
list_item = carol
```

And let's say you have the following `extensions.conf`:

### extensions.conf

```
[default]
exten => sales, hint, Custom:sales
```

What happens if someone attempts to subscribe to the "sales" presence resource?

One easy way to determine intent is to check the Supported: header in the incoming SUBSCRIBE request. If "eventlist" does not appear, then the subscriber does not support RLS and is therefore definitely subscribing to the individual sales resource as described in `extensions.conf`.

But if the subscriber does support RLS, then Asterisk's policy is to always assume that the subscriber intends to subscribe to the list, not the individual resource.

### Conflicting Batching Intervals

`notification_batch_interval` can be configured on any resource list. Consider the following configuration:

### pjsip.conf

```
[sales]
type = resource_list
event = presence
list_item = sales_b
list_item = carol
list_item = david
notification_batch_interval = 3000

[sales_b]
type = resource_list
event = presence
list_item = alice
list_item = bob
notification_batch_interval = 10000
```

What is the batch interval when a user subscribes to the sales list?

The policy that Asterisk enforces is that only the batch interval of the top-most list in the hierarchy is applied. So in the example above, the batch interval would be 3000 milliseconds since the top-most list in the hierarchy is the sales list. If the sales list did not have a batch interval configured, then there would be no batch interval for the list subscription at all.

### Limitations

#### List size

Due to limitations in the PJSIP stack, Asterisk is limited regarding the size of a SIP message that can be transmitted. Asterisk currently works around the built-in size limitation of PJSIP (4000 bytes by default) and can send a message up to 64000 bytes instead. RFC 4662 requires that when sending a NOTIFY request due to an inbound SUBSCRIBE request, we must send the full state of the resource list in response. For large lists, this may mean that the NOTIFY will exceed the size limit.

It is difficult to try to quantify the limit in terms of number of list resources since different body types are more verbose than others, and different configurations will have different variables that will factor into the size of the message (e.g. the length of SIP URIs for one system may be three times as long as the SIP URIs for a separate system, depending on how things are configured).

If you create a very large list, and you find that Asterisk is unable to send NOTIFY requests due to the size of the list, consider breaking the list into smaller sub-lists if possible.

#### Lack of dynamism

Resource lists can be updated as you please, adding and removing list items, altering the batching interval, etc. However, you will find that when a list is altered, any current subscriptions to the list are not updated to reflect the changes to the list. This is because the list is read from configuration at the time that the subscription is established, and the configuration is never again consulted during the lifetime of the subscription. If configuration is updated, then you must terminate your current subscriptions to the list and create a new subscription in order to apply the changes.

Similarly, the state of resources is locked in at the time the subscription is established. For instance, if a list contains a list item that does not exist at the time the subscription is established, if that resource comes into existence later, then the established subscription is not updated to properly reflect the added list item. The subscription must be terminated and re-established in order to have the corrected list item included.

## Configuring Outbound Registrations



This page is under construction. Please refrain from commenting here until this warning is removed.

### Overview

Like with `chan_sip`, Asterisk's PJSIP implementation allows for configuration of outbound registrations. Unlike `chan_sip`, it is not implemented in an obnoxious way. Like with most concepts in PJSIP configuration, outbound registrations are confined to a configuration section of their own.

### Configuration options

A list of outbound registration configuration options can be found on [this page](#). Here is a simple example configuration for an outbound registration to a provider:

#### On this Page

- [Overview](#)
- [Configuration options](#)
- [Outbound registrations and endpoints](#)
- [Authentication](#)
- [Dealing with Failure](#)
  - [Temporary and Permanent Failures](#)
- [CLI and AMI](#)
  - [Monitoring Status](#)
  - [Manually Unregistering](#)
- [Realtime](#)

#### pjsip.conf

```
[my_provider]
type = registration
server_uri = sip:registrar@example.com
client_uri = sip:client@example.com
contact_user = inbound-calls
```

This results in the following outbound REGISTER request being sent by Asterisk:

```
<--- Transmitting SIP request (557 bytes) to UDP:93.184.216.119:5060 --->
REGISTER sip:registrar@example.com SIP/2.0
Via: SIP/2.0/UDP 10.24.20.249:5060;rport;branch=z9hG4bKPjd1a32b43-82ed-4f98-ae24-20149cdf0749
From: <sip:client@example.com>;tag=904e0db9-8297-4bb0-89c5-5cfelcfe654
To: <sip:client@example.com>
Call-ID: 03241f7b-3936-4140-8bad-6840774b78d9
CSeq: 10266 REGISTER
Contact: <sip:inbound-calls@10.24.20.249:5060>
Expires: 3600
Allow: OPTIONS, SUBSCRIBE, NOTIFY, PUBLISH, INVITE, ACK, BYE, CANCEL, UPDATE, PRACK, MESSAGE, REFER, REGISTER
Max-Forwards: 70
Content-Length: 0
```

Let's go over how the options were applied to this REGISTER:

- The `server_uri` is the actual URI where the registrar is located. If you are registering with a SIP provider, they should give this information to you.
- The `client_uri` is used in the To and From headers of the REGISTER. In other words, this is the address of record to which you are binding a contact URI. If registering to a SIP provider, they may require you to provide a specific username in order to identify that the REGISTER is coming from you. Note that the domain of the `client_uri` is the same as the server URI. This is common when indicating that the registrar receiving the REGISTER is responsible for the URI being registered to it.
- The `contact_user` option can be seen in the user portion of the URI in the Contact header. This allows for you to control where incoming calls from the provider will be routed. Calls from the provider will arrive in this extension in the dialplan. Note that this option does not relate to endpoint-related options. For information on relating outbound registrations and endpoints, see the following section.

An English translation of the above REGISTER is "Tell the server at `sip:registrar@example.com` that when SIP traffic arrives addressed to `sip:client@example.com`, the traffic should be sent to `sip:inbound-calls@10.24.20.249`." Note in this example that `10.24.20.249` is the IP address of the Asterisk server that sent the outbound REGISTER request.

### Outbound registrations and endpoints

If you examine the configuration options linked in the previous section, you will notice that there is nothing that ties an outbound registration to an endpoint. The two are considered completely separate from each other, as far as Asterisk is concerned. However, it is likely that if you are registering to an ITSP, you will want to receive incoming calls from that provider. This means that you will need to set up an endpoint that represents this provider. An example of such

an endpoint configuration can be found [here](#), but it is a bit complex. Let's instead make a simpler one just for the sake of explanation. Assuming the previous registration has been configured, we can add the following:

```
pjsip.conf

[my_provider_endpoint]
type = endpoint

[my_provider_identify]
type = identify
match = <ip address of provider>
endpoint = my_provider
```

This represents the bare minimum necessary in order to accept incoming calls from the provider. The `identify` section makes it so that incoming SIP traffic from the IP address in the `match` option will be associated with the endpoint called `my_provider_endpoint`.


If you also wish to make outbound calls to the provider, then you would also need to add an AoR section so that Asterisk can know where to send calls directed to "my\_provider\_endpoint".

```
pjsip.conf

[my_provider_endpoint]
type = endpoint
aors = my_provider_aor

[my_provider_identify]
type = identify
match = <ip address of provider>
endpoint = my_provider

[my_provider_aor]
type = aor
contact = sip:my_provider@example.com
```

 Let me reiterate that this is the **bare minimum**. If you want calls to and from the provider to actually work correctly, you will want to set a context, codecs, authentication, etc. on the endpoint.

## Authentication

It is likely that if you are registering to a provider, you will need to provide authentication credentials. Authentication for outbound registrations is configured much the same as it is for endpoints. The `outbound_auth` option allows for you to point to a `type = auth` section in your configuration to refer to when a registrar challenges Asterisk for authentication. Let's modify our configuration to deal with this:

```
pjsip.conf

[my_provider]
type = registration
server_uri = sip:registrar@example.com
client_uri = sip:client@example.com
contact_user = inbound-calls
outbound_auth = provider_auth

[provider_auth]
type = auth
username = my_username
password = my_password
```

With this configuration, now if the registrar responds to a REGISTER by challenging for authentication, Asterisk will use the authentication credentials in the `provider_auth` section in order to authenticate. Details about what options are available in auth sections can be found [here](#) in the "auth" section.

## Dealing with Failure

### Temporary and Permanent Failures

Whenever Asterisk sends an outbound registration and receives some sort of failure response from the registrar, Asterisk makes a determination about whether a response can be seen as a permanent or temporary failure. The following responses are always seen as temporary failures:

- No Response
- 408 Request Timeout
- 500 Internal Server Error
- 502 Bad Gateway
- 503 Service Unavailable

- 504 Server Timeout
- Any 600-class response

In addition, there is an option called `auth_rejection_permanent` that can be used to determine if authentication-related rejections from a registrar are treated as permanent or temporary failures. By default, this option is enabled, but disabling the setting means the following two responses are also treated as temporary failures:

- 401 Unauthorized
- 407 Proxy Authentication Required

What is meant by temporary and permanent failures? When a temporary failure occurs, Asterisk may re-attempt registering if a `retry_interval` is configured in the outbound registration. The `retry_interval` is the number of seconds Asterisk will wait before attempting to send another REGISTER request to the registrar. By default, outbound registrations have a `retry_interval` of 60 seconds. Another configuration option, `max_retries`, determines how many times Asterisk will attempt to re-attempt registration before permanently giving up. By default, `max_retries` is set to 10.

Permanent failures result in Asterisk immediately ceasing to re-attempt the outbound registration. All responses that were not previously listed as temporary failures are considered to be permanent failures. There is one exception when it comes to permanent failures. The `forbidden_retry_interval` can be set such that if Asterisk receives a 403 Forbidden response from a registrar, Asterisk can wait the number of seconds indicated and re-attempt registration. Retries that are attempted in this manner count towards the same `max_retries` value as temporary failure retries.

Let's modify our outbound registration to set these options to custom values:

```


pjsip.conf


[my_provider]
type = registration
server_uri = sip:registrar@example.com
client_uri = sip:client@example.com
contact_user = inbound-calls
outbound_auth = provider_auth
auth_rejection_permanent = no
retry_interval = 30
forbidden_retry_interval = 300
max_retries = 20

```

In general, this configuration is more lenient than the default. We will retry registration more times, we will retry after authentication requests and forbidden responses, and we will retry more often.

## CLI and AMI

### Monitoring Status

You can monitor the status of your configured outbound registrations via the CLI and the Asterisk Manager Interface. From the CLI, you can issue the command `pjsip show registrations` to list all outbound registrations. Here is an example of what you might see:

```

<Registration/ServerURI.....> <Auth.....> <Status.....>
=====
my_provider/sip:registrar@example.com      provider_auth  Unregistered
outreg/sip:registrar@example.com           n/a           Unregistered

```

On this particular Asterisk instance, there are two outbound registrations configured. The headers at the top explain what is in each column. The "Status" can be one of the following values:

- **Unregistered:** Asterisk is currently not registered. This is most commonly seen when the registration has not yet been established. This can also be seen when the registration has been forcibly unregistered or if the registration times out.
- **Registered:** Asterisk has successfully registered.
- **Rejected:** Asterisk attempted to register but a failure occurred. See the above section for more information on failures that may occur.
- **Stopped:** The outbound registration has been removed from configuration, and Asterisk is attempting to unregister.

In addition, you can see the details of a particular registration by issuing the `pjsip show registration <registration name>` command. If I issue `pjsip show registration my_provider`, I see the following:

```

<Registration/ServerURI.....> <Auth.....> <Status.....>
=====
my_provider/sip:registrar@example.com          provider_auth    Unregistered

ParameterName      : ParameterValue
=====
auth_rejection_permanent : false
client_uri         : sip:client@example.com
contact_user       : inbound-calls
expiration         : 3600
forbidden_retry_interval : 300
max_retries        : 20
outbound_auth      : provider_auth
outbound_proxy     :
retry_interval     : 30
server_uri         : sip:registrar@example.com
support_path       : false
transport          :

```

This provides the same status line as before and also provides the configured values for the outbound registration.

AMI provides the `PJSIPShowRegistrationsOutbound` command that provides the same information as the CLI commands. Here is an example of executing the command in an AMI session:

```

action: PJSIPShowRegistrationsOutbound

Response: Success
EventList: start
Message: Following are Events for each Outbound registration

Event: OutboundRegistrationDetail
ObjectType: registration
ObjectName: my_provider
SupportPath: false
AuthRejectionPermanent: false
ServerUri: sip:registrar@example.com
ClientUri: sip:client@example.com
RetryInterval: 30
MaxRetries: 20
OutboundProxy:
Transport:
ForbiddenRetryInterval: 300
OutboundAuth: provider_auth
ContactUser: inbound-calls
Expiration: 3600
Status: Rejected
NextReg: 0

Event: OutboundRegistrationDetail
ObjectType: registration
ObjectName: outreg
SupportPath: false
AuthRejectionPermanent: true
ServerUri: sip:registrar@example.com
ClientUri: sip:client@example.com
RetryInterval: 60
MaxRetries: 10
OutboundProxy:
Transport:
ForbiddenRetryInterval: 0
OutboundAuth:
ContactUser: inbound-calls
Expiration: 3600
Status: Rejected
NextReg: 0

Event: OutboundRegistrationDetailComplete
EventList: Complete
Registered: 0
NotRegistered: 2

```

The command sends `OutboundRegistrationDetail` events for each configured outbound registration. Most information is the same as the CLI displays, but there is one additional piece of data displayed: `NextReg`. This is the number of seconds until Asterisk will send a new REGISTER request to the registrar. In this particular scenario, that number is 0 because the two outbound registrations have reached their maximum number of retries.

### Manually Unregistering



The AMI and CLI provide ways for you to manually unregister if you want. The CLI provides the `pjsip send unregister <registration name> command`. AMI provides the `PJSIPUnregister` command to do the same thing.



After manually unregistering, the specified outbound registration will continue to reregister based on its last registration expiration.

## Realtime

At the time of this wiki article writing, it is not possible, nor would it be recommended, to use dynamic realtime for outbound registrations. The code in `res_pjsip_outbound_registration.so`, the module that allows outbound registrations to occur, does not attempt to look outside of `pjsip.conf` for details regarding outbound registrations. This is done because outbound registrations are composed both of the configuration values as well as state (e.g. how many retries have we attempted for an outbound registration). When pulling configuration from a file, a reload is necessary, which makes it easy to have a safe place to transfer state information or alter configuration values when told that things have changed. With dynamic realtime, this is much harder to manage since presumably the configuration could change at any point.

If you prefer to use a database to store your configuration, you are free to use static realtime for outbound registrations instead. Like with a configuration file, you will be forced to reload (from the CLI, `module reload res_pjsip_outbound_registration.so`) in order to apply configuration changes.

## Asterisk PJSIP Troubleshooting Guide



This page is currently under construction. Please refrain from commenting until this warning is removed.

### Overview

Are you having problems getting your PJSIP setup working properly? If you are encountering a common problem then hopefully your answer can be found on this page.

Before looking any further here, you should make sure that you have gathered enough information from Asterisk to know what your issue is. It is suggested that you perform the following actions at the Asterisk CLI:

- `core set verbose 4`
- `core set debug 4`
- `pjsip set logger on`

With these options enabled, this will allow you to more easily see what is going on behind the scenes in your failing scenario. It also can help you to cross-reference entries on this page since several debug, warning, and error messages will be quoted here.

### Inbound Calls

#### **Unrecognized Endpoint**

All inbound SIP traffic to Asterisk must be matched to a configured endpoint. If Asterisk is unable to determine which endpoint the SIP request is coming from, then the incoming request will be rejected. If you are seeing messages like:

#### On this Page

- Overview
- Inbound Calls
  - Unrecognized Endpoint
  - Authentication is failing
  - Authentication Not Attempted
  - Asterisk cannot find the specified extension
  - ARGH! NAT!
- Outbound Calls
  - Asterisk says my endpoint does not exist
  - Asterisk cannot route my call
  - ARGH! NAT! (Part 2)
- Bridged Calls
  - Direct media is not being used
- Inbound Registrations
- Outbound Registrations
- Inbound Subscriptions
  - Presence/Dialog Info
  - MWI
- Configuration Issues
  - Can't create an IPv6 transport

```
[2014-10-13 16:12:17.349] DEBUG[27284]: res_pjsip_endpoint_identifier_user.c:106 username_identify: Could not identify endpoint by username 'eggowaffles'
```

or

```
[2014-10-13 16:13:07.201] DEBUG[27507]: res_pjsip_endpoint_identifier_ip.c:113 ip_identify_match_check: Source address 127.0.0.1:5061 does not match identify 'david-ident'
```

then this is a good indication that the request is being rejected because Asterisk cannot determine which endpoint the incoming request is coming from.

How does Asterisk determine which endpoint a request is coming from? Asterisk uses something called "endpoint identifiers" to determine this. There are three endpoint identifiers bundled with Asterisk: user, ip, and anonymous.

#### **Identify by User**

The user endpoint identifier is provided by the `res_pjsip_endpoint_identifier_user.so` module. If nothing has been explicitly configured with regards to endpoint identification, this endpoint identifier is the one being used. The way it works is to use the user portion of the From header from the incoming SIP request to determine which endpoint the request comes from. Here is an example INVITE:

```

<--- Received SIP request (541 bytes) from UDP:127.0.0.1:5061 --->
INVITE sip:service@127.0.0.1:5060 SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1:5061;branch=z9hG4bK-27600-1-0
From: breakfast <sip:eggowaffles@127.0.0.1:5061>;tag=27600SIPpTag001
To: sut <sip:service@127.0.0.1>
Call-ID: 1-27600@127.0.0.1
CSeq: 1 INVITE
Contact: sip:eggowaffles@127.0.0.1:5061
Max-Forwards: 70
Content-Type: application/sdp
Content-Length: 163

v=0
o=user1 53655765 2353687637 IN IP4 127.0.0.1
s=-
c=IN IP4 127.0.0.1
t=0 0
m=audio 6000 RTP/AVP 0
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=ptime:20

```

In this example, the URI in the From header is "sip:eggowaffles@127.0.0.1:5061". The user portion is "eggowaffles", so Asterisk attempts to look up an endpoint called "eggowaffles" in its configuration. If such an endpoint is not configured, then the INVITE is rejected by Asterisk. The most common cause of the problem is that the user name referenced in the From header is not the name of a configured endpoint in Asterisk.

But what if you have configured an endpoint called "eggowaffles"? It is possible that there was an error in your configuration, such as an option name that Asterisk does not recognize. If this is the case, then the endpoint may not have been loaded at all. Here are some troubleshooting steps to see if this might be the case:

- From the CLI, issue the "pjsip show endpoints" command. If the endpoint in question does not show up, then there likely was a problem attempting to load the endpoint on startup.
- Go through the logs from Asterisk startup. You may find that there was an error reported that got lost in the rest of the startup messages. For instance, be on the lookout for messages like:

```

[2014-10-13 16:25:01.674] ERROR[27771]: config_options.c:710 aco_process_var: Could not find option suitable for category
'eggowaffles' named 'setvar' at line 390 of
[2014-10-13 16:25:01.674] ERROR[27771]: res_sorcery_config.c:275 sorcery_config_internal_load: Could not create an object
of type 'endpoint' with id 'eggowaffles' from configuration file 'pjsip.conf'

```

In this case, I set an endpoint option called "setvar" instead of the appropriate "set\_var". The result was that the endpoint was not loaded.

- If you do not see such error messages in the logs, but you do not see the endpoint listed in "pjsip show endpoints", it may be that you forgot to put `type = endpoint` in your endpoint section. In this case, the entire section would be ignored since Asterisk did not know that this was an endpoint section.

### Identify by IP address

Asterisk can also recognize endpoints based on the source IP address of the SIP request. This requires setting up a `type = identify` section in your configuration to match IP addresses or networks to a specific endpoint. Here are some troubleshooting steps:

- Ensure that `res_pjsip_endpoint_identifier_ip.so` is loaded and running. From the CLI, run `module show` like `res_pjsip_endpoint_identifier_ip.so`. The output should look like the following:

Module	Description	Use Count	Status	Support Level
res_pjsip_endpoint_identifier_ip.so	PJSIP IP endpoint identifier	0	Running	core

- Run the troubleshooting steps from the Identify by User section to ensure that the endpoint you have configured has actually been properly loaded.
- From the Asterisk CLI, run the command `pjsip show endpoint <endpoint name>`. Below the headers at the top of the output, you should see something like the following:

```

Endpoint: david/6001                               Unavailable 0 of inf
  InAuth: david-auth/david
  Aor: david                                       10
  Transport: main-transport                       udp        0          0 0.0.0.0:5060
  Identify: 10.24.16.36/32

```

Notice the bottom line. This states that the endpoint is matched based on the IP address 10.24.16.36. If you do not see such a line for the endpoint that you expect to be matched, then there is likely a configuration error. If the line does appear, then ensure that the IP address listed matches what you expect for the endpoint.

- If you are noticing that Asterisk is matching the incorrect endpoint by IP address, ensure that there are no conflicts in your configuration. Run the `pjsip show endpoints` command and look for issues such as the following:

```

Endpoint: carol/6000                               Unavailable 0 of inf
  InAuth: carol-auth/carol
  Aor: carol                                       10
  Transport: main-transport                       udp        0        0 0.0.0.0:5060
  Identify: 10.0.0.0/8

Endpoint: david/6001                               Unavailable 0 of inf
  InAuth: david-auth/david
  Aor: david                                       10
  Transport: main-transport                       udp        0        0 0.0.0.0:5060
  Identify: 10.24.16.36/32

```

Notice that if a SIP request arrives from 10.24.16.36, it is ambiguous if the request should be matched to carol or david.

If you run `pjsip show endpoint <endpoint name>` and do not see an "Identify" line listed, then there is likely a configuration issue somewhere. Here are some common pitfalls

- Ensure that your identify section has `type = identify` in it. Without this, Asterisk will completely ignore the configuration section.
- Ensure that your identify section has an `endpoint` option set in it and that the endpoint is spelled correctly.
- Double-check your match lines for common errors:
  - You cannot use FQDNs or hostnames. You must use IP addresses.
  - Ensure that you do not have an invalid netmask (e.g. 10.9.3.4/255.255.255.300, 127.0.0.1/33).
  - Ensure that you have not mixed up /0 and /32 when using CIDR notation.
- If you are using a configuration method other than a config file, ensure that `sorcery.conf` is configured correctly. Since identify sections are not provided by the base `res_pjsip.so` module, you must ensure that the configuration resides in the `res_pjsip_endpoint_identifier_ip` section of `sorcery.conf`. For example, if you are using dynamic realtime, you might have the following configuration:

```

                                sorcery.conf
[res_pjsip_endpoint_identifier_ip]
identify = realtime,ps_endpoint_id_ips

```

And then you would need the corresponding config in `extconfig.conf`:

```

                                extconfig.conf
[settings]
ps_endpoint_id_ips => odbc

```

### Anonymous Identification

Anonymous endpoint identification allows for a specially-named endpoint called "anonymous" to be matched if other endpoint identifiers are not able to determine which endpoint a request originates from. The `res_pjsip_endpoint_identifier_anonymous.so` module is responsible for matching the incoming request to the anonymous endpoint. If SIP traffic that you expect to be matched to the anonymous endpoint is being rejected, try the following troubleshooting steps:

- Ensure that `res_pjsip_endpoint_identifier_anonymous.so` is loaded and running. From the Asterisk CLI, run `module show like res_pjsip_endpoint_identifier_anonymous.so`. The output should look like the following:

```

Module                Description                Use Count  Status  Support Level
res_pjsip_endpoint_identifier_anonymous.so  PJSIP Anonymous endpoint identifier  0        Running  core

```

- Ensure that the "anonymous" endpoint has been properly loaded. See the troubleshooting steps in the Identify by User section for more details about how to determine if an endpoint has been loaded.

### Authentication is failing

The first thing you should check if you believe that authentication is failing is to ensure that this is the actual problem. Consider the following SIP call from endpoint 200 to Asterisk:

```

<--- Received SIP request (1053 bytes) from UDP:10.24.16.37:5060 --->
INVITE sip:201@10.24.20.249 SIP/2.0
Via: SIP/2.0/UDP 10.24.16.37:5060;rport;branch=z9hG4bKPjQevrxvXqk9Lk5xSW.pzQQb8SAWnJ5Ll1
Max-Forwards: 70
From: "200" <sip:200@10.24.20.249>;tag=DTD-tYEwFMmBPyu0YWalLQdbEUGSLGN5
To: <sip:201@10.24.20.249>
Contact: "200" <sip:200@10.24.16.37:5060;ob>
Call-ID: q.TF2SAaX3jn8dtaLTOCuIO8FRyDCsSR
CSeq: 9775 INVITE
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
User-Agent: Digium D40 1_4_0_0_57389
Content-Type: application/sdp

```

```

Content-Length: 430

v=0
o=- 108683760 108683760 IN IP4 10.24.16.37
s=digphn
c=IN IP4 10.24.16.37
t=0 0
a=X-nat:0
m=audio 4046 RTP/AVP 0 8 9 111 18 58 118 58 96
a=rtcp:4047 IN IP4 10.24.16.37
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:9 G722/8000
a=rtpmap:111 G726-32/8000
a=rtpmap:18 G729/8000
a=rtpmap:58 L16/16000
a=rtpmap:118 L16/8000
a=rtpmap:58 L16-256/16000
a=sendrecv
a=rtpmap:96 telephone-event/8000
a=fmtp:96 0-15

<--- Transmitting SIP response (543 bytes) to UDP:10.24.16.37:5060 --->
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 10.24.16.37:5060;rport;received=10.24.16.37;branch=z9hG4bKpJQevrxvXqk9Lk5xSW.pzQQb8SAWnJ5L11
Call-ID: q.TF2SAaX3jn8dtaLTOCuIO8FRyDCsSR
From: "200" <sip:200@10.24.20.249>;tag=DTD-tYewFMmbPyy0YWallQdbEUGSLGN5
To: <sip:201@10.24.20.249>;tag=z9hG4bKpJQevrxvXqk9Lk5xSW.pzQQb8SAWnJ5L11
CSeq: 9775 INVITE
WWW-Authenticate: Digest
realm="asterisk",nonce="1413305427/8dd1b7f56aba97da45754f7052d8a688",opaque="3b9c806b61adf911",algorithm=md5,qop="auth"
Content-Length: 0

<--- Received SIP request (370 bytes) from UDP:10.24.16.37:5060 --->
ACK sip:201@10.24.20.249 SIP/2.0
Via: SIP/2.0/UDP 10.24.16.37:5060;rport;branch=z9hG4bKpJQevrxvXqk9Lk5xSW.pzQQb8SAWnJ5L11
Max-Forwards: 70
From: "200" <sip:200@10.24.20.249>;tag=DTD-tYewFMmbPyy0YWallQdbEUGSLGN5
To: <sip:201@10.24.20.249>;tag=z9hG4bKpJQevrxvXqk9Lk5xSW.pzQQb8SAWnJ5L11
Call-ID: q.TF2SAaX3jn8dtaLTOCuIO8FRyDCsSR
CSeq: 9775 ACK
Content-Length: 0

<--- Received SIP request (1343 bytes) from UDP:10.24.16.37:5060 --->
INVITE sip:201@10.24.20.249 SIP/2.0
Via: SIP/2.0/UDP 10.24.16.37:5060;rport;branch=z9hG4bKpJCrZnx79augJptGcTbYlXEs2slZNtwYeC
Max-Forwards: 70
From: "200" <sip:200@10.24.20.249>;tag=DTD-tYewFMmbPyy0YWallQdbEUGSLGN5
To: <sip:201@10.24.20.249>
Contact: "200" <sip:200@10.24.16.37:5060;ob>
Call-ID: q.TF2SAaX3jn8dtaLTOCuIO8FRyDCsSR
CSeq: 9776 INVITE
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
User-Agent: Digium D40 1_4_0_0_57389
Authorization: Digest username="200", realm="asterisk", nonce="1413305427/8dd1b7f56aba97da45754f7052d8a688",
uri="sip:201@10.24.20.249", response="2da759314909af8507a59cd1b6bc0baa", algorithm=md5,
cnonce="me-qsYc.rGU-I5A6n-Dy8IhCBg9wKe8", opaque="3b9c806b61adf911", qop=auth, nc=00000001
Content-Type: application/sdp
Content-Length: 430

v=0
o=- 108683760 108683760 IN IP4 10.24.16.37
s=digphn
c=IN IP4 10.24.16.37
t=0 0
a=X-nat:0
m=audio 4046 RTP/AVP 0 8 9 111 18 58 118 58 96
a=rtcp:4047 IN IP4 10.24.16.37
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:9 G722/8000
a=rtpmap:111 G726-32/8000
a=rtpmap:18 G729/8000
a=rtpmap:58 L16/16000
a=rtpmap:118 L16/8000
a=rtpmap:58 L16-256/16000
a=sendrecv
a=rtpmap:96 telephone-event/8000
a=fmtp:96 0-15

<--- Transmitting SIP response (543 bytes) to UDP:10.24.16.37:5060 --->
SIP/2.0 401 Unauthorized

```

Via: SIP/2.0/UDP 10.24.16.37:5060;rport;received=10.24.16.37;branch=z9hG4bKpjCrZnx79augJPtGcTbYlXEs2s1ZNtwYeC  
Call-ID: q.TF2SAaX3jn8dtaLTOCuIO8FRyDCsSR  
From: "200" <sip:200@10.24.20.249>;tag=DTD-tYEwFMmbPyu0YWaLLQdbEUGSLGN5  
To: <sip:201@10.24.20.249>;tag=z9hG4bKpjCrZnx79augJPtGcTbYlXEs2s1ZNtwYeC  
CSeq: 9776 INVITE  
WWW-Authenticate: Digest  
realm="asterisk",nonce="1413305427/8dd1b7f56aba97da45754f7052d8a688",opaque="0b5a53ab6484480a",algorithm=md5,qop="auth"  
Content-Length: 0

<--- Received SIP request (370 bytes) from UDP:10.24.16.37:5060 --->  
ACK sip:201@10.24.20.249 SIP/2.0  
Via: SIP/2.0/UDP 10.24.16.37:5060;rport;branch=z9hG4bKpjCrZnx79augJPtGcTbYlXEs2s1ZNtwYeC  
Max-Forwards: 70  
From: "200" <sip:200@10.24.20.249>;tag=DTD-tYEwFMmbPyu0YWaLLQdbEUGSLGN5  
To: <sip:201@10.24.20.249>;tag=z9hG4bKpjCrZnx79augJPtGcTbYlXEs2s1ZNtwYeC

```
Call-ID: q.TF2SAAx3jn8dtaLTOCuI08FRyDCsSR
CSeq: 9776 ACK
Content-Length: 0
```

At first glance, it would appear that the incoming call was challenged for authentication, and that 200 then failed to authenticate on the second INVITE sent. The actual problem here is that the endpoint 200 does not exist within Asterisk. Whenever a SIP request arrives and Asterisk cannot match the request to a configured endpoint, Asterisk will respond to the request with a 401 Unauthorized response. The response will contain a WWW-Authenticate header to make it look as though Asterisk is requesting authentication. Since no endpoint was actually matched, the authentication challenge created by Asterisk is just dummy information and is actually impossible to authenticate against.

The reason this is done is to prevent an information leak. Consider an attacker that sends SIP INVITEs to an Asterisk box, each from a different user. If the attacker happens to send a SIP INVITE from a user name that matches an actual endpoint on the system, then Asterisk will respond to that INVITE with an authentication challenge using that endpoint's authentication credentials. But what happens if the attacker sends a SIP INVITE from a user name that does not match an endpoint on the system? If Asterisk responds differently, then Asterisk has leaked information by responding differently. If Asterisk sends a response that looks the same, though, then the attacker is unable to easily determine what user names are valid for the Asterisk system.

So if you are seeing what appears to be authentication problems, the first thing you should do is to read the Unrecognized Endpoint section above and ensure that the endpoint you think the SIP request is coming from is actually configured properly. If it turns out that the endpoint is configured properly, here are some trouble-shooting steps to ensure that authentication is working as intended:

- Ensure that username and password in the `type = auth` section are spelled correctly and that they are **using the correct case**. If you have "Alice" as the username on your phone and "alice" as the username in Asterisk, things will go poorly.
- If you are using the `md5_cred` option in an auth section, ensure the following:
  - Ensure that you have set `auth_type = md5`.
  - Ensure that the calculated MD5 sum is composed of `username:realm:password`
  - Ensure that the calculated MD5 sum did not contain any extraneous whitespace, such as a newline character at the end.
  - Ensure there were no copy-paste errors. An MD5 sum is exactly 32 hexadecimal characters. If the option in your config file contains fewer or greater than 32 characters, or if any of the characters are not hexadecimal characters, then the MD5 sum is invalid.
- Ensure that you have specified a `username`. Asterisk does not imply a username based on the name of the auth section.
- Ensure that the configured `realm` is acceptable. In most cases, simple SIP devices like phones will authenticate to whatever realm is presented to them, so you do not need to configure one explicitly. However, if a specific realm is required, be sure it is configured. Be sure that if you are using the `md5_cred` option that this realm name is used in the calculation of the MD5 sum.
- Ensure that the endpoint that is communicating with Asterisk uses the "Digest" method of authentication and the "md5" algorithm. If they use something else, then Asterisk will not understand and reject the authentication attempt.

### Authentication Not Attempted

The opposite problem of authentication failures is that incoming calls are not being challenged for authentication where it would be expected. Asterisk chooses to challenge for authentication if the endpoint from which the request arrives has a configured `auth` option on it. From the CLI, run the `pjsip show endpoint <endpoint name>` command. Below the initial headers should be something like the following:

```
Endpoint: david/6001                               Unavailable 0 of inf
  InAuth: david-auth/david
    Aor: david                                       10
  Transport: main-transport                        udp        0          0 0.0.0.0:5060
  Identify: 10.24.16.36/32
```

Notice the "InAuth" on the second line of output. This shows that the endpoint's auth is pointing to a configuration section called "david-auth" and that the auth section has a username of "david". If you do not see an "InAuth" specified for the endpoint, then this means that Asterisk does not see that the endpoint is configured for authentication. Check the following:

- Ensure that there is an `auth` line in your endpoint's configuration.
- Ensure that the auth that your endpoint is pointing to actually exists. Spelling is important.
- Ensure that the auth that your endpoint is pointing to has `type = auth` specified in it.

### Asterisk cannot find the specified extension

If you are seeing a message like the following on your CLI when you place an incoming call:

```
[2014-10-14 13:22:45.886] NOTICE[1583]: res_pjsip_session.c:1538 new_invite: Call from '201' (UDP:10.24.18.87:5060) to extension '456789' rejected because extension not found in context 'default'.
```

then it means that Asterisk was not able to direct the incoming call to an appropriate extension in the dialplan. In the case above, I dialled "456789" on the phone that corresponds with endpoint 201. Endpoint 201 is configured with `context = default` and the "default" context in my dialplan does not have an extension "456789".

The NOTICE message can be helpful in this case, since it tells what endpoint the call is from, what extension it is looking for, and in what context it is searching. Here are some helpful tips to be sure that calls are being directed where you expect:

- Be sure that the endpoint has the expected context configured. Be sure to check spelling.
- Be sure that the extension being dialled exists in the dialplan. From the Asterisk CLI, run `dialplan show <context name>` to see the extensions for a particular context. If the extension you are dialing is not listed, then Asterisk does not know about the extension.

- Ensure that if you have modified `extensions.conf` recently that you have saved your changes and issued a `dialplan reload` from the Asterisk CLI.
- Ensure that the extension being dialled has a 1 priority.
- Ensure the the extension being dialled is in the expected dialplan context.

## ARGH! NAT!

NAT is objectively terrible. Before having a look at this section, have a look at [this page](#) to be sure that you understand the options available to help combat the problems NAT can cause.

NAT can adversely affect all areas of SIP calls, but we'll focus for now on how they can negatively affect the ability to allow for incoming calls to be set up. The most common issues are the following:

- Asterisk routes responses to incoming SIP requests to the wrong location.
- Asterisk gives the far end an unroutable private address to send SIP traffic to during the call.

### Asterisk sends traffic to unroutable address

The endpoint option that controls how Asterisk routes responses is `force_rport`. By default, this option is enabled and causes Asterisk to send responses to the address and port from which the request was received. This default behavior works well when Asterisk is on a different side of a NAT from the device that is calling in. Since Asterisk presumably cannot route responses to the device itself, Asterisk instead routes the response back to where it received the request from.

### Asterisk gives unroutable address to device

By default, Asterisk will place its own IP address into Contact headers when responding to SIP requests. This can be a problem if the Asterisk server is not routable from the remote device. The `local_net`, `external_signaling_address`, and `external_signaling_port` transport options can assist in preventing this. By setting these options, Asterisk can detect an address as being a "local" address and replace them with "external" addresses instead.

## Outbound Calls

### Asterisk says my endpoint does not exist

If you see a message like the following:

```
[2014-10-14 15:50:50.407] ERROR[2004]: chan_pjsip.c:1767 request: Unable to create PJSIP channel - endpoint 'hammerhead' was not found
```

then this means that Asterisk thinks the endpoint you are trying to dial does not exist. For troubleshooting tips about how to ensure that endpoints are loaded as expected, check the Identify by User subsection in the Incoming Calls section.

Alternatively, if you see a message like the following:

```
[2014-10-14 15:55:06.292] ERROR[2578][C-00000000]: netsock2.c:303 ast_sockaddr_resolve: getaddrinfo("hammerhead", "(null)", ...): Name or service not known
[2014-10-14 15:55:06.292] WARNING[2578][C-00000000]: chan_sip.c:6116 create_addr: No such host: hammerhead
[2014-10-14 15:55:06.292] DEBUG[2578][C-00000000]: chan_sip.c:29587 sip_request_call: Cant create SIP call - target device not registered
```

or

```
[2014-10-14 15:55:58.440] WARNING[2700][C-00000000]: channel.c:5946 ast_request: No channel type registered for 'SIP'
[2014-10-14 15:55:58.440] WARNING[2700][C-00000000]: app_dial.c:2431 dial_exec_full: Unable to create channel of type 'SIP' (cause 66 - Channel not implemented)
```

then it means that your dialplan is referencing "SIP/hammerhead" instead of "PJSIP/hammerhead". Change your dialplan to refer to the correct channel driver, and don't forget to `dialplan reload` when you are finished.

### Asterisk cannot route my call

If Asterisk is finding your endpoint successfully, it may be that Asterisk has no address information when trying to dial the endpoint. You may see a message like the following:

```
[2014-10-14 15:58:06.690] WARNING[2743]: res_pjsip/location.c:155 ast_sip_location_retrieve_contact_from_aor_list: Unable to determine contacts from empty aor list
[2014-10-14 15:58:06.690] WARNING[2834][C-00000000]: app_dial.c:2431 dial_exec_full: Unable to create channel of type 'PJSIP' (cause 3 - No route to destination)
```

If you see this, then the endpoint you are dialling either has no associated address of record (AoR) or the associated AoR does not have any contact URIs bound to it. AoRs are necessary in order to determine the appropriate destination of the call. To see if your endpoint has an associated AoR, run `pjsip show endpoint <endpoint name>` from the Asterisk CLI. The following is sample output of an endpoint that **does** have an AoR configured on it:



```
Endpoint: david/6001                               Unavailable 0 of inf
  InAuth: david-auth/david
  Aor: david                                       10
Transport: main-transport      udp      0      0 0.0.0.0:5060
Identify: 10.24.16.36/32
```

Notice the third line. The endpoint points to the AoR section called "david". If your endpoint does not have an AoR associated with it, this third line will be absent.

If you think you have associated your endpoint with an AoR, but one does not appear in the CLI, then here are some troubleshooting steps:

- Ensure that you have set the `aors` option on the endpoint. Notice that the option is not `aor` (there is an 's' at the end).
- Ensure that the AoR pointed to by the `aors` option exists. Check your spelling!

If those appear to be okay, it may be that there was an error when attempting to load the AoR from configuration. From the Asterisk CLI, run the command `pjsip show aor <aor name>`. If you see a message like

```
Unable to find object heman.
```

Then it means the AoR did not get loaded properly. Here are some troubleshooting steps to ensure your AoR is configured correctly:

- Ensure that your AoR has `type = aor` set on it.
- Ensure that there were no nonexistent configuration options set. You can check the logs at Asterisk startup to see if there were any options Asterisk did not understand. For instance, you may see something like:

```
[2014-10-14 16:16:20.658] ERROR[2939]: config_options.c:710 aco_process_var: Could not find option suitable for category
'1000' named 'awesomeness' at line 219 of
[2014-10-14 16:16:20.659] ERROR[2939]: res_sorcery_config.c:275 sorcery_config_internal_load: Could not create an object
of type 'aor' with id '1000' from configuration file 'pjsip.conf'
```

In this case, I tried to set an option called "awesomeness" on the AoR 1000. Since Asterisk did not recognize this option, AoR 1000 was unable to be loaded.

- The `contact` option can be a pitfall. There is an object type called "contact" that is documented on the wiki, which may make you think that the AoR option should point to the name of a contact object that you have configured. On the contrary, the `contact` option for an AoR is meant to be a SIP URI. The resulting contact object will be created by Asterisk based on the provided URI. Make sure when setting the `contact` that you use a full SIP URI and not just an IP address.

Another issue you may encounter is that you have properly configured an AoR on the endpoint but that this particular AoR has no contact URIs bound to it. From the CLI, run the `pjsip show aor <aor name>` command to see details about the AoR. Here is an example of an AoR that has a contact URI bound to it.

```
Aor: 201                                           1
  Contact: 201/sip:201@10.24.18.87:5060;ob         Unknown      nan
```

The "Contact:" line shows the URI "sip:201@10.24.18.87:5060;ob" is bound to the AoR 201. If the AoR does not have any contacts bound to it, then no Contact lines would appear. The absence of Contact lines can be explained by any of the following:

- If the device is expected to register, then it may be that the device is either not properly configured or that there was a registration failure. See the Inbound Registrations section for details on how to resolve that problem.
- If the device is not intended to register, then the AoR needs to have a `contact` option set on it. See the previous bulleted list for possible `contact`-related pitfalls.

## ARGH! NAT! (Part 2)

NAT makes babies cry.

For outbound calls, the main NAT issue you are likely to come across is Asterisk publishing an unroutable private address in its Contact header. If this is an issue you are facing, this can be corrected by setting the `local_net`, `external_signaling_address`, and `external_signaling_port` options for the transport you are using when communicating with the endpoint. For more information on how this can be set up, please see [this page](#).

## Bridged Calls

### Direct media is not being used

Direct media is a feature that allows for media to bypass Asterisk and flow directly between two endpoints. This can save resources on the Asterisk system and allow for more simultaneous calls. The following conditions are required for direct media. If any are not met, then direct media is not possible:

- There must only be two endpoints involved in the call.
- Both endpoints involved in the call must have the `direct_media` option enabled.
- The call must be a regular person-to-person call. Calls through `ConfBridge()` and `Meetme()` cannot use direct media.
- The sets of codecs in use by each endpoint during the call must have a non-empty intersection. In other words, each endpoint must be using at least one codec that the other endpoint is using.

- Any features in Asterisk that manipulate, record, or inject media may not be used. This includes:
  - The Monitor() and Mixmonitor() applications
  - The Chanspy() application
  - The JACK() application
  - The VOLUME() function
  - The TALK\_DETECT() function
  - The SPEEX() function
  - The PERIODIC\_HOOK() function
  - The 'L' option to the Dial() application
  - An ARI snoop
  - A jitter buffer
  - A FAX gateway
- No features that require that Asterisk intercept DTMF may be used. This includes the T, t, K, k, W, w, X, and x options to the Dial() application.
- If either endpoint has the `disable_direct_media_on_nat` option set, and a possible media NAT is detected, then direct media will not be used. This option is disabled by default, so you would have to explicitly set this option for this to be a problem.
- The two endpoints must be in the same bridge with each other. If the two endpoints are in separate bridges, and those two bridges are connected with one or more local channels, then direct media is not possible.

Double-check that all requirements are met. Unfortunately, Asterisk does not provide much in the way of debug for determining why it has chosen **not** to use direct media.

## Inbound Registrations

For inbound registrations, a lot of the same problems that can happen on inbound calls may occur. Asterisk goes through the same endpoint identification and authentication process as for incoming calls, so if your registrations are failing for those reasons, consult the troubleshooting guide for incoming calls to determine what the problem may be.

If your problem is not solved by looking in those sections, then you may have a problem that relates directly to the act of registering. Before continuing, here is a sample REGISTER request sent to an Asterisk server:

```
REGISTER sip:10.24.20.249:5060 SIP/2.0
Via: SIP/2.0/UDP 10.24.16.37:5060;rport;branch=z9hG4bKPj.rPtUH-P33vMFD68cLZjQj0Qxdu6mNx
Max-Forwards: 70
From: "200" <sip:200@10.24.20.249>;tag=Bxs-nct8-X0e7Q7tspK3Vl3iqUa0cmzcc
To: "200" <sip:200@10.24.20.249>
Call-ID: C0yYQJ8h776wbheBiUEqCin.ZhcBB.tZ
CSeq: 5200 REGISTER
User-Agent: Digium D40 1_4_0_0_57389
Contact: "200" <sip:200@10.24.16.37:5060;ob>
Expires: 300
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Content-Length: 0
```

This REGISTER was sent by the endpoint 200. The URI in the To header is "sip:200@10.24.20.249". Asterisk extracts the username portion of this URI to determine the address of record (AoR) that the REGISTER pertains to. In this case, the AoR has the same name as the endpoint, 200. The URI in the Contact header is "sip:200@10.24.16.37:5060;ob". The REGISTER request is attempting to bind this contact URI to the AoR. Ultimately, what this means is that when someone requests to reach endpoint 200, Asterisk will check the AoRs associated with the endpoint, and send requests to all contact URIs that have been bound to the AoR. In other words, the REGISTER gives Asterisk the means to locate the endpoint.

You can ensure that your configuration is sane by running the `pjsip show endpoint <endpoint name>` CLI command. Part of the output is to show all AoRs associated with a particular endpoint, as well as contact URIs that have been bound to those AoRs. Here is sample output from running `pjsip show endpoint 200` on a system where registration has succeeded:

```
Endpoint: 200/200                                Not in use    0 of inf
Aor: 200                                         1
Contact: 200/sip:200@10.24.16.37:5060;ob        Unknown      nan
Transport: main-transport                        udp          0            0 0.0.0.0:5060
```

This shows that endpoint 200 has AoR 200 associated with it. And you can also see that the contact "sip:200@10.24.16.37:5060;ob" has been bound to the AoR.

If running this command shows no AoR, then ensure that the endpoint has the `aors` option set. Note that the name is `aors`, not `aor`.

More likely, the issue will be that an AoR will be listed, but there will be no associated contact. If this is the case, here are some possible troubleshooting steps:

- Ensure that the AoR has actually been loaded. Run the CLI command `pjsip show aor <aor name>`. If no AoR is displayed, then that means the AoR was not loaded.
  - Ensure that the configuration section has `type = aor` specified.
  - Ensure that all configuration options specified on the AoR are options that Asterisk recognizes.
- Ensure that the `res_pjsip_registrar.so` module is loaded and running. Running `module show` like `res_pjsip_registrar.so` should show the following:

Module	Description	Use Count	Status	Support Level
res_pjsip_registrar.so	PJSIP Registrar Support	0	Running	core

- Ensure that the AoR has a `max_contacts` value configured on it. If this option is not set, then registration cannot succeed. You will see this message on the CLI:

```
[2014-10-16 11:34:07.887] WARNING[2940]: res_pjsip_registrar.c:685 registrar_on_rx_request: AOR '200' has no configured max_contacts. Endpoint '200' unable to register
```

Asterisk will transmit a 403 Forbidden in response to the registration attempt.

If you initially have successful registrations but they later start failing, then here are some further troubleshooting steps for you to try:

- If you intend for new registrations to replace old ones, then enable the `remove_existing` option for the AoR.
- Ensure that if you are attempting to bind multiple contacts to an AoR that the `max_contacts` for the AoR is large enough. If the `max_contacts` value is not high enough, you will see the following CLI message:

```
[2014-10-16 11:34:07.887] WARNING[2940]: res_pjsip_registrar.c:411 rx_task: Registration attempt from endpoint '200' to AOR '200' will exceed max contacts of 1
```

Asterisk will respond to the registration attempt with a 403 Forbidden.

## Outbound Registrations

If you are having troubles with outbound registrations and unfamiliar with the mechanics involved, please see [this page](#). It will explain quite a few of the concepts that Asterisk uses and may give you some clues for solving your issue.

If you are still having trouble, here are some troubleshooting steps:

- If Asterisk is not sending an outbound REGISTER at all, then it is likely that there was an error when trying to load the outbound registration.
  - Ensure that the outbound registration has `type = registration` in it.
  - Ensure that there are no configuration options that Asterisk does not recognize.
- Another reason Asterisk may not be sending an outbound REGISTER is that you do not have a valid SIP URI for your `server_uri` or `client_uri`. You may see a message like this on the CLI if this is the case:

```
[2014-10-16 12:05:16.064] ERROR[3187]: res_pjsip_outbound_registration.c:724 sip_outbound_registration_regc_alloc: Invalid server URI 'registrar@example.com' specified on outbound registration 'outreg'
```

In this case, I left off the initial "sip:" from the URI.

- If your outbound REGISTER receives no response, then you may have misconfigured the `server_uri` to point somewhere the REGISTER is not meant to be sent.
- If Asterisk has stopped sending REGISTER requests, then either the maximum number of retries has been attempted or the response that Asterisk received from the registrar was considered to be a permanent failure. If you want to get Asterisk to start sending REGISTER requests again after making configuration adjustments, you can do so by running the `module reload res_pjsip_registrar.so` CLI command.

## Inbound Subscriptions

The first thing to acknowledge with inbound subscriptions is that the handling of the inbound SUBSCRIBE messages starts the same as for inbound calls. This means that if you are having troubles where Asterisk does not recognize the endpoint sending the SUBSCRIBE or if authentication is failing, you should check the troubleshooting guide for incoming calls for details on how to solve these issues.

It is also important to ensure that `res_pjsip_pubsub.so` is loaded and running. This module is the core of all of Asterisk's handling of subscriptions, and if it is not loaded, then Asterisk will not be able to set up subscriptions properly.

## Presence/Dialog Info

A tutorial about subscribing to presence and dialog-info can be found on [this page](#). Reading that page may point you towards how to resolve the issue you are facing.

If you are attempting to subscribe to the presence or dialog event packages, then here are some troubleshooting steps for determining what is going wrong.

- Ensure that the `res_pjsip_exten_state.so` module is loaded.
- Ensure that the Event header in inbound subscribe messages are one of "presence" or "dialog".
- Ensure all necessary modules are loaded, depending on what values are in the Accept header of inbound SUBSCRIBE requests.
  - Subscriptions that use `Accept: application/pidf+xml` will need to have `res_pjsip_pidf_body_generator.so` loaded.
  - Subscriptions that use `Accept: application/xpidf+xml` will need to have `res_pjsip_xpidf_body_generator.so` loaded.
  - Subscriptions that use `Accept: application/dialog-info+xml` will need to have `res_pjsip_dialog_info_body_generator.so` loaded.
- When subscribing, you may see a message like the following on the CLI:

```
[2014-10-16 12:56:58.605] WARNING[3780]: res_pjsip_exten_state.c:337 new_subscribe: Extension blah does not exist or has no associated hint
```

The warning message is self-explanatory. If you think you have placed extension "blah" in your `extensions.conf` file and it contains a hint, then be sure that it exists in the same context as the `context` option on the endpoint that is attempting to subscribe. Also be sure that if you have recently changed your `extensions.conf` file that you have saved the changes and run the `dialplan reload` CLI command.

## MWI

If you are attempting to subscribe to the message-summary package, then here are some troubleshooting steps for determining what is going wrong.

- Ensure that the `res_pjsip_mwi.so` and the `res_pjsip_mwi_body_generator.so` modules are loaded.
- Ensure that the AoR that the MWI SUBSCRIBE is being sent to has `mailboxes` configured on it. Note that the option name is `mailboxes` and not `mailbox`.
- When subscribing to MWI, you may see a message like the following:

```
[2014-10-16 13:06:51.323] NOTICE[3963]: res_pjsip_mwi.c:566 mwi_validate_for_aor: Endpoint '200' already configured for unsolicited MWI for mailbox '200'. Denying MWI subscription to 200
```

The most likely cause of something like this is that you have an endpoint and an AoR that both have `mailboxes = 200` in your configuration. The endpoint with `mailboxes = 200` attempts to subscribe to the AoR that has `mailboxes = 200`. In this case, since Asterisk is already sending MWI notifications about mailbox 200 to the endpoint, the subscription to the AoR is denied. To fix this, remove the `mailboxes` option from your endpoint, or configure your device not to attempt to subscribe to MWI.

- Asterisk has multiple ways of having MWI state set, but the most common way is to use `app_voicemail` that comes with Asterisk. `app_voicemail` has a requirement that mailbox names must follow the format "mailbox@context". If you are using `app_voicemail` and you configure MWI in `pjsip.conf` and only provide the mailbox name without a context, then you will not receive MWI updates when the state of the mailbox changes.

## Configuration Issues

### Can't create an IPv6 transport

You've configured a transport in `pjsip.conf` to bind to an IPv6 address or block. However, Asterisk fails to create the transport when loading!

If you look into your logs you might see messages similar to the following:

```
[Dec 12 00:58:31] ERROR[10157] config_options.c: Error parsing bind=: at line 8 of
[Dec 12 00:58:31] ERROR[10157] res_sorcery_config.c: Could not create an object of type 'transport' with id 'my-ipv6-transport'
from configuration file 'pjsip.conf'
```

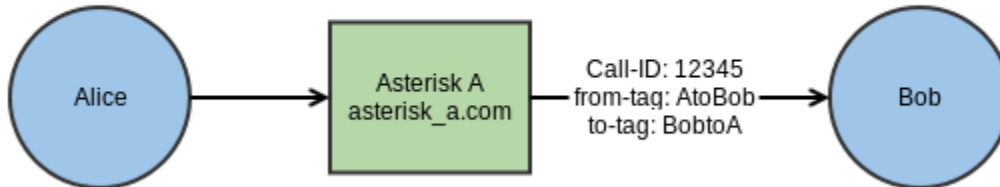
The most likely issue is that you have not compiled **pjproject** with support for IPv6. You can find instructions at [Building and Installing pjproject](#).

## res\_pjsip Remote Attended Transfers

### What is a remote SIP transfer?

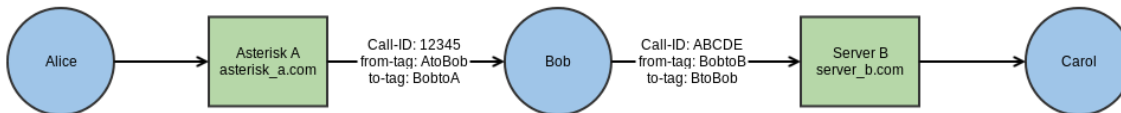
Let's imagine a scenario where Alice places a call to Bob, and then Bob performs an attended transfer to Carol. In this scenario, Alice is registered to Asterisk instance A (asterisk\_a.com), and Bob is registered to Server B (server\_b.com), a non-Asterisk PBX. The key to this scenario is that Asterisk A has been explicitly configured to be able to call Bob directly, despite the fact that Bob does not register to Asterisk A.

Initially, Alice places a call to Bob through Alice's Asterisk instance:



The arrows indicate the direction of the initial call. The Call-ID, from-tag, and to-tag will become important later.

Now, Bob wants to perform an attended transfer to Carol, so he places a call to Carol:



As you can see, Bob has simultaneous calls through two separate servers. Now when Bob performs the attended transfer, what happens? Bob will send a SIP REFER request to either Asterisk A or Server B to get the two SIP servers in contact with each other. Most phones will send the REFER to Asterisk A since it is the original leg of the call, so that is what we will do in our scenario. The REFER request has a Refer-To header that specifies details of the transfer. The Refer-To header for this particular transfer looks like the following:

```
Refer-To: <sip:carol@server_b.com?Replaces=ABCDE%3Bto-tag%3DBtoBobfrom-tag%3DBtoBob>
```

That's a bit verbose. So let's break it down a little bit. First, there is a SIP URI:

```
sip:carol@server_b.com
```

Next, there is a Replaces URI header. There are some URL-escaped character sequences in there. If we decode them, we get the following:

```
Replaces: ABCDE;to-tag=BtoBob;from-tag=BobtoB
```

If we break down the parts of this, what the Replaces section tells us is that the REFER request is saying that the SIP dialog with Call-ID "ABCDE", to-tag "BtoBob" and from-tag "BobtoB" needs to be replaced by the party (or parties) that Bob is talking to.

Asterisk has built into it a bit of an optimization to avoid unnecessary SIP traffic by looking up the dialog referred to by the Replaces header. If the dialog is found in the Asterisk system, then Asterisk simply performs a local attended transfer. This involves internal operations such as moving a channel from one bridge to another, or creating a local channel to link two bridges together.

However, in this case, the dialog referred to by Bob's Replaces header is not on Asterisk A. It is on Server B. So Asterisk A cannot perform a local attended transfer. This is where a remote attended transfer is required.

### From a SIP point of view

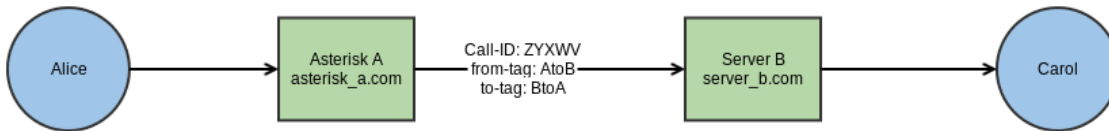
Remote attended transfers are the type of attended transfers referred to in SIP specifications, such as [RFC 5589 section 7](#). When a SIP user agent receives a REFER request, the user agent is supposed to send an INVITE to the URI in the Refer-To header to start a new call with the user agent at that URI. The INVITE should have a Replaces header that has the same contents as the Replaces URI header from the REFER request. This tells the user agent that receives the INVITE to replace the referenced dialog with this new call instead.

In the scenario above, when Asterisk A receives the REFER request from Bob, Asterisk A should respond by sending an INVITE to sip:carol@server\_b.com and add

```
Replaces: ABCDE;to-tag=BtoBob;from-tag=BobtoB
```

When Server B receives this INVITE, it will essentially swap this new call in for the call referenced by the Replaces header. By doing this, the final picture

looks something like the following:



A new dialog with Call-ID ZYXWV has replaced the previous dialog with Call-ID ABCDE. The previously-illustrated dialog between Bob and Asterisk A with Call-ID 12345 is gone because Bob's phone ended that dialog once the transfer was completed.

## How Asterisk handles this

Asterisk will rarely ever directly place outbound calls without going through the dialplan. When Asterisk A receives the REFER request from Bob, Asterisk does not immediately send an INVITE with Replaces header to Server B. Instead, Asterisk A looks for a specifically-named extension called "external\_replaces". Asterisk searches for this extension in the context specified by the `TRANSFER_CONTEXT` channel variable on Bob's channel. If `TRANSFER_CONTEXT` is not specified, then Asterisk searches for the extension in Bob's endpoint's context setting. Once in the dialplan, it is the job of the dialplan writer to determine whether to complete the transfer or not.

In the `external_replaces` extension, you will have access to the following channel variables:

- `SIPTRANSFER`: This variable is set to "yes" upon entering the `external_replaces` extension, and indicates that a SIP transfer is happening. This is only useful if, for whatever reason, you are using the `external_replaces` extension for additional purposes than a SIP remote attended transfer.
- `SIPREFERRINGCONTEXT`: This is the dialplan context in which the `external_replaces` extension was found. This may be useful if your `external_replaces` extension calls into subroutines or jumps to other contexts.
- `SIPREFERTOHR`: This is the SIP URI in the Refer-To header in the REFER request sent by the transferring party.

The big reason why Asterisk calls into the dialplan instead of automatically sending an INVITE to the Refer-To URI is for security purposes. If Asterisk automatically sent an INVITE out without going through the dialplan, there are chances that transfers could be used to place calls to unwanted destinations that could, for instance, charge you a lot of money for the call.

## Writing your external\_replaces extension

Now that the theory has been presented, you'll need to write your `external_replaces` extension. One option you have is to not write an `external_replaces` extension at all. This will prevent any remote attended transfers from succeeding.

If you do want to write an `external_replaces` extension, the first thing you want to do is determine if you want to perform the remote attended transfer. `SIPREFERTOHR`, and values provided by the `CHANNEL()` dialplan function can help you to decide if you want to allow the transfer. For instance, you might use `CHANNEL(endpoint)` to see which PJSIP endpoint is performing the transfer, and you can inspect `SIPREFERTOHR` to determine if the transfer is destined for a trusted domain.



Asterisk dialplan contains functions for manipulating strings. However, there is nothing in the dialplan that is specifically designed to parse a URI. While you could use string manipulation functions for looking at URI details, it is recommended that you do so in an AGI script using a programming language that provides URI parsing libraries.

If you decide not to perform the transfer, the simplest thing to do is to call the `Hangup()` application.



Calling `Hangup()` in this situation can have different effects depending on what type of phone Bob is using. Asterisk updates the phone with a notification that the attended transfer failed. It is up to the phone to decide if it wants to try to reinvite itself back into the original conversation with Alice or simply hang up.

If you decide to perform the transfer, the most straightforward way to do this is with the `Dial()` application. Here is an example of how one might complete the transfer

```
exten => external_replaces,1,NoOp()
same => n,Dial(PJSIP/default_outgoing/${SIPREFERTOHR})
```

Let's examine that `Dial()` more closely. First, we're dialing using PJSIP, which is pretty obvious. Next, we have the endpoint name. The endpoint name could be any configured endpoint you want to use to make this call. Remember that endpoint settings are things such as what codecs to use, what user name to place in the from header, etc. By default, if you just dial `PJSIP/some_endpoint`, Asterisk looks at `some_endpoint`'s configured `aors` to determine what location to send the outgoing call to. However, you can override this default behavior and specify a URI to send the call to instead. This is what is being done in this `Dial()` statement. We're dialing using settings for an endpoint called "default\_outgoing", presumably used as a default endpoint for outgoing calls. We're sending the call out to the URI specified by `SIPREFERTOHR` though. Using the scenario on this page, the `Dial()` command would route the call to `sip:carol@server_b`.

## Avoiding Remote Attended Transfers

In Asterisk, remote attended transfers are sometimes necessary, but avoiding them is typically a good idea. The biggest reason is the security concerns of

allowing users to make calls to untrusted domains.

The easiest but most severe way to prevent remote attended transfers is to set `allow_transfer = no` for all endpoints. The problem with doing this is that it also prevents local attended transfers and blind transfers.

A second way has been discussed already, and that is not to write an `external_replaces` extension. This prevents any attempted remote attended transfers from succeeding, but it does not help to prevent the remote attended transfer from happening in the first place.

Another way is to configure your Asterisk server to only call phones that are directly registered to it and trusted SIP servers. In the scenario we have been inspecting, the remote attended transfer could have been avoided by having Asterisk A call Bob through Server B instead of dialing Bob directly. By receiving the initial call through Server B, Bob will send his REFER request to Server B, who being aware of all necessary dialogs, may be able to perform a local attended transfer (assuming it can do the same optimization as Asterisk). Configuring Asterisk this way is not necessarily guaranteed to prevent all remote attended transfer attempts, but it will help to lessen them.

# PJSIP Transport Selection

## Sending Messages

The process by which an underlying transport is chosen for sending of a message is broken up into different steps depending on the type of message.

### ***SIP Request Handling***

#### **1. URI Parsing**

The PJSIP stack fundamentally acts on URIs. When sending to a URI it is parsed into the various parts (user, host, port, user parameters). For the purposes of transport selection the transport parameter is examined. This specifies the type of transport. If this parameter is not present it is assumed to be UDP. This is important because it is used in DNS resolution. If a "sips" URI scheme is used an automatic switchover to TLS will occur.

#### **2. DNS SRV Resolution (If host portion is not an IP address and no port is present in the URI)**

The transport type from above is used to determine which SRV record to look up. This means that the original URI **must** include the transport type for TCP and TLS types UNLESS the "sips" URI scheme is used which automatically switches to TLS.

#### **3a. Transport Selection (No explicit transport provided)**

Now that the underlying type of transport is known and the resolved target exists the transport selection process can begin.

##### ***Connection-less protocols (such as UDP)***

A transport, decided upon by a hashing mechanism, matching the transport type and address family is selected.

##### ***Connection-oriented protocols (such as TCP or TLS)***

An already open connection to the resolved IP address and port is searched for. If the connection exists it is reused for the request. If no connection exists the first transport matching the transport type and address family as configured in pjsip.conf is chosen. It is instructed to establish a new connection to the resolved IP address and port.

#### **On this Page**

- [Sending Messages](#)
  - [SIP Request Handling](#)
  - [SIP Response Handling](#)
- [Best Configuration Strategies](#)
  - [IPv4 Only \(Single Interface\)](#)
  - [IPv4 Only \(Multiple Interfaces\)](#)
  - [IPv6 Only \(Single Interface\)](#)
  - [IPv4+IPv6 Combined \(Single Interface\)](#)
- [Common Issues](#)
  - [Changeover to TCP when sending via UDP](#)
  - [Sending using a transport that is not available](#)

#### **3b. Transport Selection (Explicit transport provided)**

##### ***Connection-less protocols (such as UDP)***

The provided transport is used.

##### ***Connection-oriented protocols (such as TCP or TLS)***

The provided transport is instructed to establish a new connection to the resolved IP address and port.



This does NOT currently attempt to reuse any existing connections. A new one will always be created. This is an issue being tracked under issue [ASTERISK-22658](#).

#### **4. Multihomed Transport Selection (Connection-less protocols)**

Before the message is sent out the transport the routing table is queried to determine what interface it will be going out on.

##### ***Local source interface IP address matches source IP address in message***

The message is left untouched and passed to the transport.



### **Local source interface IP address differs from source IP address in message**

The message contents are updated with the different source address information. If a transport is bound to the new source address the outgoing transport for the message is changed to it.

## **5. Message is sent**

The message is provided to the transport and it is instructed to send it.

## **SIP Response Handling**

### **1. Transport Selection**

#### **Connection-oriented protocols (such as TCP or TLS)**

If the connection the request was received on is still open it is used to send the response.

If no connection exists or the connection is no longer open the first configured transport in pjsip.conf matching the transport type and address family is selected. It is instructed to establish a new connection to the destination IP address and port.

#### **Connection-less protocol with maddr in URI of the topmost Via header**

A transport, decided upon by a hashing mechanism, matching the transport type and address family is selected. The transport type and address family of the transport the request was received on is used.

#### **Connection-less protocol with rport in URI of the topmost Via header**

The transport the request is received on is used as the transport for the response.

#### **Connection-less protocol without rport in URI of the topmost Via header**

A transport, decided upon by a hashing mechanism, matching the transport type and address family is selected. The transport type and address family of the transport the request was received on is used.

## **2. Message is sent**

The message is provided to the selected transport and it is instructed to send it.

## **Best Configuration Strategies**

### **IPv4 Only (Single Interface)**

Configure a wildcard transport. This is simple as it requires no special configuration such as knowing the IP address and has no downsides.

```
[system-udp]
type=transport
protocol=udp
bind=0.0.0.0

[system-tcp]
type=transport
protocol=tcp
bind=0.0.0.0

[system-tls]
type=transport
protocol=tls
bind=0.0.0.0:5061
cert_file=certificate

[phone]
type=endpoint
```

This example includes an endpoint without a transport explicitly defined. Since there is only one transport configured for each address family and transport type each respective one will be used depending on the URI dialed. For requests to this endpoint the logic in section 3a will be used.

### **IPv4 Only (Multiple Interfaces)**

Configure a transport for each interface. This allows each transport to be specified on endpoints and also ensures that the SIP messages contain the right information.

```
[system-internet-udp]
type=transport
protocol=udp
bind=5.5.5.5

[system-internet-tcp]
type=transport
protocol=tcp
bind=5.5.5.5

[system-internet-tls]
type=transport
protocol=tls
bind=5.5.5.5:5061
cert_file=certificate

[system-local-udp]
type=transport
protocol=udp
bind=192.168.1.1

[system-local-tcp]
type=transport
protocol=tcp
bind=192.168.1.1

[system-local-tls]
type=transport
protocol=tls
bind=192.168.1.1:5061
cert_file=certificate

[phone-internet]
type=endpoint
transport=system-internet-udp

[phone-local]
type=endpoint
transport=system-local-udp

[phone-unspecified]
type=endpoint
```

This example includes three endpoints which are each present on different networks. To ensure that outgoing requests to the first two endpoints travel over the correct transport the transport has been explicitly specified on each. For requests to these endpoints the logic in section 3b will be used. For requests to the "phone-unspecified" endpoint since no transport has been explicitly specified the logic in section 3a will be used.

### **IPv6 Only (Single Interface)**

Configure a transport with the IPv6 address:

```
[system-udp6]
type=transport
protocol=udp
bind=[2001:470:e20f:42::42]

[system-tcp6]
type=transport
protocol=tcp
bind=[2001:470:e20f:42::42]
```

### **IPv4+IPv6 Combined (Single Interface)**

Configure two transports, one with the IPv4 address and one with the IPv6 address.

```
[system-udp]
type=transport
protocol=udp
bind=192.168.1.1

[system-tcp]
type=transport
protocol=tcp
bind=192.168.1.1

[system-udp6]
type=transport
protocol=udp
bind=[2001:470:e20f:42::42]

[system-tcp6]
type=transport
protocol=tcp
bind=[2001:470:e20f:42::42]
```



It might be tempting to use a wildcard IPv6 address to bind a single transport to allow both IPv6 and IPv4. In this configuration IPv6 mapped IPv4 addresses will be used which is unsupported by PJSIP. This will cause a SIP message parsing failure.

## **Common Issues**

### **Changeover to TCP when sending via UDP**

If you turn the "disable\_tcp\_switch" option off in the pjsip.conf system section it is possible for an automatic switch to TCP to occur when sending a large message out using UDP. If your system has not been configured with a TCP transport this will fail. The sending of the message may also fail if the remote side is not listening on TCP.

### **Sending using a transport that is not available**

If a transport can not be found during the transport selection process you will receive a warning message:

```
Failed to send Request msg INVITE/cseq=7846 (tdta0x7fa920002e50)! err=171060 (Unsupported transport (PJSIP_EUNSUPTRANSPORT))
```

This can occur due to using a transport type (such as TCP) or address family when a transport meeting the requirements does not exist.

## PJSIP Configuration Wizard

The PJSIP Configuration Wizard (module `res_pjsip_config_wizard`) is a new feature in Asterisk 13.2.0. While the basic `chan_pjsip` configuration objects (endpoint, aor, etc.) allow a great deal of flexibility and control they can also make configuring standard scenarios like `trunk` and `user` more complicated than similar scenarios in `sip.conf` and `users.conf`. The PJSIP Configuration Wizard aims to ease that burden by providing a single object called 'wizard' that be used to configure most common `chan_pjsip` scenarios.

The following configurations demonstrate a simple ITSP scenario.

pjsip_wizard.conf	pjsip.conf
<pre>[my-itsp] type = wizard sends_auth = yes sends_registrations = yes remote_hosts = sip.my-itsp.net outbound_auth/username = my_username outbound_auth/password = my_password endpoint/context = default aor/qualify_frequency = 15</pre>	<pre>[my-itsp] type = endpoint aors = my-itsp outbound_auth = my-itsp-auth context = default  [my-itsp] type = aor contact = sip:sip.my-itsp.net qualify_frequency = 15  [my-itsp-auth] type = auth auth_type = userpass username = my_username password = my_password  [my-itsp-reg] type = registration outbound_auth = my-itsp-auth server_uri = sip:sip.my-itsp.net client_uri = sip:my_username@sip.my-itsp.net  [my-itsp-identify] type = identify endpoint = my-itsp match = sip.my-itsp.net</pre>

Both produce the same result. In fact, the wizard creates standard `chan_pjsip` objects behind the scenes. In the above example...

- An endpoint and aor are created with the same name as the wizard.
- The `endpoint/context` and `aor/qualify_frequency` parameters are added to them.
- `remote_hosts` captures the remote host for all objects.
- A contact for the aor is created for each remote host.
- `sends_auth=yes` causes an auth object to be created.
- `outbound_auth/username` and `outbound_auth/password` are added to it.
- An `outbound_auth` line is added to the endpoint.
- `sends_registrations=yes` causes a registration object to be created.
- An `outbound_auth` line is added to the registration.
- The `server_uri` and `client_uri` are constructed using the remote host and username.
- An identify object is created and a match is added for each remote host.

### Configuration Reference:

Parameter	Description
type	Must be wizard

sends_auth	Will create an outbound auth object for the endpoint and registration. At least <code>outbound/username</code> must be specified.  default = no
accepts_auth	Will create an inbound auth object for the endpoint. At least 'inbound/username' must be specified.  default = no
sends_registrations	Will create an outbound registration object for each host in <code>remote_hosts</code> .  default = no
remote_hosts	A comma separated list of remote hosts in the form of <code>&lt;ipaddress   hostname&gt;[:port][, ... ]</code> If specified, a static contact for each host will be created in the aor. If <code>accepts_registrations</code> is no, an identify object is also created with a match line for each remote host. Hostnames must resolve to A, AAAA or CNAME records. SRV records are not currently supported.  default = ""
transport	The transport to use for the endpoint and registrations  default = the pjsip default
server_uri_pattern	The pattern used to construct the registration <code>server_uri</code> . The replaceable parameter <code>\${REMOTE_HOST}</code> is available for use.  default = sip:\${REMOTE_HOST}
client_uri_pattern	The pattern used to construct the registration <code>client_uri</code> . The replaceable parameters <code>\${REMOTE_HOST}</code> and <code>\${USERNAME}</code> are available for use.  default = {{sip:\${USERNAME}@\${REMOTE_HOST}}}
contact_pattern	The pattern used to construct the aor contact. The replaceable parameter <code>\${REMOTE_HOST}</code> is available for use.  default = sip:\${REMOTE_HOST}
has_phoneprov	Will create a phoneprov object. If yes, both <code>phoneprov/MAC</code> and <code>phoneprov/PROFILE</code> must be specified.  default = no
has_hint	Enables the automatic creation of dialplan hints.  Two entries will be created. One hint for 'hint_exten' and one application to execute when 'hint_exten' is dialed.
hint_context	The context into which hints are placed.
hint_exten	The extension this hint will be registered with.
hint_application	An application with parameters to execute when 'hint_exten' is dialed.  Example: <code>Gosub(stdexten, \${EXTEN}, 1( \${HINT} ) )</code>
<object>/<parameter>	These parameters are passed unmodified to the native object.

### Configuration Notes:

- Wizards must be defined in `pjsip_wizard.conf`.
- Using `pjsip_wizard.conf` doesn't remove the need for `pjsip.conf` or any other config file.
- Transport, system and global sections still need to be defined in `pjsip.conf`.
- You can continue to create discrete endpoint, aor, etc. objects in `pjsip.conf` but there can be no name collisions between wizard created objects and discretely created objects.

- An endpoint and aor are created for each wizard.
  - The endpoint and aor are named the same as the wizard.
  - Parameters are passed to them using the `endpoint/` and `aor/` prefixes.
  - A contact is added to the aor for each remote host using the `contact_pattern` and `${REMOTE_HOST}`.
- `sends_auth` causes an auth object to be created.
  - The name will be `<wizard_name>-oauth`.
  - Parameters are passed to it using the `outbound_auth/` prefix.
  - The endpoint automatically has an `outbound_auth` parameter added to it.
  - Registrations automatically have an `outbound_auth` parameter added to them (if registrations are created, see below).
- `accepts_auth` causes an auth object to be created.
  - The name will be `<wizard_name>-iauth`.
  - Parameters are passed to it using the `inbound_auth/` prefix.
  - The endpoint automatically has an `auth` parameter added to it.
- `sends_registrations` causes an outbound registration object to be created for each remote host.
  - The name will be `<wizard_name>-reg-<n>` where n starts at 0 and increments by 1 for each remote host.
  - Parameters are passed to them using the `registration/` prefix.
  - You should not use a wizard in situations where you need to pass different parameters to each registration.
  - `server_uri` and `client_uri` are constructed using their respective patterns using `${REMOTE_HOST}` and `${USERNAME}`.
- If `accepts_registrations` is specified, `remote_hosts` must NOT be specified and no contacts are added to the aor. This causes registrations to be accepted.
- If `accepts_registrations` is NOT specified or set to `no`, then an identify object is created to match incoming requests to the endpoint.
  - The name will be `<wizard_name>-identify`.
  - Parameters are passed to it using the `identify/` prefix although there really aren't any to pass.
- If `has_phoneprov` is specified, a phoneprov object is created.
  - The name will be `<wizard_name>-phoneprov`.
  - Both `phoneprov/MAC` and `phoneprov/PROFILE` must be specified.
- `has_hint` causes hints to be automatically created.
  - `hint_exten` must be specified.
- All created objects must pass the same edit criteria they would have to pass if they were specified discretely.
- All created objects will have a `@pjsip_wizard=<wizard_name>` parameter added to them otherwise they are indistinguishable from discretely created ones.
- All created object are visible via the CLI and AMI as though they were created discretely.

**Full Examples:**

Phones:

Configuration	Notes
---------------	-------

```

[user_defaults](!)
type = wizard
transport = ipv4
accepts_registrations = yes
sends_registrations = no
accepts_auth = yes
sends_auth = no
has_hint = yes
hint_context = DLPN_DialPlan1
hint_application =
Gosub(stdexten, ${EXTEN}, 1( ${HINT} ))
endpoint/context = DLPN_DialPlan1
endpoint/allow_subscribe = yes
endpoint/allow = !all,ulaw,gsm,g722
endpoint/direct_media = yes
endpoint/force_rport = yes
endpoint/disable_direct_media_on_nat = yes
endpoint/direct_media_method = invite
endpoint/ice_support = yes
endpoint/moh_suggest = default
endpoint/send_rpid = yes
endpoint/rewrite_contact = yes
endpoint/send_pai = yes
endpoint/allow_transfer = yes
endpoint/trust_id_inbound = yes
endpoint/device_state_busy_at = 1
endpoint/trust_id_outbound = yes
endpoint/send_diversion = yes
aor/qualify_frequency = 30
aor/authenticate_qualify = no
aor/max_contacts = 1
aor/remove_existing = yes
aor/minimum_expiration = 30
aor/support_path = yes
phoneprov/PROFILE = profile1

[bob](user_defaults)
hint_exten = 1000
inbound_auth/username = bob
inbound_auth/password = bobspassword

[alice](user_defaults)
hint_exten = 1001
endpoint/callerid = Alice <1001>
endpoint/allow = !all,ulaw
inbound_auth/username = alice
inbound_auth/password = alicespassword
has_phoneprov = yes
phoneprov/MAC = deadbeef4dad

```

This example demonstrates the power of both wizards and templates.

Once the template is created, adding a new phone could be as simple as creating a wizard object with nothing more than a username and password. You don't even have to specify 'type' because it's inherited from the template.

Of course, you can override ANYTHING in the wizard object or specify everything and not use templates at all.

Trunk to an ITSP requiring registration:

Configuration	Notes
---------------	-------



```

[trunk_defaults](!)
type = wizard
transport = ipv4
endpoint/allow_subscribe = no
endpoint/allow = !all,ulaw
aor/qualify_frequency = 30
registration/expiration = 1800

[myitsp](trunk_defaults)
sends_auth = yes
sends_registrations = yes
endpoint/context = DID_myitsp
remote_hosts = sip1.myitsp.net,sip2.myitsp.net
accepts_registrations = no
endpoint/send_rpid = yes
endpoint/send_pai = yes
outbound_auth/username = my_username
outbound_auth/password = my_password

[my_other_itsp](trunk_defaults)
sends_auth = yes
sends_registrations = yes
endpoint/context = DID_myitsp
remote_hosts = sip1.my-other-itsp.net,sip2.my-other-itsp.net
accepts_registrations = no
endpoint/send_rpid = yes
endpoint/send_pai = yes
outbound_auth/username = my_username
outbound_auth/password = my_password
registration/expiration = 900
registration/support_path = no

```

This is an example of trunks to 2 different ITSPs each of which has a primary and backup server.

It also shows most of the endpoint and aor parameters being left at their defaults.

In this scenario, each wizard object takes the place of an endpoint, aor, auth, identify and 2 registrations.

#### Trunk between trusted peers:

Configuration	Notes
<pre> [trusted-peer](trunk_defaults) endpoint/context = peer_context remote_hosts = sip1.peer.com:45060 sends_registrations = no accepts_registrations = no sends_auth = no accepts_auth = no </pre>	<p>This one's even simpler. The <code>sends_</code> and <code>accepts_</code> parameters all default to <code>no</code> so you don't really even have to specify them unless your template has them set to <code>yes</code>.</p>

## Configuring res\_pjsip for IPv6

### Tell Asterisk and PJSIP to Speak IPv6

The configuration described here happens in the pjsip.conf file within transport and endpoint sections. For more information about the transport side of things see [PJSIP Transport Selection](#)

#### **Bind PJSIP to a specific interface**

To configure res\_pjsip for communication over an IPv6 interface you must modify the bind address for your transports in pjsip.conf.

```
[transport-udp6]
type=transport
protocol=udp
bind=[fe80::5e26:aff:fe4b:4399]

[transport-tcp6]
type=transport
protocol=tcp
bind=[fe80::5e26:aff:fe4b:4399]
```

#### **Bind PJSIP to the first available IPv6 interface**

A transport can be configured to automatically bind to the first available IPv6 interface. You use "::" as the bind address.

```
[transport-auto-ipv6]
type=transport
protocol=udp
bind=::
```

#### **Configure a PJSIP endpoint to use RTP over IPv6**

If the communication involves an endpoint (almost always) then the endpoint must be configured to allow RTP via ipv6.

```
[mytrunk]
type=endpoint
transport=transport-udp6
context=from-external
disallow=all
allow=ulaw
rtp_ipv6=yes
```

## Publishing Extension State

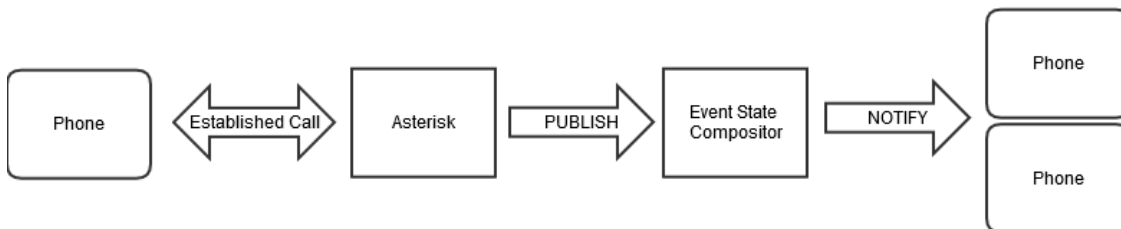
### Background

Functionality exists within PJSIP, as of Asterisk 14, that allows extension state to be published to another entity, commonly referred to as an event state compositor. Instead of each device subscribing to Asterisk and receiving a NOTIFY as extension state changes, PJSIP can be configured to send a single PUBLISH request for each extension state change to the other entity. These PUBLISH requests are triggered based on extension state changes made to hints in the dialplan.



### Why Do It

Publishing extension state allows the SUBSCRIBE and NOTIFY functionality to be handled by the other entity. Each device subscribes to the event state compositor and receives NOTIFY messages from it instead. This can scale further as less state is present in Asterisk, and also allows multiple Asterisk instances to be used while still making extension state available to everyone from the central event state compositor.



### What Can Be Published?

PJSIP has a pluggable body type system. Any type that can be subscribed to for extension state can be published. As of this writing the available body types are:

- application/dialog-info+xml
- application/pdf+xml
- application/xpdf+xml

The PUBLISH request will contain the same body that a NOTIFY request would.

### Configuration

The publishing of extension state is configured by specifying an **outbound publish** in the pjsip.conf configuration file. This tells PJSIP how to publish to another entity and gives it information about what to publish. The outbound publishing of extension state has some additional arguments, though, which allow more control.

The **@body** option specifies what body type to publish. This is a required option.

The **@context** option specifies a filter for context. This is a regular expression and is optional.

The **@exten** option specifies a filter for extensions. This is a regular expression and is optional.

An additional option which is required on the outbound publish is the **multi\_user** option. This enables support in the outbound publish module for publishing to different users. This is needed for extension state publishing so the specific extension can be published to. Without this option enabled all PUBLISH requests would go to the same user.

### Example Configuration

*This configuration would limit outbound publish to only extension state changes as a result of a hint named "1000" in the context "users".*

```
[test-esc]
type=outbound-publish
server_uri=sip:172.16.0.100
from_uri=sip:172.16.0.100
event=dialog
multi_user=yes
@body=application/dialog-info+xml
@context=^users
@exten=^1000
```

**This configuration would limit outbound publish to all extension state changes a result of hints in the context "users".**

```
[test-esc]
type=outbound-publish
server_uri=sip:172.16.0.100
from_uri=sip:172.16.0.100
event=dialog
multi_user=yes
@body=application/dialog-info+xml
@context=^users
```

You are also not limited to a single configured outbound publish. You can have as many as you want, provided they have different names. Each one can go to the same server with a different body type, or to different servers.

### **What About Making It More Dynamic?**

As part of the work to implement the publishing of extension state, the concept of **autohints** were also created. Autohints are created automatically as a result of a device state change. The extension name used is the name of the device, without the technology. They can be enabled by setting "autohints=yes" in a context in extensions.conf like so:

```
[users]
autohints=yes
```

For example, once enabled, if a device state change occurs for "PJSIP/alice" and no hint named "alice" exists, then one will be automatically created in lieu of explicit definition of the following:

```
exten => alice,hint,PJSIP/alice
```

Despite being added after startup, this hint will still be given to the extension state publishing for publishing.

### **The Other Entity**

Throughout this page, I've mentioned another entity; but what can you use? Kamailio! Kamailio has event state compositor support available using the [presence module](#). It can be configured to accept SUBSCRIBE and PUBLISH requests, persist information in a database, and to then send NOTIFY messages to each subscribed device. The module exports the handle\_publish and handle\_subscribe functions for handling each.

This module works perfectly with the PJSIP extension state publishing support. The Asterisk configuration needs to use a URI to the Kamailio server and the Kamailio server has to explicitly trust traffic from the Asterisk instance, or authentication needs to be configured.

# Real-time Text (T.140)

## Real-time text in Asterisk

The SIP channel has support for real-time text conversation calls in Asterisk (T.140). This is a way to perform text based conversations in combination with other media, most often video. The text is sent character by character as a media stream.

During a call sometimes there are losses of T.140 packets and a solution to that is to use redundancy in the media stream (RTP). See "[http://en.wikipedia.org/wiki/Text\\_over\\_IP](http://en.wikipedia.org/wiki/Text_over_IP)"[http://en.wikipedia.org/wiki/Text\\_over\\_IP](http://en.wikipedia.org/wiki/Text_over_IP) and RFC 5194 for more information.

The supported real-time text codec is t.140.

Real-time text redundancy support is now available in Asterisk.

### **ITU-T T.140**

You can find more information about T.140 at [www.itu.int](http://www.itu.int). RTP is used for the transport T.140, as specified in RFC 4103.

### **How to enable T.140**

In order to enable real-time text with redundancy in Asterisk, modify sip.conf to add:

```
[general]
disallow=all
allow=ulaw
allow = alaw
allow=t140
allow=t140red
textsupport=yes
```

The codec settings may change, depending on your phones. The important settings here are to allow t140 and 140red and enable text support.

### **General information about real-time text support in Asterisk**

With the configuration above, calls will be supported with any combination of real-time text, audio and video.

Text for both t140 and t140red is handled on channel and application level in Asterisk conveyed in Text frames, with the subtype "t140". Text is conveyed in such frames usually only containing one or a few characters from the real-time text flow. The packetization interval is 300 ms, handled on lower RTP level, and transmission redundancy level is 2, causing one original and two redundant transmissions of all text so that it is reliable even in high packet loss situations. Transmitting applications do not need to bother about the transmission interval. The t140red support handles any buffering needed during the packetization intervals.

### **Clients known to support text, audio/text or audio/video/text calls with Asterisk:**

- Omnitor Allan eC - SIP audio/video/text softphone
- AuPix APS-50 - audio/video/text softphone.
- France Telecom eConf â€“audio/video/text softphone.
- SIPcon1 - open source SIP audio/text softphone available in Sourceforge.

### **Credits**

- Asterisk real-time text support is developed by AuPix
- Asterisk real-time text redundancy support is developed by Omnitor

The work with Asterisk real-time text redundancy was supported with funding from the National Institute on Disability and Rehabilitation Research (NIDRR), U.S. Department of Education, under grant number H133E040013 as part of a co-operation between the Telecommunication Access Rehabilitation Engineering Research Center of the University of Wisconsin â€“ Trace Center joint with Gallaudet University, and Omnitor.

Olle E. Johansson, Edvina AB, has been a liason between the Asterisk project and this project.

## Inter-Asterisk eXchange protocol, version 2 (IAX2)

## Why IAX2?

The first question most people are thinking at this point is "Why do you need another VoIP protocol? Why didn't you just use SIP or H.323?"

Well, the answer is a fairly complicated one, but in a nutshell it's like this... Asterisk is intended as a very flexible and powerful communications tool. As such, the primary feature we need from a VoIP protocol is the ability to meet our own goals with Asterisk, and one with enough flexibility that we could use it as a kind of laboratory for inventing and implementing new concepts in the field. Neither H.323 or SIP fit the roles we needed, so we developed our own protocol, which, while not standards based, provides a number of advantages over both SIP and H.323, some of which are:

- **Interoperability with NAT/PAT/Masquerade firewalls** - IAX2 seamlessly interoperates through all sorts of NAT and PAT and other firewalls, including the ability to place and receive calls, and transfer calls to other stations.
- **High performance, low overhead protocol** – When running on low-bandwidth connections, or when running large numbers of calls, optimized bandwidth utilization is imperative. IAX2 uses only 4 bytes of overhead.
- **Internationalization support** – IAX2 transmits language information, so that remote PBX content can be delivered in the native language of the calling party.
- **Remote dialplan polling** – IAX2 allows a PBX or IP phone to poll the availability of a number from a remote server. This allows PBX dialplans to be centralized.
- **Flexible authentication** – IAX2 supports cleartext, MD5, and RSA authentication, providing flexible security models for outgoing calls and registration services.
- **Multimedia protocol** – IAX2 supports the transmission of voice, video, images, text, HTML, DTMF, and URL's. Voice menus can be presented in both audibly and visually.
- **Call statistic gathering** – IAX2 gathers statistics about network performance (including latency and jitter), as well as providing end-to-end latency measurement.
- **Call parameter communication** – Caller\*ID, requested extension, requested context, etc. are all communicated through the call.
- **Single socket design** – IAX2's single socket design allows up to 32768 calls to be multiplexed.

While we value the importance of standards based (i.e. SIP) call handling, hopefully this will provide a reasonable explanation of why we developed IAX2 rather than starting with SIP.

## Introduction to IAX2

This section is intended as an introduction to the Inter-Asterisk eXchange v2 (or simply IAX2) protocol. It provides both a theoretical background and practical information on its use.



## IAX2 Configuration

For examples of a configuration, please see the `iax.conf.sample` in the `/configs` directory of your source code distribution.

## Configuring chan\_ix2 for IPv6

### Configuration

IAX uses the 'bindaddr' and 'bindport' options to specify on which address and port the IAX2 channel driver will listen for incoming requests. They accept IPv6 as well as IPv4 addresses.

### Examples

```
bindport=4569
```

The default port to listen on is 4569. Bindport must be specified **before** bindaddr or may be specified on a specific bindaddr if followed by colon and port (e.g. bindaddr=192.168.0.1:4569).

For IPv6 the address needs to be in brackets then colon and port (e.g. bindaddr=[2001:db8::1]:4569).

```
bindaddr=192.168.0.1:459  
bindaddr=[2001:db8::1]:4569
```

You can specify 'bindaddr' more than once to bind to multiple addresses, but the first will be the default. IPv6 addresses are accepted.



For details IAX configuration examples see the `iax.conf.sample` file that comes with the source.

# IAX2 Jitterbuffer

## *The new jitterbuffer*

You must add `jitterbuffer=yes` to either the `[general]` part of `iax.conf`, or to a peer or a user. (just like the old jitterbuffer). Also, you can set `max_jitterbuffer=n`, which puts a hard-limit on the size of the jitterbuffer of "n milliseconds". It is not necessary to have the new jitterbuffer on both sides of a call; it works on the receive side only.

## **PLC**

The new jitterbuffer detects packet loss. PLC is done to try to recreate these lost packets in the codec decoding stage, as the encoded audio is translated to slinear. PLC is also used to mask jitterbuffer growth.

This facility is enabled by default in iLBC and speex, as it has no additional cost. This facility can be enabled in adpcm, alaw, g726, gsm, lpc10, and ulaw by setting `genericplc = true` in the `plc` section of `codecs.conf`.

## **Trunk Timestamps**

To use this, both sides must be using Asterisk v1.2 or later. Setting `trunktimestamps=yes` in `iax.conf` will cause your box to send 16-bit timestamps for each trunked frame inside of a trunk frame. This will enable you to use jitterbuffer for an IAX2 trunk, something that was not possible in the old architecture.

The other side must also support this functionality, or else, well, bad things will happen. If you don't use trunk timestamps, there's lots of ways the jitterbuffer can get confused because timestamps aren't necessarily sent through the trunk correctly.

## **Communication with Asterisk v1.0.x systems**

You can set up communication with v1.0.x systems with the new jitterbuffer, but you can't use trunks with `trunktimestamps` in this communication.

If you are connecting to an Asterisk server with earlier versions of the software (1.0.x), do not enable both jitterbuffer and trunking for the involved peers/users in order to be able to communicate. Earlier systems will not support `trunktimestamps`.

You may also compile `chan_iax2.c` without the new jitterbuffer, enabling the old backwards compatible architecture. Look in the source code for instructions.

## **Testing and monitoring**

You can test the effectiveness of PLC and the new jitterbuffer's detection of loss by using the new CLI command `iax2 test losspkt n`. This will simulate `n` percent packet loss coming in to `chan_iax2`. You should find that with PLC and the new JB, 10 percent packet loss should lead to just a tiny amount of distortion, while without PLC, it would lead to silent gaps in your audio.

`iax2 show netstats` shows you statistics for each iax2 call you have up. The columns are "RTT" which is the round-trip time for the last PING, and then a bunch of stats for both the local side (what you're receiving), and the remote side (what the other end is telling us they are seeing). The remote stats may not be complete if the remote end isn't using the new jitterbuffer.

The stats shown are:

- Jit: The jitter we have measured (milliseconds)
- Del: The maximum delay imposed by the jitterbuffer (milliseconds)
- Lost: The number of packets we've detected as lost.
- %: The percentage of packets we've detected as lost recently.
- Drop: The number of packets we've purposely dropped (to lower latency).
- OOO: The number of packets we've received out-of-order
- Kpkts: The number of packets we've received / 1000.

## **Reporting problems**

There's a couple of things that can make calls sound bad using the jitterbuffer:

The JB and PLC can make your calls sound better, but they can't fix everything. If you lost 10 frames in a row, it can't possibly fix that. It really can't help much more than one or two consecutive frames.

- Bad timestamps: If whatever is generating timestamps to be sent to you generates nonsensical timestamps, it can confuse the jitterbuffer. In particular, discontinuities in timestamps will really upset it: Things like timestamps sequences which go 0, 20, 40, 60, 80, 34000, 34020, 34040, 34060... It's going to think you've got about 34 seconds of jitter in this case, etc.. The right solution to this is to find out what's causing the sender to send us such nonsense, and fix that. But we should also figure out how to make the receiver more robust in cases like this. `chan_iax2` will actually help fix this a bit if it's more than 3 seconds or so, but at some point we should try to think of a better way to detect this kind of thing and resynchronize.
- Different clock rates are handled very gracefully though; it will actually deal with a sender sending 20% faster or slower than you expect just fine.
- Really strange network delays: If your network "pauses" for like 5 seconds, and then when it restarts, you are sent some packets that are

5 seconds old, we are going to see that as a lot of jitter. We already throw away up to the worst 20 frames like this, though, and the "maxjitterbuffer" parameter should put a limit on what we do in this case.

# IAX2 Security

## IAX2 Security

Copyright (c) 2009 - Digium, Inc.

All Rights Reserved.

Document Version 1.0

09/03/09

Asterisk Development Team <asteriskteam@digium.com>

- 1 Introduction
  - 1.1 Overview
- 2 User Guide
  - 2.1 Configuration
    - 2.1.1 Quick Start
    - 2.1.2 Controlled Networks
      - 2.1.2.1 Full Upgrade
      - 2.1.2.2 Partial Upgrade
    - 2.1.3 Public Networks
      - 2.1.3.1 Full Upgrade
      - 2.1.3.2 Partial Upgrade
      - 2.1.3.3 Guest Access
  - 2.2 CLI Commands
    - 2.2.1 `iax2 show callnumber usage`
    - 2.2.2 `iax2 show peer`
- 3 Protocol Modification
  - 3.1 Overview
  - 3.2 Call Token Validation
  - 3.3 Example Message Exchanges
    - 3.3.1 Call Setup
    - 3.3.2 Call Setup, client does not support CALLTOKEN
    - 3.3.3 Call Setup, client supports CALLTOKEN, server does not
    - 3.3.4 Call Setup from client that sends invalid token
- 4 Asterisk Implementation
  - 4.1 CALLTOKEN IE Payload

## Introduction

### Overview

A change has been made to the IAX2 protocol to help mitigate denial of service attacks. This change is referred to as call token validation. This change affects how messages are exchanged and is not backwards compatible for an older client connecting to an updated server, so a number of options have been provided to disable call token validation as needed for compatibility purposes.

In addition to call token validation, Asterisk can now also limit the number of connections allowed per IP address to disallow one host from preventing other hosts from making successful connections. These options are referred to as call number limits.

For additional details about the configuration options referenced in this document, see the sample configuration file, `iax.conf.sample`. For information regarding the details of the call token validation protocol modification, see [#Protocol Modification](#).

## User Guide

### Configuration

#### Quick Start

We strongly recommend that administrators leave the IAX2 security enhancements in place where possible. However, to bypass the security enhancements completely and have Asterisk work exactly as it did before, the following options can be specified in the `[general]` section of `iax.conf`:

```
iax.conf

[general]
...
calltokenoptional = 0.0.0.0/0.0.0.0
maxcallnumbers = 16382
...
```

## Controlled Networks

This section discusses what needs to be done for an Asterisk server on a network where no unsolicited traffic will reach the IAX2 service.

### Full Upgrade

If all IAX2 endpoints have been upgraded, then no changes to configuration need to be made.

### Partial Upgrade

If only some of the IAX2 endpoints have been upgraded, then some configuration changes will need to be made for interoperability. Since this is for a controlled network, the easiest thing to do is to disable call token validation completely, as described under [#Quick Start](#).

## Public Networks

This section discusses the use of the IAX2 security functionality on public networks where it is possible to receive unsolicited IAX2 traffic.

### Full Upgrade

If all IAX2 endpoints have been upgraded to support call token validation, then no changes need to be made. However, for enhanced security, the administrator may adjust call number limits to further reduce the potential impact of malicious call number consumption. The following configuration will allow known peers to consume more call numbers than unknown source IP addresses:

```
iax.conf

[general]
; By default, restrict call number usage to a low number.
maxcallnumbers = 16
...

[callnumberlimits]
; For peers with known IP addresses, call number limits can
; be set in this section. This limit is per IP address for
; addresses that fall in the specified range.
; <IP>/<mask> = <limit>
192.168.1.0/255.255.255.0 = 1024
...

[peerA]
; Since we won't know the IP address of a dynamic peer until
; they register, a max call number limit can be set in a
; dynamic peer configuration section.
type = peer
host = dynamic
maxcallnumbers = 1024
...
```

### Partial Upgrade

If only some IAX2 endpoints have been upgraded, or the status of an IAX2 endpoint is unknown, then call token validation must be disabled to ensure interoperability. To reduce the potential impact of disabling call token validation, it should only be disabled for a specific peer or user as needed. By using the auto option, call token validation will be changed to required as soon as we determine that the peer supports it.

```
iax.conf

[friendA]
requirecalltoken = auto
...
```

Note that there are some cases where auto should not be used. For example, if multiple peers use the same authentication details, and they have not all upgraded to support call token validation, then the ones that do not support it will get locked out. Once an upgraded client successfully completes an

authenticated call setup using call token validation, Asterisk will require it from then on. In that case, it would be better to set the `requirecalltoken` option to `no`.

### Guest Access

Guest access via IAX2 requires special attention. Given the number of existing IAX2 endpoints that do not support call token validation, most systems that allow guest access should do so without requiring call token validation.

```
iax.conf

[guest]
; Note that the name "guest" is special here. When the code
; tries to determine if call token validation is required, it
; will look for a user by the username specified in the
; request. Guest calls can be sent without a username. In
; that case, we will look for a defined user called "guest" to
; determine if call token validation is required or not.
type = user
requirecalltoken = no
...
```

Since disabling call token validation for the guest account allows a huge hole for malicious call number consumption, an option has been provided to segregate the call numbers consumed by connections not using call token validation from those that do. That way, there are resources dedicated to the more secure connections to ensure that service is not interrupted for them.

```
iax.conf

[general]
maxcallnumbers_nonvalidated = 2048
...
```

### CLI Commands

`iax2 show callnumber usage`

Usage: `iax2 show callnumber usage [IP address]`

Show current IP addresses which are consuming IAX2 call numbers.

`iax2 show peer`

This command will now also show the configured call number limit and whether or not call token validation is required for this peer.

### Protocol Modification

This section discusses the modification that has been made to the IAX2 protocol. This information would be most useful to implementors of IAX2.

### Overview

The IAX2 protocol uses a call number to associate messages with which call they belong to. The available amount of call numbers is finite as defined by the protocol. Because of this, it is important to prevent attackers from maliciously consuming call numbers. To achieve this, an enhancement to the IAX2 protocol has been made which is referred to as call token validation.

Call token validation ensures that an IAX2 connection is not coming from a spoofed IP address. In addition to using call token validation, Asterisk will also limit how many call numbers may be consumed by a given remote IP address. These limits have defaults that will usually not need to be changed, but can be modified for a specific need.

The combination of call token validation and call number limits is used to mitigate a denial of service attack to consume all available IAX2 call numbers. An alternative approach to securing IAX2 would be to use a security layer on top of IAX2, such as DTLS [RFC 4347](#) or IPsec [RFC 4301](#).

### Call Token Validation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

For this section, when the word "request" is used, it is referring to the command that starts an IAX2 dialog.

This modification adds a new IAX2 frame type, and a new information element be defined.

Frame Type: CALLTOKEN — 0x28 (40)

IE: CALLTOKEN — 0x36 (54)

When a request is initially sent, it SHOULD include the CALLTOKEN IE with a zero-length payload to indicate that this client supports the CALLTOKEN exchange. When a server receives this request, it MUST respond with the IAX2 message CALLTOKEN. The CALLTOKEN message MUST be sent with a source call number of 0, as a call number will not yet be allocated for this call.

For the sake of backwards compatibility with clients that do not support token validation, server implementations MAY process requests that do not indicate CALLTOKEN support in their initial request. However, this SHOULD NOT be the default behavior, as it gives up the security benefits gained by CALLTOKEN validation.

After a client sends a request with an empty CALLTOKEN IE, it MUST be prepared to receive a CALLTOKEN response, or to receive a response that would be given in the case of a valid CALLTOKEN. This is how a client must behave to inter operate with IAX2 server implementations that do not yet support CALLTOKEN validation.

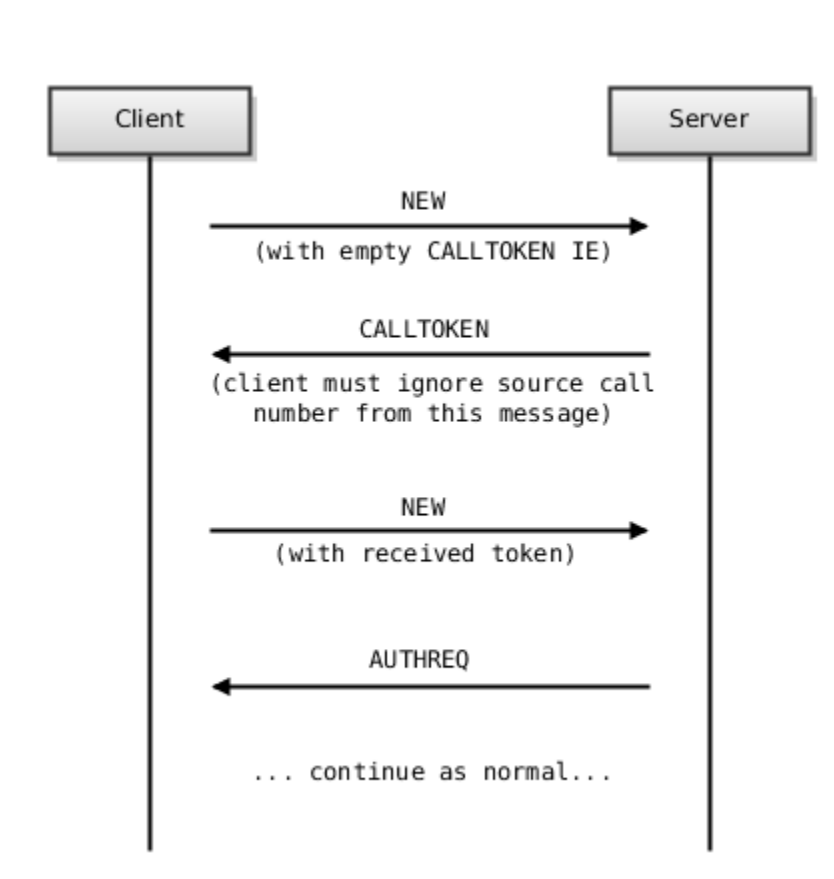
When an IAX2 client receives a CALLTOKEN response, it MUST send its initial request again. This request MUST include the CALLTOKEN IE with a copy of the value of the CALLTOKEN IE received in the CALLTOKEN response. The IE value is an opaque value. Clients MUST be able to accept a CALLTOKEN payload of any length, up to the maximum length allowed in an IAX2 IE.

The value of the payload in the CALLTOKEN IE is an implementation detail. It is left to the implementor to decide how sophisticated it should be. However, it MUST be enough such that when the CALLTOKEN IE is sent back, it can be used to verify that the source IP address and port number has not been spoofed.

If a server receives a request with an invalid CALLTOKEN IE value, then it MUST drop it and not respond.

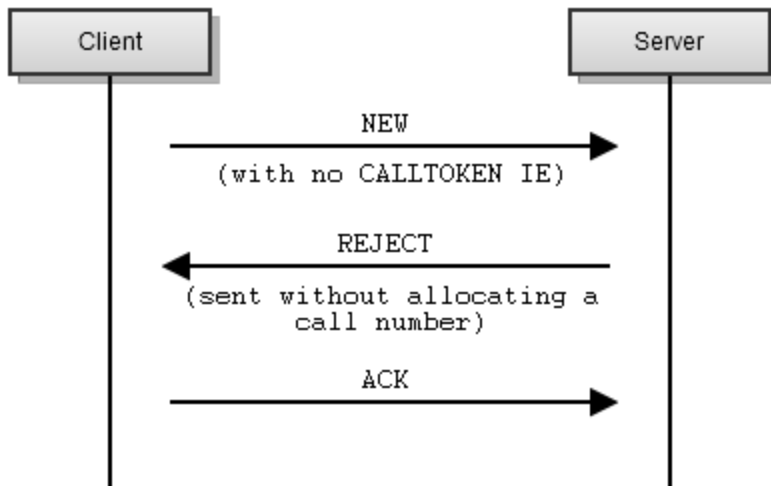
### Example Message Exchanges

#### Call Setup

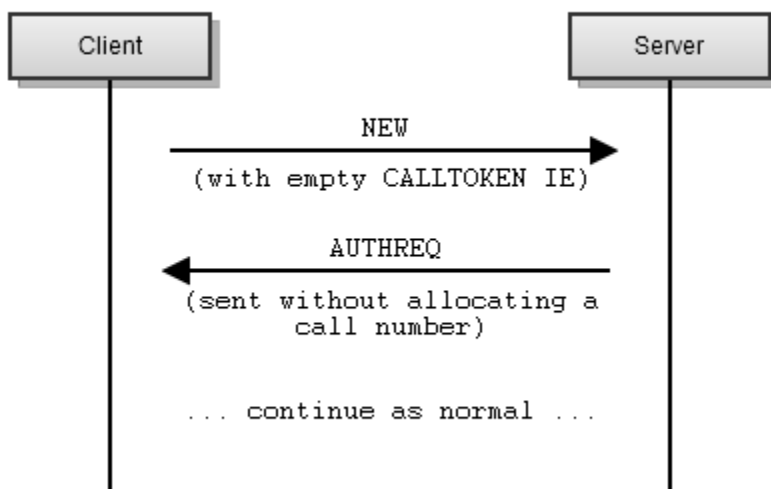


#### Call Setup, client does not support CALLTOKEN

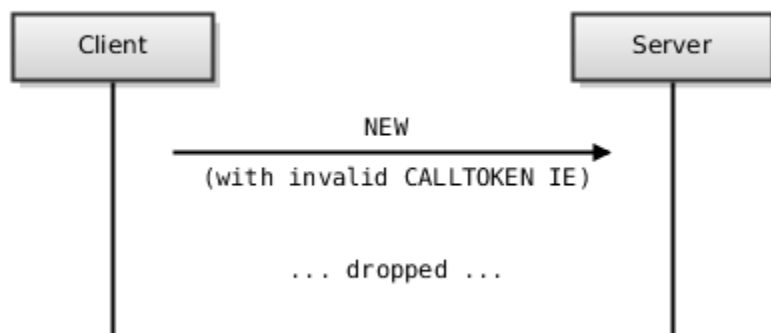




Call Setup, client supports CALLTOKEN, server does not



Call Setup from client that sends invalid token



## Asterisk Implementation

This section includes some additional details on the implementation of these changes in Asterisk.

### ***CALLTOKEN IE Payload***

For Asterisk, we will encode the payload of the CALLTOKEN IE such that the server is able to validate a received token without having to store any information after transmitting the CALLTOKEN response. The CALLTOKEN IE payload will contain:

- A timestamp (epoch based)
- SHA1 hash of the remote IP address and port, the timestamp, as well some random data generated when Asterisk starts.

When a CALLTOKEN IE is received, its validity will be determined by recalculating the SHA1 hash. If it is a valid token, the timestamp is checked to determine if the token is expired. The token timeout will be hard coded at 10 seconds for now. However, it may be made configurable at some point if it seems to be a useful addition. If the server determines that a received token is expired, it will treat it as an invalid token and not respond to the request.

By using this method, we require no additional memory to be allocated for a dialog, other than what is on the stack for processing the initial request, until token validation is complete.

However, one thing to note with this CALLTOKEN IE encoding is that a token would be considered valid by Asterisk every time a client sent it until we considered it an expired token. However, with use of the "maxcallnumbers" option, this is not actually a problem. It just means that an attacker could hit their call number limit a bit quicker since they would only have to acquire a single token per timeout period, instead of a token per request.

# DAHDI



Under Construction



Top-level page for DAHDI **channel driver** information

What are the software and hardware requirements for chan\_dahdi?

How do I configure it?

What are some examples of Analog and PRI configurations?

Links to use:

[Digium Asterisk Hardware Device Interface](#)

# Local Channel

## Overview

Most [Channel Drivers](#) in Asterisk provide capability to connect Asterisk to external devices via specific protocols (e.g. [chan\\_pjsip](#)), whereas Local Channels provide a channel type for calling back into Asterisk itself.

That is, when dialing a Local Channel you are dialing within Asterisk into the Asterisk [dialplan](#).

Usage of Local Channels between other channel technologies can add additional programmatic flexibility, but of course at some level of performance cost. Local Channels are often used to execute dialplan logic from [Applications](#) that would expect to connect directly with a channel.

Two of the most common areas where Local channels are used include members configured for queues, and in use [with callfiles](#). Another interesting case could be that you want to ring multiple destinations, but with different information for each call, such as different callerID for each outgoing request.

In this section you'll find [Local Channel Examples](#) that illustrate usage plus details on [Local Channel Optimization](#) and a list of [Local Channel Modifiers](#).

## The Local Channel in Asterisk Architecture

Previous to Asterisk 12, Local Channel functionality was provided by the `chan_local` module. In Asterisk 12, `chan_local` was moved into the Asterisk system core and is no longer a [loadable module](#).

## Local Channel Examples

Local channels are best demonstrated through the use of an example. In the sub-pages here you'll find several examples of Local Channel usage.

## Delay Dialing Devices Example

Lets say when someone calls extension 201, we want to ring both the desk phone and their cellphone at the same time, but we want to wait about 6 seconds to start dialing the cellphone. This is useful in a situation when someone might be sitting at their desk, but don't want both devices ringing at the same time, but also doesn't want to wait for the full ring cycle to execute on their desk phone before rolling over to their cellphone.

The dialplan for this would look something like the following:

```
[devices]
exten => 201,1,Verbose(2,Call desk phone and cellphone but with delay)
exten => 201,n,Dial(Local/deskphone-201@extensions&Local/cellphone-201@extensions,30)
exten => 201,n,Voicemail(201@default,${IF(${DIALSTATUS} = BUSY)?b:u})
exten => 201,n,Hangup()

[extensions]
; Dial the desk phone
exten => deskphone-201,1,Verbose(2,Dialing desk phone of extension 201)
exten => deskphone-201,n,Dial(SIP/0004f2040001) ; SIP device with MAC address
                                           ; of 0004f2040001

; Dial the cellphone
exten => cellphone-201,1,Verbose(2,Dialing cellphone of extension 201)
exten => cellphone-201,n,Verbose(2,-- Waiting 6 seconds before dialing)
exten => cellphone-201,n,Wait(6)
exten => cellphone-201,n,Dial(DAHDI/g0/14165551212)
```

When someone dials extension 201 in the [devices] context, it will execute the Dial() application, and call two Local channels at the same time:

```
Local/deskphone-201@extensions
Local/cellphone-201@extensions
```

It will then ring both of those extensions for 30 seconds before rolling over to the Voicemail() application and playing the appropriate voicemail recording depending on whether the \${DIALSTATUS} variable returned BUSY or not.

When reaching the deskphone-201 extension, we execute the Dial() application which calls the SIP device configured as '0004f2040001' (the MAC address of the device). When reaching the cellphone-201 extension, we dial the cellphone via the DAHDI channel using group zero (g0) and dialing phone number 1-416-555-1212.

## Dialing Destinations with Different Information

With Asterisk, we can place a call to multiple destinations by separating the technology/destination pair with an ampersand (&). For example, the following Dial() line would ring two separate destinations for 30 seconds:

```
exten => 201,1,Dial(SIP/0004f2040001&DAHDI/g0/14165551212,30)
```

That line would dial both the SIP/0004f2040001 device (likely a SIP device on the network) and dial the phone number 1-416-555-1212 via a DAHDI interface. In our example though, we would be sending the same callerID information to both end points, but perhaps we want to send a different callerID to one of the destinations?

We can send different callerIDs to each of the destinations if we want by using the Local channel. The following example shows how this is possible because we would Dial() two different Local channels from our top level Dial(), and that would then execute some dialplan before sending the call off to the final destinations.

```
[devices]
exten => 201,1,NoOp()
exten => 201,n,Dial(Local/201@internal&Local/201@external,30)
exten => 201,n,Voicemail(201@default,${IF(${DIALSTATUS} = BUSY)?b:u})
exten => 201,n,Hangup()

[internal]
exten => 201,1,Verbose(2,Placing internal call for extension 201)
exten => 201,n,Set(CALLERID(name)=From Sales)
exten => 201,n,Dial(SIP/0004f2040001,30)

[external]
exten => 201,1,Verbose(2,Placing external call for extension 201)
exten => 201,n,Set(CALLERID(name)=Acme Cleaning)
exten => 201,n,Dial(DAHDI/g0/14165551212)
```

With the dialplan above, we've sent two different callerIDs to the destinations:

- "From Sales" was sent to the local device SIP/0004f2040001
- "Acme Cleaning" was sent to the remote number 1-416-555-1212 via DAHDI

Because each of the channels is independent from the other, you could perform any other call manipulation you need. Perhaps the 1-416-555-1212 number is a cell phone and you know you can only ring that device for 18 seconds before the voicemail would pick up. You could then limit the length of time the external number is dialed, but still allow the internal device to be dialed for a longer period of time.

## Trivial Local Channel Example

In our dialplan (extensions.conf), we can Dial() another part of the dialplan through the use Local channels. To do this, we can use the following dialplan:

```
[devices]
exten => 201,1,Verbose(2,Dial another part of the dialplan via the Local chan)
exten => 201,n,Verbose(2,Outside channel: ${CHANNEL})
exten => 201,n,Dial(Local/201@extensions)
exten => 201,n,Hangup()

[extensions]
exten => 201,1,Verbose(2,Made it to the Local channel)
exten => 201,n,Verbose(2,Inside channel: ${CHANNEL})
exten => 201,n,Dial(SIP/some-named-extension,30)
exten => 201,n,Hangup()
```

The output of the dialplan would look something like the following. The output has been broken up with some commentary to explain what we're looking at.

```
- Executing [201@devices:1] Verbose("SIP/my_desk_phone-00000014", "2,Dial another part of the dialplan via the
Local chan") in new stack
== Dial another part of the dialplan via the Local chan
```

We dial extension 201 from SIP/my\_desk\_phone which has entered the [devices] context. The first line simply outputs some information via the Verbose() application.

```
- Executing [201@devices:2] Verbose("SIP/my_desk_phone-00000014",
"2,Outside channel: SIP/my_desk_phone-00000014") in new stack
== Outside channel: SIP/my_desk_phone-00000014
```

The next line is another Verbose() application statement that tells us our current channel name. We can see that the channel executing the current dialplan is a desk phone (aptly named 'my\_desk\_phone').

```
- Executing [201@devices:3] Dial("SIP/my_desk_phone-00000014", "Local/201@extensions") in new stack
- Called 201@extensions
```

Now the third step in our dialplan executes the Dial() application which calls extension 201 in the [extensions] context of our dialplan. There is no requirement that we use the same extension number - we could have just as easily used a named extension, or some other number. Remember that we're dialing another channel, but instead of dialing a device, we're "dialing" another part of the dialplan.

```
- Executing [201@extensions:1] Verbose("Local/201@extensions-7cf4;2", "2,Made it to the Local
channel") in new stack == Made it to the Local channel
```

Now we've verified we've dialed another part of the dialplan. We can see the channel executing the dialplan has changed to Local/201@extensions-7cf4;2. The part '-7cf4;2' is just the unique identifier, and will be different for you.

```
- Executing [201@extensions:2] Verbose("Local/201@extensions-7cf4;2", "2,Inside channel:
Local/201@extensions-7cf4;2") in new stack
== Inside channel: Local/201@extensions-7cf4;2
```

Here we use the Verbose() application to see what our current channel name is. As you can see the current channel is a Local channel which we created from our SIP channel.

```
- Executing [201@extensions:3] Dial("Local/201@extensions-7cf4;2", "SIP/some-named-extension,30") in new stack
```

And from here, we're using another Dial() application to call a SIP device configured in sip.conf as [some-named-extension].

Now that we understand a simple example of calling the Local channel, let's expand upon this example by using Local channels to call two devices at the same time, but delay calling one of the devices.



## Using Callfiles and Local Channels

Another example is to use callfiles and Local channels so that you can execute some dialplan prior to performing a Dial(). We'll construct a callfile which will then utilize a Local channel to lookup a bit of information in the AstDB and then place a call via the channel configured in the AstDB.

First, lets construct our callfile that will use the Local channel to do some lookups prior to placing our call. More information on constructing callfiles is located in the doc/callfiles.txt file of your Asterisk source.

Our callfile will simply look like the following:

```
Channel: Local/201@devices
Application: Playback
Data: silence/l&tt-weasels
```

Add the callfile information to a file such as 'callfile.new' or some other appropriately named file.

Our dialplan will perform a lookup in the AstDB to determine which device to call, and will then call the device, and upon answer, Playback() the silence/1 (1 second of silence) and the tt-weasels sound files.

Before looking at our dialplan, lets put some data into AstDB that we can then lookup from the dialplan. From the Asterisk CLI, run the following command:

```
*CLI> database put phones 201/device SIP/0004f2040001
```

We've now put the device destination (SIP/0004f2040001) into the 201/device key within the phones family. This will allow us to lookup the device location for extension 201 from the database.

We can then verify our entry in the database using the 'database show' CLI command:

```
*CLI> database show /phones/201/device : SIP/0004f2040001
```

Now lets create the dialplan that will allow us to call SIP/0004f2040001 when we request extension 201 from the extensions context via our Local channel.

```
[devices]
exten => 201,1,NoOp( )
exten => 201,n,Set(DEVICE=${DB(phones/${EXTEN}/device)})
exten => 201,n,GotoIf($[${ISNULL}(${DEVICE})]?hangup) ; if nothing returned,
                                                    ; then hangup
exten => 201,n,Dial(${DEVICE},30)
exten => 201,n(hangup(),Hangup())
```

Then, we can perform a call to our device using the callfile by moving it into the /var/spool/asterisk/outgoing/ directory.

```
mv callfile.new /var/spool/asterisks/outgoing*
```

Then after a moment, you should see output on your console similar to the following, and your device ringing. Information about what is going on during the output has also been added throughout.

```
- Attempting call on Local/201@devices for application Playback(silence/l&tt-weasels) (Retry 1)
```

You'll see the line above as soon as Asterisk gets the request from the callfile.

```
- Executing [201@devices:1] NoOp("Local/201@devices-ecf0:2", "") in new stack
- Executing [201@devices:2] Set("Local/201@devices-ecf0:2", "DEVICE=SIP/0004f2040001") in new stack
```

This is where we performed our lookup in the AstDB. The value of SIP/0004f2040001 was then returned and saved to the DEVICE channel variable.

```
- Executing [201@devices:3] GotoIf("Local/201@devices-ecf0:2", "0?hangup") in new stack
```

We perform a check to make sure \${DEVICE} isn't NULL. If it is, we'll just hangup here.

```
- Executing [201@devices:4] Dial("Local/201@devices-ecf0:2", "SIP/0004f2040001,30") in new stack
- Called 000f2040001
- SIP/0004f2040001-00000022 is ringing
```

Now we call our device SIP/0004f2040001 from the Local channel.

```
SIP/0004f2040001-00000022 answered Local/201@devices-ecf0:2*
```

We answer the call.

```
> Channel Local/201@devices-ecf0;1 was answered.  
> Launching Playback(silence/1&tt-weasels) on Local/201@devices-ecf0;1
```

We then start playing back the files.

```
- <Local/201@devices-ecf0;1> Playing 'silence/1.slin' (language 'en')  
== Spawn extension (devices, 201, 4) exited non-zero on 'Local/201@devices-ecf0;2'
```

At this point we now see the Local channel has been optimized out of the call path. This is important as we'll see in examples later. By default, the Local channel will try to optimize itself out of the call path as soon as it can. Now that the call has been established and audio is flowing, it gets out of the way.

```
- <SIP/0004f2040001-00000022> Playing 'tt-weasels.ulaw' (language 'en')  
[Mar 1 13:35:23] NOTICE[16814]: pbx_spool.c:349 attempt_thread: Call completed to Local/201@devices
```

We can now see the tt-weasels file is played directly to the destination (instead of through the Local channel which was optimized out of the call path) and then a NOTICE stating the call was completed.

# Local Channel Optimization

## Default Channel Optimization

By default, the [Local channel](#) will try to optimize itself out of the call path. This means that once the Local channel has established the call between the destination and Asterisk, the Local channel will get out of the way and let Asterisk and the end point talk directly, instead of flowing through the Local channel.

This can have some adverse effects when you're expecting information to be available during the call that gets associated with the Local channel. When the Local channel is optimized out of the call path, any `Dial()` flags, or channel variables associated with the Local channel are also destroyed and are no longer available to Asterisk.

Diagrams really help to show what is going on:

### ***Figure 1***

This is a call in an unanswered (ringing) state - from SIP to SIP using Local Channels in between.

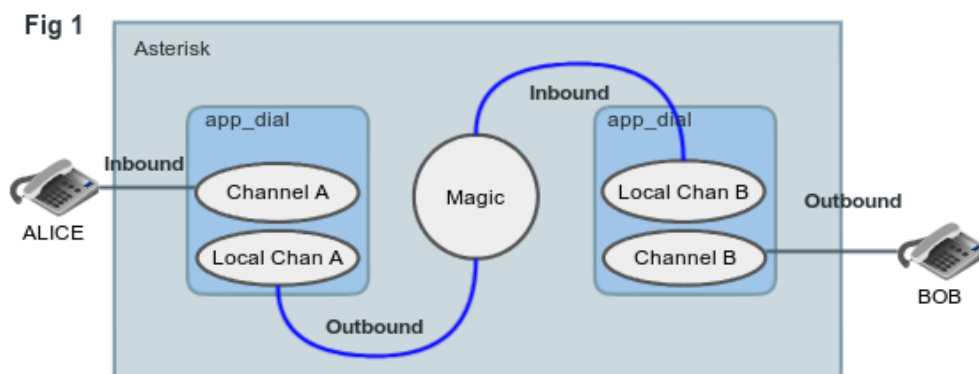
### ***Figure 2***

By default, after the callee answers this is what the call would look like with the Local Channels optimizing out.

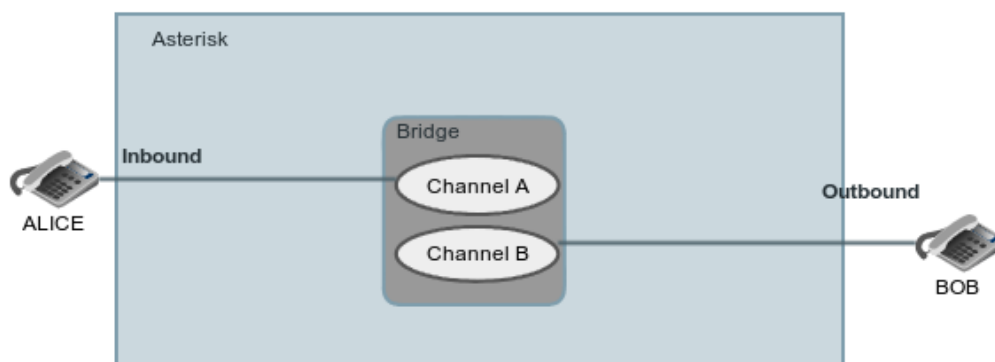
### ***Figure 3***

This is what the call would look like when established if you called the Local Channel with `/n`. You can see the Local Channels get pushed into bridges with channels they were connected with through `app_dial` previously.

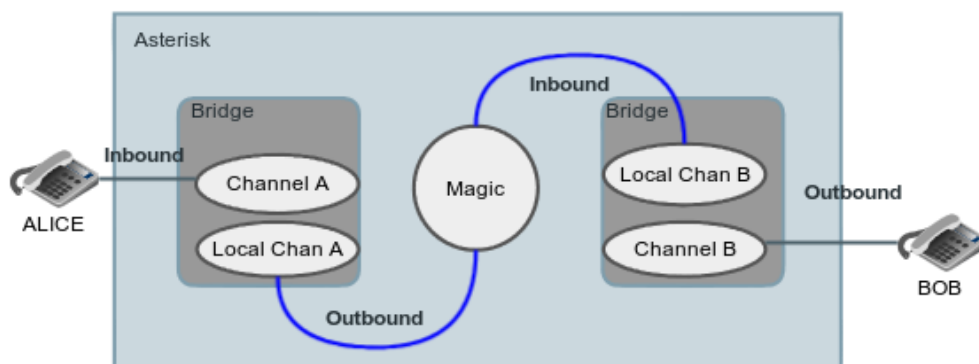
### Call SIP to SIP via Local Channels (unanswered state)



**Fig 2** Call established with Local Channels optimized out



**Fig 3** Call established with non-optimized Local Channels



## Disabling Local Channel Optimization

You may have read about the `/n` modifier in [Local Channel Modifiers](#). We can force the Local channel to remain in the call path by utilizing the `/n` directive. By adding `/n` to the end of the channel dial-string, we can keep the Local channel in the call path, along with any channel variables, or other channel specific information.

### When to disable optimization

Lets take a look at an example that demonstrates when the use of the `/n` directive is necessary. If we spawn a Local channel which does a `Dial()` to a SIP channel, but we use the `L()` option (which is used to limit the amount of time a call can be active, along with warning tones when the time is nearly up), it will be associated with the Local channel, which is then optimized out of the call path, and thus won't perform as expected.

This following dialplan will not perform as expected.

```
[services]
exten => 2,1,Dial(SIP/PHONE_B,,L(60000:45000:15000))

[internal]
exten => 4,1,Dial(Local/2@services)
```

In order to make this behave as we expect (limiting the call), we would change:

```
[internal]
exten => 4,1,Dial(Local/2@services)
```

...into the following:

```
[internal]
exten => 4,1,Dial(Local/2@services/n)
```

By adding `/n` to the end of the dial-string, our Local channel will now stay in the call path and not go away.

### **Detailed walk-through of example call-flow**

Why does adding the `/n` option all of a sudden make the 'L' option work? First we need to show an overview of the call flow that doesn't work properly, and discuss the information associated with the channels:

1. SIP device PHONE\_A calls Asterisk via a SIP INVITE.
2. Asterisk accepts the INVITE and then starts processing dialplan logic in the `[internal]` context.
3. Our dialplan calls `Dial(Local/2@services)` - notice the lack of the `"n"`.
4. The Local channel then executes dialplan at extension 2 within the `[services]` context.
5. Extension 2 within `[services]` then performs `Dial()` to PHONE\_B with the line: `Dial(SIP/PHONE_B,,L(60000:45000:15000))`
6. SIP/PHONE\_B then answers the call.
7. Even though the L option was given when dialing the SIP device, the L information is stored in the channel that is doing the `Dial()` which is the Local channel, and not the endpoint SIP channel.
8. The Local channel in the middle, containing the information for tracking the time allowance of the call, is then optimized out of the call path, losing all information about when to terminate the call.
9. SIP/PHONE\_A and SIP/PHONE\_B then continue talking indefinitely.

Now, if we were to add `/n` to our dialplan at step three (3) then we would force the Local channel to stay in the call path, and the `L()` option associated with the `Dial()` from the Local channel would remain, and our warning sounds and timing would work as expected.

There are two workarounds for the above described scenario:

1. Use what we just described, `Dial(Local/2@services/n)` to cause the Local channel to remain in the call path so that the `L()` option used inside the Local channel is not discarded when optimization is performed.
2. Place the `L()` option at the outermost part of the path so that when the middle is optimized out of the call path, the information required to make `L()` work is associated with the outside channel. The L information will then be stored on the calling channel, which is PHONE\_A. For example:

```
[services]
exten => 2,1,Dial(SIP/PHONE_B)

[internal]
exten => 4,1,Dial(Local/2@services,,L(60000:45000:15000));
```

# Local Channel Modifiers

## Usage

Dial-string modifiers exist that allow changing the default behavior of a Local Channel.

The modifiers are added to a channel by adding a slash followed by a flag onto the end of the Local Channel dial-string.

For example below we are adding the "n" modifier to the dial-string.

```
Local/101@mycontext/n
```

You can add more than one modifier by adding them directly adjacent to the previous modifier.

```
Local/101@mycontext/nj
```

## List of Modifiers

- 'n' - Instructs the Local channel to not do a native transfer (the "n" stands for *No release*) upon the remote end answering the line. This is an esoteric, but important feature if you expect the Local channel to handle calls exactly like a normal channel. If you do not have the "no release" feature set, then as soon as the destination (inside of the Local channel) answers the line and one audio frame passes, the variables and dial plan will revert back to that of the original call, and the Local channel will become a *zombie* and be removed from the active channels list. This is desirable in some circumstances, but can result in unexpected dialplan behavior if you are doing fancy things with variables in your call handling. Read about [Local Channel Optimization](#) to better understand when this option is necessary.
- 'j' - Allows you to use the generic jitterbuffer on incoming calls going to Asterisk applications. For example, this would allow you to use a jitterbuffer for an incoming SIP call to Voicemail by putting a Local channel in the middle. The 'j' option must be used in conjunction with the 'n' option to make sure that the Local channel does not get optimized out of the call. This option is available starting in the Asterisk 1.6.0 branch.
- 'm' - Will cause the Local channel to forward music on hold (MoH) start and stop requests. Normally the Local channel acts on them and it is started or stopped on the Local channel itself. This options allows those requests to be forwarded through the Local channel. This option is available starting in the Asterisk 1.4 branch.
- 'b' - This option causes the Local channel to return the actual channel that is behind it when queried. This is useful for transfer scenarios as the actual channel will be transferred, not the Local channel. This option is available starting in the Asterisk 1.6.0 branch and was removed in Asterisk 12.

## Motif



Under Construction



Page for information on the Motif channel driver, describing configuration, pointing to any resources and a top-level page for any examples or tutorials such as calling with Google Voice.

## Calling using Google

- Prerequisites
  - RTP configuration
  - Motif configuration
    - Example Motif Configuration
  - XMPP Configuration
    - Example XMPP Configuration
    - More about Priorities
  - Phone configuration
  - Dialplan configuration
    - Incoming calls
    - Outgoing calls



This new page replaces the [old page](#). The old page documents behavior that is not functional or supported going forward. This new page documents behavior as of Asterisk 11. For more information, please see the blog posting <http://blogs.digium.com/2012/07/24/asterisk-11-development-the-motive-for-motif/>

### Prerequisites

Asterisk communicates with Google Voice and Google Talk using the `chan_motif` Channel Driver and the `res_xmpp` Resource module. Before proceeding, please ensure that both are compiled and part of your installation. Compilation of `res_xmpp` and `chan_motif` for use with Google Talk / Voice are dependant on the `iksemel` library files as well as the OpenSSL development libraries presence on your system.

Calling using Google Voice or via the Google Talk web client requires the use of Asterisk 11.0 or greater. Older versions of Asterisk will not work.

For basic calling between Google Talk web clients, you need a Google Mail account.

For calling to and from the PSTN, you will need a Google Voice account.

In your Google account, you'll want to change the Chat setting from the default of "Automatically allow people that I communicate with often to chat with me and see when I'm online" to the second option of "Only allow people that I've explicitly approved to chat with me and see when I'm online."

IPv6 is currently not supported. Use of IPv4 is required.

Google Voice can now be used with Google Apps accounts.

### RTP configuration

ICE support is required for `chan_motif` to operate. It is disabled by default and must be explicitly enabled in the RTP configuration file `rtp.conf` as follows.

```
[general]
icesupport=yes
```

If this option is not enabled you will receive the following error message.

```
Unable to add Google ICE candidates as ICE support not available or no candidates available
```

### Motif configuration

The Motif channel driver is configured with the `motif.conf` configuration file, typically located in `/etc/asterisk`. What follows is an example configuration for successful operation.

#### Example Motif Configuration

```
[google]
context=incoming-motif
disallow=all
allow=ulaw
connection=google
```

This general section of this configuration specifies several items.

1. That calls will terminate to or originate from the **incoming-motif** context; `context=incoming-motif`
2. That all codecs are first explicitly disallowed
3. That G.711 ulaw is allowed
4. The an XMPP connection called "google" is to be used

Google lists supported audio codecs on this page - [https://developers.google.com/talk/open\\_communications](https://developers.google.com/talk/open_communications)

Per section, 5, the supported codecs are:



1. PCMA
2. PCMU
3. G.722
4. GSM
5. iLBC
6. Speex

Our experience shows this not to be the case. Rather, the codecs, supported by Asterisk, and seen in an invite from Google Chat are:

1. PCMA
2. PCMU
3. G.722
4. iLBC
5. Speex 16kHz
6. Speex 8kHz

It should be noted that calling using Google Voice requires the G.711 ulaw codec. So, if you want to make sure Google Voice calls work, allow G.711 ulaw, at a minimum.

## XMPP Configuration

The `res_xmpp` Resource is configured with the `xmpp.conf` configuration file, typically located in `/etc/asterisk`. What follows is an example configuration for successful operation.

### Example XMPP Configuration

```
[general]
[google]
type=client
serverhost=talk.google.com
username=example@gmail.com
secret=examplepassword
priority=25
port=5222
usetls=yes
usesasl=yes
status=available
statusmessage="I am available"
timeout=5
```

The default general section does not need any modification.

The google section of this configuration specifies several items.

1. The type is set to client, as we're connecting to Google as a service; `type=client`
2. The serverhost is Google's talk server; `serverhost=talk.google.com`
3. Our username is configured as `your_google_username@gmail.com`; `username=your_google_username@gmail.com`
4. Our password is configured using the secret option; `secret=your_google_password`
5. Google's talk service operates on port 5222; `port=5222`
6. Our priority is set to 25; `priority=25`
7. TLS encryption is required by Google; `usetls=yes`
8. Simple Authentication and Security Layer (SASL) is used by Google; `usesasl=yes`
9. We set a status message so other Google chat users can see that we're an Asterisk server; `statusmessage="I am available"`
10. We set a timeout for receiving message from Google that allows for plenty of time in the event of network delay; `timeout=5`

### More about Priorities

As many different connections to Google are possible simultaneously via different client mechanisms, it is important to understand the role of priorities in the routing of inbound calls. Proper usage of the priority setting can allow use of a Google account that is not otherwise entirely dedicated to voice services.

With priorities, the higher the setting value, the more any client using that value is preferred as a destination for inbound calls, in deference to any other client with a lower priority value. Known values of commonly used clients include the Gmail chat client, which maintains a priority of **20**, and the Windows GTalk client, which uses a priority of **24**. The maximum allowable value is **127**. Thus, setting one's **priority** option for the XMPP peer in `res_xmpp.conf` to a value higher than 24 will cause inbound calls to flow to Asterisk, even while one is logged into either Gmail or the Windows GTalk client.

Outbound calls are unaffected by the priority setting.

## Phone configuration

Now, let's create a phone. The configuration of a SIP device for this purpose would, in `sip.conf`, typically located in `/etc/asterisk`, look something like:

```
[malcolm]
type=peer
secret=my_secure_password
host=dynamic
context=local
```

## Dialplan configuration

### Incoming calls

Next, let's configure our dialplan to receive an incoming call from Google and route it to the SIP phone we created. To do this, our dialplan, `extensions.conf`, typically located in `/etc/asterisk`, would look like:

```
[incoming-motif]
exten => s,1,NoOp()
same => n,Wait(1)
same => n,Answer()
same => n,SendDTMF(1)
same => n,Dial(SIP/malcolm,20)
```



Did you know that the Google Chat client does this same thing; it waits, and then sends a DTMF 1. Really.

This example uses the "s" unmatched extension, because we're only configuring one client connection in this example.

In this example, we're Waiting 1 second, answering the call, sending the DTMF "1" back to Google, and **then** dialing the call. We do this, because inbound calls from Google enable, even if it's disabled in your Google Voice control panel, call screening. Without this SendDTMF event, you'll have to confirm with Google whether or not you want to answer the call.



#### Using Google's voicemail

Another method for accomplishing the sending of the DTMF event is to use Dial option "D." The D option tells Asterisk to send a specified DTMF string after the called party has answered. DTMF events specified before a colon are sent to the **called** party. DTMF events specified after a colon are sent to the **calling** party.

In this example then, one does not need to actually answer the call first, though one should still wait at least a second for things, like STUN setup, to finish. This means that if the called party doesn't answer, Google will resort to sending the call to one's Google Voice voicemail box, instead of leaving it at Asterisk.

```
exten => s,1,Dial(SIP/malcolm,20,D(:1))
```



#### Filtering Caller ID

The inbound CallerID from Google is going to look a bit nasty, e.g.:

```
+15555551212@voice.google.com/srvres-MTAuMjE4LjIuMTk3Ojk4MzM=
```

Your VoIP client (SIPDroid) might not like this, so let's simplify that Caller ID a bit, and make it more presentable for your phone's display. Here's the example that we'll step through:

```
exten => s,1,NoOp()
same => n,Set(crazygooglecid=${CALLERID(name)})
same => n,Set(stripcrazysuffix=${CUT(crazygooglecid,@,1)})
same => n,Set(CALLERID(all)=${stripcrazysuffix})
same => n,Dial(SIP/malcolm,20,D(:1))
```

First, we set a variable called **crazygooglecid** to be equal to the name field of the CALLERID function. Next, we use the CUT function to grab everything that's before the @ symbol, and save it in a new variable called **stripcrazysuffix**. We'll set this new variable to the CALLERID that we're going to use for our Dial. Finally, we'll actually Dial our internal destination.

### Outgoing calls

Outgoing calls to Google Talk users take the form of:

```
exten => 100,1,Dial(Motif/google/mybuddy@gmail.com,,r)
```

Where the technology is "Motif," the dialing peer is "google" as defined in `xmpp.conf`, and the dial string is the Google account name.

We use the Dial option "r" because Google doesn't provide ringing indications.

Outgoing calls made to Google Voice take the form of:

```
exten => _1XXXXXXXXXX,1,Dial(Motif/google/${EXTEN}@voice.google.com,,r)
```

Where the technology is "Motif," the dialing peer is "google" as defined in motif.conf, and the dial string is a full E.164 number, sans the plus character.

Again, we use Dial option "r" because Google doesn't provide ringing indications.

## mISDN

## **Introduction to mISDN**

This package contains the mISDN Channel Driver for the Asterisk PBX. It supports every mISDN Hardware and provides an interface for Asterisk.

## mISDN Features

- NT and TE mode
- PP and PMP mode
- BRI and PRI (with BNE1 and BN2E1 Cards)
- Hardware bridging
- DTMF detection in HW+mISDNdsp
- Display messages on phones (on those that support it)
- app\_SendText
- HOLD/RETRIEVE/TRANSFER on ISDN phones : )
- Allow/restrict user number presentation
- Volume control
- Crypting with mISDNdsp (Blowfish)
- Data (HDLC) callthrough
- Data calling (with app\_ptyfork +pppd)
- Echo cancellation
- Call deflection
- Some others

## mISDN Fast Installation Guide

It is easy to install mISDN and mISDNuser. This can be done by:

You can download latest stable releases from <http://www.misdn.org/downloads/>

Just fetch the newest head of the GIT (mISDN project moved from CVS) In details this process described here: <http://www.misdn.org/index.php/GIT> then compile and install both with:

```
cd mISDN ; make && make install
```

(you will need at least your kernel headers to compile mISDN).

```
cd mISDNuser ; make && make install
```

Now you can compile chan\_misdn, just by making Asterisk:

```
cd asterisk ; ./configure && make && make install
```

That's all!

Follow the instructions in the mISDN Package for how to load the Kernel Modules. Also install process described in [http://www.misdn.org/index.php/Installing\\_mISDN](http://www.misdn.org/index.php/Installing_mISDN)

## **mISDN Pre-Requisites**

To compile and install this driver, you'll need at least one mISDN Driver and the mISDNuser package. Chan\_misdn works with both, the current release version and the development (svn trunk) version of Asterisk.

You should use Kernels = 2.6.9



## mISDN Configuration

First of all you must configure the mISDN drivers, please follow the instructions in the mISDN package to do that, the main config file and config script is:

```
/etc/init.d/misdn-init and /etc/misdn-init.conf
```

Now you will want to configure the misdn.conf file which resides in the Asterisk config directory (normally /etc/asterisk).

### ***misdn.conf: [general] subsection***

The misdn.conf file contains a "general" subsection, and user subsections which contain misdn port settings and different Asterisk contexts. In the general subsection you can set options that are not directly port related. There is for example the very important debug variable which you can set from the Asterisk cli (command line interface) or in this configuration file, bigger numbers will lead to more debug output. There's also a trace file option, which takes a path+filename where debug output is written to.

### ***misdn.conf: [default] subsection***

The default subsection is another special subsection which can contain all the options available in the user/port subsections. The user/port subsections inherit their parameters from the default subsection.

### ***misdn.conf: user/port subsections***

The user subsections have names which are unequal to "general". Those subsections contain the ports variable which mean the mISDN Ports. Here you can add multiple ports, comma separated.

Especially for TE-Mode Ports there is a msns option. This option tells the chan\_misdn driver to listen for incoming calls with the given msns, you can insert a " " as single msn, which leads to getting every incoming call. If you want to share on PMP TE S0 with Asterisk and a phone or ISDN card you should insert here the msns which you assign to Asterisk. Finally a context variable resides in the user subsections, which tells chan\_misdn where to send incoming calls to in the Asterisk dial plan (extension.conf).\*

### ***Dial and Options String***

The dial string of chan\_misdn got more complex, because we added more features, so the generic dial string looks like:

```
mISDN/<port>[:bchannel] |g:<group>/<extension>[ /<OPTIONSSTRING> ]
```

The Optionsstring looks Like:

```
:<optchar><optarg>:<optchar><optarg>...
```

The ":" character is the delimiter. The available options are:

- a - Have Asterisk detect DTMF tones on called channel
- c - Make crypted outgoing call, optarg is keyindex
- d - Send display text to called phone, text is the optarg
- e - Perform echo cancelation on this channel, takes taps as optarg (32,64,128,256)
- e! - Disable echo cancelation on this channel
- f - Enable fax detection
- h - Make digital outgoing call
- h1 - Make HDLC mode digital outgoing call
- i - Ignore detected DTMF tones, don't signal them to Asterisk, they will be transported inband.
- jb - Set jitter buffer length, optarg is length
- jt - Set jitter buffer upper threshold, optarg is threshold
- jn - Disable jitter buffer
- n - Disable mISDN DSP on channel. Disables: echo cancel, DTMF detection, and volume control.
- p - Caller ID presentation, optarg is either 'allowed' or 'restricted'
- s - Send Non-inband DTMF as inband
- vr - Rx gain control, optarg is gain
- vt - Tx gain control, optarg is gain

chan\_misdn registers a new dial plan application "misdn\_set\_opt" when loaded. This application takes the Optionsstring as argument. The Syntax is:

```
misdn_set_opt (<OPTIONSSTRING> )
```

When you set options in the dialstring, the options are set in the external channel. When you set options with misdn\_set\_opt, they are set in the current incoming channel. So if you like to use static encryption, the scenario looks as follows:

```
Phone1 --> * Box 1 --> PSTN_TE PSTN_TE --> * Box 2 --> Phone2
```

The encryption must be done on the PSTN sides, so the dialplan on the boxes are:

- Box 1:

```
exten => _${CRYPT_PREFIX}X.,1,Dial(mISDN/g:outbound/:c1)
```

- Box 2:

```
exten => ${CRYPT_MSN},1,misdn_set_opt(:c1)  
exten => ${CRYPT_MSN},2,dial(${PHONE2})
```

## mISDN CLI Commands

At the Asterisk cli you can try to type in:

```
misdn <tab> <tab>
```

Now you should see the misdn cli commands:

- clean -> pid (cleans a broken call, use with care, leads often to a segmentation fault)
- send -> display (sends a Text Message to a Asterisk channel, this channel must be an misdn channel)
- set -> debug (sets debug level)
- show ->
  - config (shows the configuration options)
  - channels (shows the current active misdn channels)
  - channel (shows details about the given misdn channels)
  - stacks (shows the current ports, their protocols and states)
  - fullstacks (shows the current active and inactive misdn channels)
- restart -> port (restarts given port (L2 Restart) ) - reload (reloads misdn.conf)

You can only use "misdn send display" when an Asterisk channel is created and isdn is in the correct state. "correct state" means that you have established a call to another phone (must not be isdn though).

Then you use it like this:

```
misdn send display mISDN/1/101 "Hello World!"
```

where 1 is the Port of the Card where the phone is plugged in, and 101 is the msn (callerid) of the Phone to send the text to.

## **mISDN Variables**

mISDN Exports/Imports a few Variables:

- MISDN\_ADDRESS\_COMPLETE : Is either set to 1 from the Provider, or you can set it to 1 to force a sending complete.\*

## mISDN Debugging and Bug Reports

If you encounter problems, you should set up the debugging flag, usually `debug=2` should be enough. The messages are divided into Asterisk and mISDN parts. mISDN Debug messages begin with an `!`, Asterisk messages begin with an `"`, the rest is clear I think.\*

Please take a trace of the problem and open a report in the Asterisk issue tracker at <https://issues.asterisk.org> in the "channel drivers" project, "chan\_misdn" category. Read the bug guidelines to make sure you provide all the information needed.

## mISDN Examples

Here are some examples of how to use chan\_misdn in the dialplan (extensions.conf):

```
[globals]
OUT_PORT=1 ; The physical Port of the Card
OUT_GROUP=Externe1 ; The Group of Ports defined in misdn.conf

[misdnIn]
exten => _X.,1,Dial(mISDN/${OUT_PORT}/${EXTEN})
exten => _0X.,1,Dial(mISDN/g:${OUT_GROUP}/${EXTEN:1})
exten => _1X.,1,Dial(mISDN/g:${OUT_GROUP}/${EXTEN:1}/:dHello)
exten => _1X.,1,Dial(mISDN/g:${OUT_GROUP}/${EXTEN:1}/:dHello Test:n)
```

On the last line, you will notice the last argument (Hello); this is sent as Display Message to the Phone.

## mISDN Known Problems

- Q: I cannot hear any tone after a successful CONNECT to the other end.
- A: You forgot to load mISDNdsp, which is now needed by chan\_misdn for switching and DTMF tone detection.

## Mobile Channel

chan\_mobile pages



## Introduction to the Mobile Channel

Asterisk Channel Driver to allow Bluetooth Cell/Mobile Phones to be used as FXO devices, and Headsets as FXS devices.

## Mobile Channel Features

- Multiple Bluetooth Adapters supported.
- Multiple phones can be connected.
- Multiple headsets can be connected.
- Asterisk automatically connects to each configured mobile phone / headset when it comes in range.
- CLI command to discover bluetooth devices.
- Inbound calls on the mobile network to the mobile phones are handled by Asterisk, just like inbound calls on a Zap channel.
- CLI passed through on inbound calls.
- Dial outbound on a mobile phone using Dial(Mobile/device/nnnnnn) in the dialplan.
- Dial a headset using Dial(Mobile/device) in the dialplan.
- Application MobileStatus can be used in the dialplan to see if a mobile phone / headset is connected.
- Supports devicestate for dialplan hinting.
- Supports Inbound and Outbound SMS.
- Supports 'channel' groups for implementing 'GSM Gateways'

## Mobile Channel Requirements

In order to use `chan_mobile`, you must have a working bluetooth subsystem on your Asterisk box. This means one or more working bluetooth adapters, and the BlueZ packages.

Any bluetooth adapter supported by the Linux kernel will do, including usb bluetooth dongles.

The BlueZ package you need is `bluez-utils`. If you are using a GUI then you might want to install `bluez-pin` also. You also need `libbluetooth`, and `libbluetooth-dev` if you are compiling Asterisk from source.

You need to get bluetooth working with your phone before attempting to use `chan_mobile`. This means 'pairing' your phone or headset with your Asterisk box. I dont describe how to do this here as the process differs from distro to distro. You only need to pair once per adapter.

See <http://www.bluez.org> for details about setting up Bluetooth under Linux.

## Mobile Channel Concepts

chan\_mobile deals with both bluetooth adapters and bluetooth devices. This means you need to tell chan\_mobile about the bluetooth adapters installed in your server as well as the devices (phones / headsets) you wish to use.

chan\_mobile currently only allows one device (phone or headset) to be connected to an adapter at a time. This means you need one adapter for each device you wish to use simultaneously. Much effort has gone into trying to make multiple devices per adapter work, but in short it doesnt.

Periodically chan\_mobile looks at each configured adapter, and if it is not in use (i.e. no device connected) will initiate a search for devices configured to use this adapter that may be in range. If it finds one it will connect the device and it will be available for Asterisk to use. When the device goes out of range, chan\_mobile will disconnect the device and the adapter will become available for other devices.

## Configuring chan\_mobile

The configuration file for chan\_mobile is `/etc/asterisk/mobile.conf`. It is a normal Asterisk config file consisting of sections and key=value pairs.

See `configs/mobile.conf.sample` for an example and an explanation of the configuration.

## Using chan\_mobile

chan\_mobile.so must be loaded either by loading it using the Asterisk CLI, or by adding it to /etc/asterisk/modules.conf  
Search for your bluetooth devices using the CLI command 'mobile search'. Be patient with this command as it will take 8 - 10 seconds to do the discovery. This requires a free adapter.  
Headsets will generally have to be put into 'pairing' mode before they will show up here.  
This will return something like the following :-

```
*CLI> mobile search
Address Name Usable Type Port
00:12:56:90:6E:00 LG TU500 Yes Phone 4
00:80:C8:35:52:78 Toaster No Headset 0
00:0B:9E:11:74:A5 Hello II Plus Yes Headset 1
00:0F:86:0E:AE:42 Daves Blackberry Yes Phone 7
```

This is a list of all bluetooth devices seen and whether or not they are usable with chan\_mobile. The Address field contains the 'bd address' of the device. This is like an ethernet mac address. The Name field is whatever is configured into the device as its name. The Usable field tells you whether or not the device supports the Bluetooth Handsfree Profile or Headset profile. The Type field tells you whether the device is usable as a Phone line (FXO) or a headset (FXS) The Port field is the number to put in the configuration file.

Choose which device(s) you want to use and edit /etc/asterisk/mobile.conf. There is a sample included with the Asterisk-addons source under configs/mobile.conf.sample.

Be sure to configure the right bd address and port number from the search. If you want inbound calls on a device to go to a specific context, add a context= line, otherwise the default will be used. The 'id' of the device [bitinbrackets] can be anything you like, just make it unique.

If you are configuring a Headset be sure to include the type=headset line, if left out it defaults to phone.

The CLI command 'mobile show devices' can be used at any time to show the status of configured devices, and whether or not the device is capable of sending / receiving SMS via bluetooth.

```
*CLI> mobile show devices
ID Address Group Adapter Connected State SMS
headset 00:0B:9E:11:AE:C6 0 blue No Init No
LGTU550 00:E0:91:7F:46:44 1 dlink No Init No
```

As each phone is connected you will see a message on the Asterisk console :-

```
Loaded chan_mobile.so => (Bluetooth Mobile Device Channel Driver)
- Bluetooth Device blackberry has connected.
- Bluetooth Device dave has connected.
```

To make outbound calls, add something to you Dialplan like the following :- (modify to suit)

```
; Calls via LGTU5500
exten => _9X.,1,Dial(Mobile/LGTU550/${EXTEN:1},45)
exten => _9X.,n,Hangup
```

To use channel groups, add an entry to each phones definition in mobile.conf like group=n where n is a number.

Then if you do something like Dial(Mobile/g1/123456) Asterisk will dial 123456 on the first connected free phone in group 1.

Phones which do not have a specific 'group=n' will be in group 0.

To dial out on a headset, you need to use some other mechanism, because the headset is not likely to have all the needed buttons on it. res\_clioriginate is good for this :-

```
*CLI> originate Mobile/headset extension NNNNN@context
```

This will call your headset, once you answer, Asterisk will call NNNNN at context context

## Mobile Channel Dialplan Hints

chan\_mobile supports 'device status' so you can do something like

```
exten => 1234, hint, SIP/30&Mobile/dave&Mobile/blackberry
```

## MobileStatus Application

chan\_mobile also registers an application named MobileStatus. You can use this in your Dialplan to determine the 'state' of a device. For example, suppose you wanted to call dave's extension, but only if he was in the office. You could test to see if his mobile phone was attached to Asterisk, if it is dial his extension, otherwise dial his mobile phone.

```
exten => 40,1,MobileStatus(dave,DAVECELL)
exten => 40,2,GotoIf(${ "${DAVECELL}" = "1" ]?3:5)
exten => 40,3,Dial(ZAP/g1/0427466412,45,tT)
exten => 40,4,Hangup
exten => 40,5,Dial(SIP/40,45,tT)
exten => 40,6,Hangup
```

MobileStatus sets the value of the given variable to :-

- 1 = Disconnected. i.e. Device not in range of Asterisk, or turned off etc etc
- 2 = Connected and Not on a call. i.e. Free
- 3 = Connected and on a call. i.e. Busy



## Mobile Channel DTMF Debouncing

DTMF detection varies from phone to phone. There is a configuration variable that allows you to tune this to your needs. e.g. in mobile.conf

```
[LGTU550]
address=00:12:56:90:6E:00
port=4
context=incoming-mobile
dtmfskip=50
```

change dtmfskip to suit your phone. The default is 200. The larger the number, the more chance of missed DTMF. The smaller the number the more chance of multiple digits being detected.

## Mobile Channel SMS Sending and Receiving

If Asterisk has detected your mobile phone is capable of SMS via bluetooth, you will be able to send and receive SMS.

Incoming SMS's cause Asterisk to create an inbound call to the context you defined in mobile.conf or the default context if you did not define one. The call will start at extension 'sms'. Two channel variables will be available, SMSSRC = the number of the originator of the SMS and SMSTXT which is the text of the SMS. This is not a voice call, so grab the values of the variables and hang the call up.

So, to handle incoming SMS's, do something like the following in your dialplan

```
[incoming-mobile]
exten => sms,1,Verbose(Incoming SMS from ${SMSSRC} ${SMSTXT})
exten => sms,n,Hangup()
```

The above will just print the message on the console.

If you use res\_jabber, you could do something like this :-

```
[incoming-mobile]
exten => sms,1,JabberSend(transport,user@jabber.somewhere.com,SMS from ${SMSSRC}
${SMSTXT})
exten => sms,2,Hangup()
```

To send an SMS, use the application MobileSendSMS like the following :-

```
exten => 99,1,MobileSendSMS(dave,0427123456,Hello World)
```

This will send 'Hello World' via device 'dave' to '0427123456'

## Mobile Channel Debugging

Different phone manufacturers have different interpretations of the Bluetooth Handsfree Profile Spec. This means that not all phones work the same way, particularly in the connection setup / initialisation sequence. I've tried to make chan\_mobile as general as possible, but it may need modification to support some phone i've never tested.

Some phones, most notably Sony Ericsson 'T' series, dont quite conform to the Bluetooth HFP spec. chan\_mobile will detect these and adapt accordingly. The T-610 and T-630 have been tested and work fine.

If your phone doesnt behave has expected, turn on Asterisk debugging with 'core set debug 1'.

This will log a bunch of debug messages indicating what the phone is doing, importantly the rfcmm conversation between Asterisk and the phone. This can be used to sort out what your phone is doing and make chan\_mobile support it.

Be aware also, that just about all mobile phones behave differently. For example my LG TU500 wont dial unless the phone is a the 'idle' screen. i.e. if the phone is showing a 'menu' on the display, when you dial via Asterisk, the call will not work. chan\_mobile handles this, but there may be other phones that do other things too...

Important: Watch what your mobile phone is doing the first few times. Asterisk wont make random calls but if chan\_mobile fails to hangup for some reason and you get a huge bill from your telco, dont blame me

# Unistim

# Introduction to the Unistim channel

## Unified Networks IP Stimulus (UNIStim) Channel Driver for Asterisk

This is a channel driver for Unistim protocol. You can use at least a Nortel i2002, i2004 and i2050.

Following features are supported : Send/Receive CallerID, Redial, SoftKeys, SendText(), Music On Hold, Message Waiting Indication (MWI), Distinctive ring, Transfer, Threeway call, History, Forward, Dynamic SoftKeys.

### How to configure the i2004

1. Power on the phone
2. Wait for message "Nortel Networks"
3. Press quickly the four buttons just below the LCD screen, in sequence from left to right
4. If you see "Locating server", power off or reboot the phone and try again
5. DHCP : 0
6. SET IP : a free ip of your network
7. NETMSK / DEF GW : netmask and default gateway
8. S1 IP : ip of the asterisk server
9. S1 PORT : 5000
10. S1 ACTION : 1
11. S1 RETRY COUNT : 10
12. S2 : same as S1

### How to place a call

The line=> entry in unistim.conf does not add an extension in asterisk by default. If you want to do that, add extension=line in your phone context.

If you have this entry on unistim.conf :

```
[violet]
device=006038abcdef
line => 102
```

then use:

```
exten => 2100,1,Dial(USTM/102@violet)
```

You can display a text with :

```
exten => 555,1,SendText(Sends text to client. Greetings)
```

### Rebooting a Nortel phone

- Press mute,up,down,up,down,up,mute,9,release(red button)

### Distinctive ring

1. You need to append /r to the dial string.
2. The first digit must be from 0 to 7 (inclusive). It's the 'melody' selection.
3. The second digit (optional) must be from 0 to 3 (inclusive). It's the ring volume. 0 still produce a sound.

Select the ring style #1 and the default volume :

```
exten => 2100,1,Dial(USTM/102@violet/r1)
```

Select the ring style #4 with a very loud volume :

```
exten => 2100,1,Dial(USTM/102@violet/r43)
```

### Country code

- You can use the following codes for country= (used for dial tone) - us fr au nl uk fi es jp no at nz tw cl se be sg il br hu lt pl za pt ee mx in de ch dk cn
- If you want a correct ring, busy and congestion tone, you also need a valid entry in indications.conf and check if res\_indications.so is loaded.
- language= is also supported but it's only used by Asterisk (for more information see <http://www.voip-info.org/wiki/view/Asterisk+multi-language> ). The end user interface of the phone will stay in english.

### Bookmarks, Softkeys

#### Layout

	-----	
	5            2	
	4            1	
	3            0	

- When the second letter of bookmark= is @, then the first character is used for positioning this entry
- If this option is omitted, the bookmark will be added to the next available sofkey
- Also work for linelabel (example : linelabel="5@Line 123")
- You can change a softkey programmatically with SendText(@position@icon@label@extension) ex: SendText(@1@55@Stop Forwd@908)

### Autoprovisioning

- This feature must only be used on a trusted network. It's very insecure : all unistim phones will be able to use your asterisk pbx.
- You must add an entry called **template**. Each new phones will be based on this profile.
- You must set a least line=>. This value will be incremented when a new phone is registered. device= must not be specified. By default, the phone will asks for a number. It will be added into the dialplan. Add extension=line for using the generated line number instead.

Example :

```
[general]
port=5000
autoprovisioning=yes

[template]
line => 100
bookmark=Support@123 ; Every phone will have a softkey Support
```

- If a first phone have a mac = 006038abcdef, a new device named USTM/100@006038abcdef will be created.
- If a second phone have a mac = 006038000000, it will be named USTM/101@006038000000 and so on.
- When autoprovisioning=tn, new phones will ask for a tn, if this number match a tn= entry in a device, this phone will be mapped into.

Example:

```
[black]
tn=1234
line => 100
```

- If a user enter TN 1234, the phone will be known as USTM/100@black.

### History

- Use the two keys located in the middle of the Fixed feature keys row (on the bottom of the phone) to enter call history.
- By default, chan\_unistim add any incoming and outgoing calls in files (/var/log/asterisk/unistimHistory). It can be a privacy issue, you can disable this feature by adding callhistory=0. If history files were created, you also need to delete them. callhistory=0 will NOT disable normal asterisk CDR logs.

### Forward

- This feature requires chan\_local (loaded by default)

### Generic asterisk features

You can use the following entries in unistim.conf

- Billing - accountcode amaflags
- Call Group - callgroup pickupgroup (untested)
- Music On Hold - musiconhold
- Language - language (see section Country Code)
- RTP NAT - nat (control ast\_rtp\_setnat, default = 0. Obscure behaviour)

### **Trunking**

- It's not possible to connect a Nortel Succession/Meridian/BCM to Asterisk via chan\_unistim. Use either E1/T1 trunks, or buy UTPS (UNISTIM Terminal Proxy Server) from Nortel.

### **Wiki, Additional infos, Comments :**

- <http://www.voip-info.org/wiki-Asterisk+UNISTIM+channels>

### **\*BSD :**

- Comment #define HAVE\_IP\_PKTINFO in chan\_unistim.c
- Set public\_ip with an IP of your computer
- Check if unistim.conf is in the correct directory

### **Issues**

- As always, NAT can be tricky. If a phone is behind a NAT, you should port forward UDP 5000 (or change `general port=` in unistim.conf) and UDP 10000 (or change `yourphone rtp_port=`)
- Only one phone per public IP (multiple phones behind the same NAT don't work). You can either :
  - Setup a VPN
  - Install asterisk inside your NAT. You can use IAX2 trunking if you're master asterisk is outside.
  - If asterisk is behind a NAT, you must set `general public_ip=` with your public IP. If you don't do that or the bindaddr is invalid (or no longer valid, eg dynamic IP), phones should be able to display messages but will be unable to send/receive RTP packets (no sound)
- Don't forget : this work is based entirely on a reverse engineering, so you may encounter compatibility issues. At this time, I know three ways to establish a RTP session. You can modify `yourphone rtp_method=` with 0, 1, 2 or 3. 0 is the default method, should work. 1 can be used on new firmware (black i2004) and 2 on old violet i2004. 3 can be used on black i2004 with chrome.
- If you have difficulties, try unistim debug and set verbose 3 on the asterisk CLI. For extra debug, uncomment #define DUMP\_PACKET 1 and recompile chan\_unistim.

## Protocol information

### Protocol versions

31 October 2008

UNISlim Firmware Release 3.1 for IP Phones, includes:

- 0604DCG for Phase II IP Phones (2001, 2002 2004),
- 0621C6H for IP Phone 2007,
- 0623C6J, 0624C6J, 0625C6J and 0627C6J for IP Phone 1110, 1120E,1140E and 1150E respectively
- 062AC6J for IP Phone 1210, 1220, and 1230

27 February 2009

UNISlim Firmware Release 3.2 for IP Phones, including:

- 0604DCJ for Phase II IP Phones (2001, 2002 & 2004),
- 0621C6M for IP Phone 2007,
- 0623C6N, 0624C6N, 0625C6N and 0627C6N for IP Phone 1110, 1120E,1140E and 1150E respectively
- 062AC6N for IP Phone 1210, 1220, and 1230

30 June 2009

UNISlim Firmware Release 3.3 for IP Phones:

- 0604DCL for Phase II IP Phones (2001, 2002 & 2004),
- 0621C6P for IP Phone 2007,
- 0623C6R, 0624C6R, 0625C6R and 0627C6R for IP Phone 1110, 1120E,1140E and 1150E respectively
- 062AC6R for IP Phone 1210, 1220, and 1230

27 November 2009

UNISlim Software Release 4.0 for IP Phones, includes:

- 0621C7A for IP Phone 2007,
- 0623C7F, 0624C7F, 0625C7F and 0627C7F for IP Phone 1110, 1120E,1140E and 1150E respectively
- 062AC7F for IP Phone 1210, 1220, and 1230

28 February 2010

UNISlim Software Release 4.1 IP Deskphone Software

- 0621C7D / 2007 IP Deskphone
- 0623C7J / 1110 IP Deskphone
- 0624C7J / 1120E IP Deskphone
- 0625C7J / 1140E IP Deskphone
- 0627C7J / 1150E IP Deskphone
- 0626C7J / 1165E IP Deskphone
- 062AC7J / 1210 IP Deskphone
- 062AC7J / 1220 IP Deskphone
- 062AC7J / 1230 IP Deskphone

29 2010

UNISlim Software Release 4.2 IP Deskphone Software

- 0621C7G / 2007 IP Deskphone
- 0623C7M / 1110 IP Deskphone
- 0624C7M / 1120E IP Deskphone
- 0625C7M / 1140E IP Deskphone
- 0627C7M / 1150E IP Deskphone
- 0626C7M / 1165E IP Deskphone
- 062AC7M / 1210 IP Deskphone
- 062AC7M / 1220 IP Deskphone
- 062AC7M / 1230 IP Deskphone

### Protocol description

Query Audio Manager

(16 xx 00 xx...)

Note:

Ensure that the handshake commands

1A 04 01 08

1A 07 07 01 23 45 67

are sent to i2004 before sending the commands in column 2. (Requests attributes of the Audio manager)

16 05 00 01 00

Note: Last byte can contain any value. The message length should be 5.

If the length is wrong it is ignored e.g. send



16 04 00 01  
16 06 00 01 00 03  
(Requests options setting of the Audio manager)  
16 05 00 02 03  
Note: Last byte can contain any value. The message length should be 5.  
If the length is wrong it is ignored.  
(Requests Alerting selection)  
16 05 00 04 0F  
Note: Last byte can contain any value. The message length should be 5.  
If the length is wrong it is ignored.  
(Requests adjustable Rx volume information command)  
16 05 00 08 00  
Note: Last byte can contain any value. The message length should be 5.  
If the length is wrong it is ignored.  
(Requests the i2004 to send the APB's Default Rx Volume command. The APB Number or stream based tone is provided in the last byte of the command below)  
16 05 00 10 00 (none)  
16 05 00 10 01 (Audio parameter bank 1, NBHS)  
16 05 00 10 02 (Audio parameter bank 2, NBHDS)  
16 05 00 10 03 (Audio parameter bank 3, NBHF)  
16 05 00 10 04 (Audio parameter bank 4, WBHS)  
16 05 00 10 05 (Audio parameter bank 5, WBHDS)  
16 05 00 10 06 (Audio parameter bank 6, WBHF)  
16 05 00 10 07 (Audio parameter bank 7,)  
16 05 00 10 08 (Audio parameter bank 8,)  
16 05 00 10 09 (Audio parameter bank 9,)  
16 05 00 10 0A (Audio parameter bank 0xA,)  
16 05 00 10 0B (Audio parameter bank 0xB,)  
16 05 00 10 0C (Audio parameter bank 0xC,)  
16 05 00 10 0D (Audio parameter bank 0xD,)  
16 05 00 10 0E (Audio parameter bank 0xE,)  
16 05 00 10 0F (Audio parameter bank 0xF,)  
16 05 00 10 10 (Alerting tone)  
16 05 00 10 11 (Special tones)  
16 05 00 10 12 (Paging tones)  
16 05 00 10 13 (Not Defined)  
16 05 00 10 1x (Not Defined)  
(Set the volume range in configuration message for each of the APBs and for alerting, paging and special tones (see below) and then send the following commands)  
(Requests handset status, when NBHS is 1) connected 2) disconnected)  
16 05 00 40 09  
Note: Last byte can contain any value. The message length should be 5.  
If the length is wrong it is ignored  
(Requests headset status, when HDS is disconnected)  
16 05 00 80 0A  
(Requests headset status, when HDS is connected)  
16 05 00 80 0A  
Note: Last byte can contain any value. The message length should be 5.  
If the length is wrong it is ignored  
(Requests handset and headset status when NBHS and HDS are disconnected)  
16 05 00 C0 05  
(Requests handset and headset status when NBHS and HDS are connected)  
16 05 00 C0 05  
(Send an invalid message)  
16 03 00  
(Send an invalid message. Is this an invalid msg??)  
16 06 00 22 22 22  
Query Supervisory headset status  
(16 03 01)  
16 03 01

Audio Manager Options  
(16 04 02 xx)  
(Maximum tone volume is one level lower than physical maximum  
Volume level adjustments are not performed locally in the i2004  
Adjustable Rx volume reports not sent to the NI when volume keys are pressed  
Single tone frequency NOT sent to HS port while call in progress.  
Single tone frequency NOT sent to HD port while call in progress.  
Automatic noise squelching disabled.  
HD key pressed command sent when i2004 receives make/break sequence.)  
16 04 02 00  
(Maximum tone volume is set to the physical maximum)

16 04 02 01  
(then requests options setting of the Audio manager by sending 16 04 00 02)  
(Volume level adjustments are performed locally in the i2004)  
16 04 02 02  
(then requests options setting of the Audio manager by sending 16 04 00 02)  
(Adjustable Rx volume reports sent to the NI when volume keys are pressed)  
16 04 02 04  
(then requests options setting of the Audio manager by sending 16 04 00 02)  
(Single tone frequency sent to HS port while call in progress)  
16 04 02 08  
(then requests options setting of the Audio manager by sending 16 04 00 02)  
(Single tone frequency sent to HD port while call in progress)  
16 04 02 10  
(then requests options setting of the Audio manager by sending 16 04 00 02)  
(Automatic noise squelching enabled.)  
16 04 02 20  
(then requests options setting of the Audio manager by sending 16 04 00 02)  
(Headset Rfeature Key Pressed command sent when i2004 receives  
make/break sequence.)  
16 04 02 40  
(then requests options setting of the Audio manager by sending 16 04 00 02)  
(In this case both bit 1 and bit 3 are set, hence Volume level  
adjustments are performed  
locally in the i2004 and Single tone frequency sent to HS port while  
call in progress.)  
16 04 02 0A

#### Mute/un-mute

(16 xx 04 xx...)  
(In this case two phones are conneted. Phone 1 is given the ID  
47.129.31.35 and phone 2  
is given the ID 47.129.31.36. Commands are sent to phone 1 )  
(TX is muted on stream ID 00)  
16 05 04 01 00  
(TX is un-muted on stream ID 00)  
16 05 04 00 00  
(RX is muted on stream ID 00)  
16 05 04 03 00  
(RX is un-muted on stream ID 00)  
16 05 04 02 00  
(TX is muted on stream ID 00, Rx is un-muted on stream ID 00)  
16 07 04 01 00 02 00  
(TX is un-muted on stream ID 00, Rx is muted on stream ID 00)  
16 07 04 00 00 03 00  
(TX is un-muted on stream ID 00, Rx is un-muted on stream ID 00)  
16 07 04 00 00 02 00

#### Transducer Based tone on

(16 04 10 xx)  
(Alerting on)  
16 04 10 00  
(Special tones on, played at down loaded tone volume level)  
16 04 10 01  
(paging on)  
16 04 10 02  
(not defined)  
16 04 10 03  
(Alerting on, played at two steps lower than down loaded tone volume level)  
16 04 10 08  
(Special tones on, played at two steps lower than down loaded tone volume level)  
16 04 10 09

#### Transducer Based tone off

(16 04 10 xx)  
16 04 11 00 (Alerting off)  
16 04 11 01 (Special tones off)  
16 04 11 02 (paging off)  
16 04 11 03 (not defined)

#### Alerting tone configuration

(16 05 12 xx xx)  
(Note: Volume range is set here for all tones. This should be noted  
when testing the volume level message)  
(HF speaker with different warbler select values, tone volume range set to max)  
16 05 12 10 00  
16 05 12 11 0F  
16 05 12 12 0F

16 05 12 13 0F  
16 05 12 14 0F  
16 05 12 15 0F  
16 05 12 16 0F  
16 05 12 17 0F  
(HF speaker with different cadence select values, tone volume range set to max)  
16 05 12 10 0F  
16 05 12 10 1F  
16 05 12 10 2F  
16 05 12 10 3F  
16 05 12 10 4F  
16 05 12 10 5F  
16 05 12 10 6F  
16 05 12 10 7F (configure cadence with alerting tone cadence download  
message before sending this message)  
(HS speaker with different warbler select values, tone volume level set to max)  
16 05 12 00 0F  
16 05 12 01 0F  
16 05 12 02 0F  
16 05 12 03 0F  
16 05 12 04 0F  
16 05 12 05 0F  
16 05 12 06 0F  
16 05 12 07 0F  
(HS speaker with different cadence select values, tone volume range set to max)  
16 05 12 00 0F  
16 05 12 00 1F  
16 05 12 00 2F  
16 05 12 00 3F  
16 05 12 00 4F  
16 05 12 00 5F  
16 05 12 00 6F  
16 05 12 00 7F (configure cadence with alerting tone cadence download  
message before sending this message)  
(HD speaker with different warbler select values, tone volume range set to max)  
16 05 12 08 0F  
16 05 12 09 0F  
16 05 12 0A 0F  
16 05 12 0B 0F  
16 05 12 0C 0F  
16 05 12 0D 0F  
16 05 12 0E 0F  
16 05 12 0F 0F  
(HD speaker with different cadence select values, tone volume level set to max)  
16 05 12 08 0F  
16 05 12 08 1F  
16 05 12 08 2F  
16 05 12 08 3F  
16 05 12 08 4F  
16 05 12 08 5F  
16 05 12 08 6F  
16 05 12 08 7F (configure cadence with alerting tone cadence download  
message before sending this message)

Special tone configuration

(16 06 13 xx xx)

(Note: Volume range is set here for all tones. This should be noted  
when testing the volume level message)

(HF speaker with different tones, tone volume range is varied)

16 06 13 10 00 01  
16 06 13 10 01 01  
16 06 13 10 08 01  
16 06 13 10 02 07  
16 06 13 10 03 07  
16 06 13 10 04 11  
16 06 13 10 05 11  
16 06 13 10 06 18  
16 06 13 10 07 18  
16 06 13 10 08 1F  
(HF speaker with different cadences and tones; tone volume level is varied)  
16 06 13 10 00 01  
16 06 13 10 10 01  
16 06 13 10 20 07  
16 06 13 10 30 07  
16 06 13 10 40 11  
16 06 13 10 50 11  
16 06 13 10 60 18

16 06 13 10 70 18 (configure cadence with special tone cadence  
download message before sending this message)  
(HS speaker with different tones, tone volume range is varied)  
16 06 13 00 00 01  
16 06 13 00 01 01  
16 06 13 00 02 07  
16 06 13 00 03 07  
16 06 13 00 04 11  
16 06 13 00 05 11  
16 06 13 00 06 18  
16 06 13 00 07 18  
(HS speaker with different cadences and tones; tone volume range is varied)  
16 06 13 00 00 01  
16 06 13 00 10 01  
16 06 13 00 20 07  
16 06 13 00 30 07  
16 06 13 00 40 11  
16 06 13 00 50 11  
16 06 13 00 60 18  
16 06 13 00 70 18 (configure cadence with special tone cadence  
download message before sending this message)  
(HD speaker with different tones, tone volume range is varied)  
16 06 13 08 00 01  
16 06 13 08 01 01  
16 06 13 08 02 07  
16 06 13 08 03 07  
16 06 13 08 04 11  
16 06 13 08 05 11  
16 06 13 08 06 18  
16 06 13 08 07 18  
(HD speaker with different cadences and tones; tone volume range is varied)  
16 06 13 08 00 01  
16 06 13 08 10 01  
16 06 13 08 20 07  
16 06 13 08 30 07  
16 06 13 08 40 11  
16 06 13 08 50 11  
16 06 13 08 60 18  
16 06 13 08 70 18 (configure cadence with special tone cadence  
download message before sending this message)

Paging tone configuration

(16 05 14 xx xx)  
(Note: Volume range is set here for all tones. This should be noted  
when testing the volume level message)  
(HF speaker with different cadence select values, tone volume range set to max)  
16 05 14 10 0F  
16 05 14 10 1F  
16 05 14 10 2F  
16 05 14 10 3F  
16 05 14 10 4F  
16 05 14 10 5F  
16 05 14 10 6F  
16 05 14 10 7F (configure cadence with paging tone cadence download  
message before sending this message)  
(HS speaker with different cadence select values, tone volume range set to max)  
16 05 14 00 0F  
16 05 14 00 1F  
16 05 14 00 2F  
16 05 14 00 3F  
16 05 14 00 4F  
16 05 14 00 5F  
16 05 14 00 6F  
16 05 14 00 7F (configure cadence with paging tone cadence download  
message before sending this message)  
(HD speaker with different cadence select values, tone volume level set to max)  
16 05 14 08 0F  
16 05 14 08 1F  
16 05 14 08 2F  
16 05 14 08 3F  
16 05 14 08 4F  
16 05 14 08 5F  
16 05 14 08 6F  
16 05 14 08 7F (configure cadence with paging tone cadence download  
message before sending this message)

Alerting Tone Cadence Download

(16 xx 15 xx xx...)  
16 08 15 00 0A 0f 14 1E  
(.5 sec on, 0.75 sec off; 1 sec on 1.5 sec off, cyclic)  
16 0C 15 01 0A 0f 14 1E 05 0A 0A 14  
(.5 sec on, 0.75 sec off; 1 sec on 1.5 sec off; 0.25sec on, 0.5sec  
off; 0.5 sec on, 1 sec off , one shot)

#### Special Tone Cadence Download

(16 xx 16 xx xx...)  
16 05 16 0A 10  
(125ms on, 200 ms off)  
16 09 16 0A 10 14 1E  
(125ms on, 200 ms off; 250ms on, 375ms off )

#### Paging Tone Cadence Download

(16 xx 17 xx xx...)  
16 06 17 01 0A 10  
(125ms on, 200 ms off, 250HZ)  
16 06 17 04 05 10  
(62.5ms on, 200 ms off, 500 Hz)  
16 09 17 01 0A 10 10 14 1E  
(125ms on, 200 ms off; 250ms on, 375ms off, 250 Hz, 100Hz )  
16 0C 17 01 0A 10 04 14 1E 10 0A 10  
(125ms on, 200 ms off; 250ms on, 375ms off; 125ms on, 200 ms off,  
250Hz, 1000Hz, 500 Hz )  
16 0C 17 01 1E 10 12 3c 1E 10 28 10  
(375ms on, 200 ms off; 750ms on, 375ms off; 500ms on, 200 ms off,  
250Hz, (333Hz,1000Hz), 500 Hz )

#### Transducer Based Tone Volume Level

(16 04 18 xx)  
(Ensure that the volume range is set properly in the alerting, special  
and paging  
tone configuration e.g if the volume range is set to zero, this  
message will always output  
max volume) (Different volume level for alerting tone. Note: Send the  
command below  
and then send the alerting on command and alerting off commands)

16 04 18 00  
16 04 18 10  
16 04 18 20  
16 04 18 30  
16 04 18 40  
16 04 18 50  
16 04 18 60  
16 04 18 70  
16 04 18 80  
16 04 18 90  
16 04 18 F0  
(HF:Volume range for alerting tone is changed here using these commands)

16 05 12 10 0F  
16 05 12 10 00  
16 05 12 10 04

(HD:Volume range for alerting tone is changed here using these commands)

16 05 12 08 0F  
16 05 12 08 00  
16 05 12 08 04

(Different volume level for special tone)

16 04 18 01  
16 04 18 11  
16 04 18 21  
16 04 18 31  
16 04 18 41  
16 04 18 51  
16 04 18 61  
16 04 18 71  
16 04 18 81  
16 04 18 91  
16 04 18 A1  
16 04 18 B1  
16 04 18 C1  
16 04 18 D1  
16 04 18 E1  
16 04 18 F1

(HF:Volume range for special tone is changed here using these commands)

16 06 13 10 20 07  
16 06 13 10 25 07

16 06 13 10 2F 07  
 (HD:Volume range for special tone is changed here using these commands)  
 16 06 13 08 20 07  
 16 06 13 08 25 07  
 16 06 13 08 2F 07  
 (Different volume level for paging tone)  
 16 04 18 02  
 16 04 18 12  
 16 04 18 22  
 16 04 18 32  
 16 04 18 42  
 16 04 18 52  
 16 04 18 62  
 16 04 18 72  
 16 04 18 82  
 16 04 18 92  
 16 04 18 F2  
 (HF:Volume range for paging tone is changed here using these commands)  
 16 05 14 10 0F  
 16 06 14 10 00  
 16 06 14 10 04  
 (HD:Volume range for paging tone is changed here using these commands)  
 16 06 14 08 0F  
 16 06 14 08 00  
 16 06 14 08 04  
  
 Alerting Tone Test  
 (16 04 19 xx)  
 (tones 667Hz, duration 50 ms and 500Hz duration 50 ms)  
 16 04 19 00  
 (tones 333Hz, duration 50 ms and 250Hz duration 50 ms)  
 16 04 19 01  
 (tones 333 Hz + 667 Hz duration 87.5 ms and 500Hz + 1000Hz duration 87.5 ms)  
 16 04 19 02  
 (tones 333 Hz, duration 137.5 ms; 500Hz duration 75 ms; 667Hz duration 75 ms)  
 16 04 19 03  
 (tones 500Hz, duration 100 ms and 667Hz duration 100 ms)  
 16 04 19 04  
 (tones 500Hz, duration 400 ms and 667Hz duration 400 ms)  
 16 04 19 05  
 (tones 250Hz, duration 100 ms and 333Hz duration 100 ms)  
 16 04 19 06  
 (tones 250Hz, duration 400 ms and 333 Hz, duration 400ms)  
 16 04 19 07  
  
 Visual Transducer Based Tones Enable  
 (16 04 1A xx)  
 Visual tone enabled  
 16 04 1A 01  
 (Visual tone disabled)  
 16 04 1A 00  
  
 Stream Based Tone On  
 (16 06 1B xx xx xx)  
 (Dial tone is summed with data on Rx stream 00 at volume level -3dBm0)  
 16 06 1B 00 00 08  
 (Dial tone replaces the voice on Rx stream 00 at volume level -6dBm0)  
 16 06 1B 80 00 10  
 (Dial tone is summed with voice on Tx stream 00 at volume level -3dBm0)  
 16 06 1B 40 00 08  
 (Dial tone replaces the voice on Tx stream 00 at volume level -3dBm0)  
 16 06 1B C0 00 08  
  
 (Line busy tone is summed with data on Rx stream 00 at volume level -3dBm0)  
 16 06 1B 02 00 08  
 (Line busy tone replaces the voice on Rx stream 00 at volume level -6dBm0)  
 16 06 1B 82 00 10  
 (Line busy tone is summed with voice on Tx stream 00 at volume level -3dBm0)  
 16 06 1B 42 00 08  
 (Line busy tone replaces the voice on Tx stream 00 at volume level -3dBm0)  
 16 06 1B C2 00 08  
  
 (ROH tone is summed with data on Rx stream 00 at volume level -3dBm0)  
 16 06 1B 05 00 08  
 (ROH tone replaces the voice on Rx stream 00 at volume level -6dBm0)  
 16 06 1B 85 00 10  
 (ROH tone is summed with voice on Tx stream 00 at volume level -3dBm0)  
 16 06 1B 45 00 08

(ROH tone replaces the voice on Tx stream 00 at volume level -3dBm0)  
16 06 1B C5 00 08  
(Recall dial tone is summed with data on Rx stream 00 at volume level -3dBm0)  
16 06 1B 01 00 08  
(Recall dial tone replaces the voice on Rx stream 00 at volume level -6dBm0)  
16 06 1B 81 00 10

(Reorder tone is summed with data on Rx stream 00 at volume level -3dBm0)  
16 06 1B 03 00 08  
(Reorder dial tone replaces the voice on Rx stream 00 at volume level -6dBm0)  
16 06 1B 83 00 10

(Audible Ringing tone is summed with data on Rx stream 00 at volume level -3dBm0)  
16 06 1B 04 00 08  
(Audible Ringing tone replaces the voice on Rx stream 00 at volume level -6dBm0)  
16 06 1B 84 00 10

(Stream based tone ID 06 is summed with data on Rx stream 00 at volume level -3dBm0;  
Tone ID 06 is downloaded using both the frequency and cadence down load commands)  
16 06 1B 06 00 08  
(Stream based tone ID 06 replaces the voice on Rx stream 00 at volume level -6dBm0)  
16 06 1B 86 00 10

(Stream based tone ID 0F is summed with data on Rx stream 00 at volume level -3dBm0;  
Tone ID 0x0F is downloaded using both the frequency and cadence down load commands)  
16 06 1B 0F 00 08  
(Stream based tone ID 0F replaces the voice on Rx stream 00 at volume level -6dBm0)  
16 06 1B 8F 00 10

Stream Based Tone Off  
(16 05 1C xx xx)  
(Dial tone is turned off on Rx stream 00)  
16 05 1C 00 00  
(Dial tone is turned off on Tx stream 00)  
16 05 1C 40 00  
(Line busy tone is turned off on Rx stream 00)  
16 05 1C 02 00  
(Line busy tone is turned off on Tx stream 00)  
16 05 1C 42 00  
(ROH tone is turned off on Rx stream 00)  
16 05 1C 05 00  
(ROH tone is turned off on Tx stream 00)  
16 05 1C 45 00  
(Recall dial tone is turned off on Rx stream 00)  
16 05 1C 01 00  
(Reorder tone is turned off on Rx stream 00)  
16 05 1C 03 00  
(Audible Ringing tone is turned off on Rx stream 00)  
16 05 1C 04 00  
(Stream based tone ID 06 is turned off on Rx stream 00)  
16 05 1C 06 00  
(Stream based tone ID 0F is turned off on Rx stream 00)  
16 05 1C 0F 00

Stream Based Tone Frequency Component List Download (up to 4 frequencies can be specified)  
(16 xx 1D xx...)  
Note: Frequency component download and cadence download commands sent to the i2004 first.  
Then send the stream based tone ID on command to verify that tones are turned on.  
16 06 1D 06 2C CC  
(1400Hz )  
16 08 1D 07 2C CC 48 51  
(1400 Hz and 2250Hz)

Stream Based Tone Cadence Download (up to 4 cadences can be specified)  
(16 xx 1E xx...)  
Note: Frequency component download and cadence download commands sent to the i2004 first. Then  
send the stream based tone ID on command to verify that tones are turned on.

16 06 1E 26 0A 0A  
(200 ms on and 200 ms off with tone turned off after the full sequence)  
16 08 1E 07 0A 0A 14 14  
(20 ms on and 20 ms off for first cycle, 400 ms on and 400 ms off for the second cycle with sequence repeated)  
16 05 1E 26 0A  
(In this case tone off period is not specified hence tone is played until stream based tone off command is received.)

#### Select Adjustable Rx Volume

(16 04 20 xx)  
16 04 20 01  
(Audio parameter block 1)  
16 04 20 03  
(Audio parameter block 3)  
16 04 20 08  
(Alerting Rx volume)  
16 04 20 09  
(Special tone Rx volume)  
16 04 20 0a  
(Paging tone Rx volume)

#### Set APB's Rx Volume Levels

(16 05 21 xx xx)  
16 05 21 01 25  
(? Rx volume level 5 steps louder than System RLR)  
16 05 21 01 05  
(? Rx volume level 5 steps quieter than System RLR)

#### Change Adjustable Rx Volume

16 03 22  
(Rx volume level is one step quieter for the APB/tones selected through Select Adjustable Rx Volume command)  
16 03 23  
(Rx volume level is one step louder for the APB/tones selected through Select Adjustable Rx Volume command)

#### Adjust Default Rx Volume

(16 04 24 xx)  
16 04 24 01  
(Default Rx volume level is one step quieter for the APB 1)  
16 04 25 01  
(Default Rx volume level is one step louder for the APB 1)

#### Adjust APB's Tx and/or STMR Volume Level

(16 04 26 xx)  
(First ensure that the Tx and STMR volume level are set to maximum by repeatedly (if needed) sending the command  
16 04 26 F2 to APB2.  
Rest of the commands are sent to i2004 individually and then the query command below is used to verify if the commands are sent correctly) (Enable both Tx Vol adj. and STMR adj; Both Tx volume and STMR volume are one step louder on APB 2)  
16 04 26 F2  
(Enable both Tx Vol adj. and STMR adj; Both Tx volume and STMR volume are one step quieter on APB 2)  
16 04 26 A2  
(Enable Tx Vol adj. and disable STMR adj; Tx volume is one step louder on APB 3)  
16 04 26 C3  
(Enable Tx Vol adj. and disable STMR adj; Tx volume is one step quieter on APB 3)  
16 04 26 83  
(Disable both Tx Vol adj. and STMR adj on APB 1)  
16 04 26 01

#### Query APB's Tx and/or STMR Volume Level

(16 04 27 XX)  
(Query Tx volume level and STMR volume level on APB 2)  
16 04 27 32  
(Query STMR volume level on APB 1)  
16 04 27 11  
(Query STMR volume level on APB 2)  
16 04 27 12  
(Query STMR volume level on APB 3)  
16 04 27 13



(Query Tx volume level on APB 1)  
16 04 27 21  
(Query Tx volume level on APB 2)  
16 04 27 22  
(Query Tx volume level on APB 3)  
16 04 27 23

APB Download  
(16 xx-1F xx...)  
16 09 28 FF AA 88 03 00 00

Open Audio Stream  
(16 xx 30 xx...)  
(If Audio stream is already open it has to be closed before another  
open audio stream command is sent)  
16 15 30 00 00 00 01 00 13 89 00 00 13 89 00 00 2F 81 1F 23  
(Open G711 ulaw Audio stream to 2F.81.1F.9F)  
16 15 30 00 00 08 08 01 00 13 89 00 00 13 89 00 00 2F 81 1F 23  
(Open G711 Alaw Audio stream to 2F.81.1F.9F)  
16 15 30 00 00 12 12 01 00 13 89 00 00 13 89 00 00 2F 81 1F 23  
(Open G729 Audio stream to 2F.81.1F.9F)  
16 15 30 00 00 04 04 01 00 13 89 00 00 13 89 00 00 2F 81 1F 23  
(Open G723? ulaw Audio stream to 2F.81.1F.9F)

Close Audio Stream  
(16 05 31 xx xx)  
16 05 31 00 00

Connect Transducer  
(16 06 32 xx xx xx)  
16 06 32 C0 11 00  
(Connect the set in Handset mode with no side tone)  
16 06 32 C0 01 00  
(Connect the set in Handset mode with side tone)  
16 06 32 C1 12 00  
(Connect the set in Headset mode with no side tone)  
16 06 32 C1 02 00  
(Connect the set in Headset mode with side tone)  
16 06 32 C2 03 00  
(Connect the set in Hands free mode)

Frequency Response Specification  
(16 xx 33 xx...)  
Filter Block Download 16 xx 39 xx  
Voice Switching debug 16 04 35 11  
(Full Tx, Disable switch loss bit)  
16 04 35 12  
(Full Rx, Disable switch loss bit)  
Voice Switching Parameter Download 16 08 36 01 2D 00 00 02  
(APB 1, AGC threshold index 0, Rx virtual pad 0, Tx virtual pad 0,  
dynamic side tone enabled)  
Query RTCP Statistics 16 04 37 12  
(queries RTCP bucket 2, resets RTCP bucket info.)  
Configure Vocoder Parameters 16 0A 38 00 00 CB 00 E0 00 A0  
(For G711 ulaw 20 ms, NB)  
16 0A 38 00 08 CB 00 E0 00 A0  
(G711 Alaw 20 ms, NB)  
16 0A 38 00 00 CB 01 E0 00 A0  
(For G711 ulaw 10 ms, WB)  
16 0A 38 00 08 CB 01 E0 00 A0  
(G711 Alaw 10 ms, WB)  
16 08 38 00 12 C1 C7 C5  
(For G729 VAD On, High Pass Filter Enabled, Post Filter Enabled)  
16 09 38 00 04 C9 C5 C7 C1  
(G723 VAD On, High Pass Filter Enabled, Post Filter Enabled at 5.3 KHz)  
16 09 38 00 04 C0 C7 C5 C9  
(G723 VAD Off, High Pass Filter Enabled, Post Filter Enabled at 5.3 KHz)  
16 09 38 00 04 C1 C5 C7 C8  
(G723 VAD On, High Pass Filter Enabled, Post Filter Enabled at 6.3 KHz)  
16 09 38 00 04 C0 C7 C5 C8  
(G723 VAD Off, High Pass Filter Enabled, Post Filter Enabled at 6.3 KHz)  
Query RTCP Bucket's SDES Information (39 XX) (The first nibble in the  
last byte indicates the bucket ID)  
16 04 39 21  
16 04 39 22  
16 04 39 23  
16 04 39 24  
16 04 39 25

16 04 39 26  
16 04 39 27

16 04 39 01  
16 04 39 12  
16 04 39 23  
16 04 39 34  
16 04 39 45  
16 04 39 56  
16 04 39 67

# Skinny

chan\_skinny stuff

## Skinny call logging

Page for details about call logging.

Calls are logged in the devices placed call log (directories->Place Calls) when a call initially connects to another device. Subsequent changes in the device (eg forwarded) are not reflected in the log.

If a call is not placed to a channel they will not be recorded in the log. eg a call to voicemail will not be recorded. You can force these to be recorded by including `progress()`, then `ringing()` in the dialplan.

Example (This will produce a logged call):

```
exten => 100,1,NoOp
exten => 100,n,Progress
exten => 100,n,Ringing
exten => 100,n,VoiceMailMain(${CALLERID(num)}@mycontext,s)
```

Example (This will not):

```
exten => 100,1,NoOp
exten => 100,n,VoiceMailMain(${CALLERID(num)}@mycontext,s)
```

## **Skinny Dev Notes**

A spot to keep development notes.

## Keepalives

Been doing some mucking around with cisco phones. Things found out about keepalives documented here.

It appears the minimum keepalive is 10. Any setting below this reverts to the the device setting 10 seconds.

Keepalive timings seem to vary by device type (and probably firmware).

Device	F/Ware	Proto	1st KA	Behavior w/ no response
7960	7.2(3.0)	6	15 Sec	KA, KA, KA, UNREG
7961	8.5.4(1.6)	17	As set	KA, KA*2, KA*2, UNREG
7920	4.0-3-2	5	As set	KA, KA, KA, KA+Reset Conn

For example, with keepalive set to 20:

- the 7960 will UNREG in 75 sec (ka@15, ka@35, ka@55, unreg@75) (straight after registration); or
- the 7960 will UNREG in 80 sec (ka@20, ka@40, ka@60, unreg@80) (after 1 keepalive ack sent);
- the 7961 will UNREG in 120 sec (ka@20, ka@60, ka@100, unreg@120).

Other info:

- Devices appear to consider themselves still registered (with no indication provided to user) until the unregister/reset conn occurs.
- Devices generally do not respond to keepalives or reset their own timings (see below for exception)
- After unregister (but no reset obviously) keepalives are still sent, further, the device now responds to keepalives with a keepalive\_ack, but this doesn't affect the timing of their own keepalives.

chan\_skinny impact:

- need to revise keepalive timing with is currently set to unregister at  $1.1 * \text{keepalive time}$

Testing wifi (7920 with keepalive set to 20), immediately after a keepalive:

- removed from range for 55 secs - at 58 secs 3 keepalives received, connection remains.
- removed from range for 65 secs - at about 80 secs, connection reset and device reloads.
- server set to ignore 2 keepalives - 3rd keepalive at just under 60secs, connection remains.
- server set to ignore 3 keepalives - 4th keepalive at just under 80secs, connection reset by device anyway.
- looks like timing should be about  $3 * \text{keepalive}$  (ie 60secs), maybe  $5 * \text{keepalive}$  for 7961 (v17?)

More on ignoring keepalives at the server (with the 7920) (table below)

- if keepalive is odd, the time used is rounded up to the next even number (ie 15 will result in 16 secs)
- the first keepalive is delayed by 1 sec if keepalive is less than 30, 15 secs if less than 120, else 105 secs
- these two lead to some funny numbers
- if set to 119, the first will be at 135 secs (119 rounded up + 15), and subsequent each 120 secs
- if set to 120, the first will be at 225 secs (120 not rounded + 105), and subsequent each 120 secs
- similarly if set to 29, the first will be 31 then 30, where if set to 30 the first will be 45 then 30
- only tested out to 600 secs (where the first is still delayed by 105 secs)
- device resets the connection 20 secs after the 3rd unreplied keepalive
- keepalives below 20 seem unreliable in that they do not reset the connection
- above 20secs and after the first keepalive, the device will reset at  $(\text{TRUNC}((\text{KA}+1)/2)*2)*3+20$
- before the first keepalive, add 1 if  $\text{KA} < 30$ , add 15 if  $\text{KA} < 120$ , else add 105
- actually, about a second earlier. After the first missed KA, the next will be about a second early
- not valid for other devices

Set	First (s)	Then (s)	Packets (#)	Reset (s)
20	21	20	3	20
25	27	26	3	20
26	27	26	3	20
29	31	30	3	20
30	45	30	3	20
60	75	60	3	20
90	105	90	3	20
119	135	120	3	20
120	225	120	3	20
600	705	600	3	20



## **Skinny device stuff**

Collection of notes on weird device stuff.

### 7937 Conference Phone

- firmware appears to have 10 speedial buttons hardcoded into the firmware.



# RTP Packetization

## Overview

Asterisk currently supports configurable RTP packetization per codec for select RTP-based channels.

### Channels

These channel drivers allow RTP packetization on a user/peer/friend or global level:

- chan\_sip
- chan\_skinny
- chan\_h323
- chan\_oooh323 (Asterisk-Addons)
- chan\_gtalk
- chan\_jingle
- chan\_motif (Asterisk 11+)

### Configuration

To set a desired packetization interval on a specific codec, append that interval to the allow= statement.

Example:

```
allow=ulaw:30,alaw,g729:60
```

No packetization is specified in the case of alaw in this example, so the default of 20ms is used.

### Autoframing

In addition, chan\_sip has the ability to negotiate the desired framing at call establishment.

In sip.conf if autoframing=yes is set in the global section, then all calls will try to set the packetization based on the remote endpoint's preferences. This behaviour depends on the endpoints ability to present the desired packetization (ptime\:) in the SDP. If the endpoint does not include a ptime attribute, the call will be established with 20ms packetization.

Autoframing can be set at the global level or on a user/peer/friend basis. If it is enabled at the global level, it applies to all users/peers/friends regardless of their preferred codec packetization.

### Codec framing options

The following table lists the minimum and maximum values that are valid per codec, as well as the increment value used for each. Please note that the maximum values here are only recommended maximums, and should not exceed the RTP MTU.

Name	Minimum (ms)	Maximum (ms)	Default (ms)	Increment (ms)
<b>g723</b>	30	300	30	30
<b>gsm</b>	20	300	20	20
<b>ulaw</b>	10	150	20	10
<b>alaw</b>	10	150	20	10
<b>g726</b>	10	300	20	10
<b>ADPCM</b>	10	300	20	10
<b>SLIN</b>	10	70	20	10
<b>lpc10</b>	20	20	20	20
<b>g729</b>	10	230	20	10
<b>speex</b>	10	60	20	10
<b>ilbc</b>	30	30	30	30
<b>g726_aal2</b>	10	300	20	10

Invalid framing options are handled based on the following rules:

1. If the specified framing is less than the codec's minimum, then the minimum value is used.
2. If the specific framing is greater than the codec's maximum, then the maximum value is used

3. If the specified framing does not meet the increment requirement, the specified framing is rounded down to the closest valid framing options.

# Dialplan

## The Asterisk dialplan

The dialplan is essentially a scripting language specific to Asterisk and one of the primary ways of instructing Asterisk on how to behave. It ties everything together, allowing you to route and manipulate calls in a programmatic way. The pages in this section will describe what the elements of dialplan are and how to use them in your configuration.

## Dialplan configuration file

The Asterisk dialplan is found in the extensions.conf file in the [configuration directory](#), typically /etc/asterisk.

If you modify the dialplan, you can use the Asterisk CLI command "dialplan reload" to load the new dialplan without disrupting service in your PBX.

## Example dialplan

The example dial plan, in the configs/samples/extensions.conf.sample file is installed as extensions.conf if you run "make samples" after [installation of Asterisk](#). The sample file includes many examples of dialplan programming for specific scenarios and environments often common to Asterisk implementations.

### On This Page

- [The Asterisk dialplan](#)
- [Dialplan configuration file](#)
- [Example dialplan](#)

### Topics

- [Contexts, Extensions, and Priorities](#)
- [Special Dialplan Extensions](#)
- [Include Statements](#)
- [Switch Statements](#)
- [Variables](#)
- [Pattern Matching](#)
- [Subroutines](#)
- [Expressions](#)
- [Conditional Applications](#)

# Contexts, Extensions, and Priorities

## Dialplan Format

The dialplan in `extensions.conf` is organized into sections, called contexts. Contexts are the basic organizational unit within the dialplan, and as such, they keep different sections of the dialplan independent from each other. You can use contexts to separate out functionality and features, enforce security boundaries between the various parts of our dialplan, as well as to provide different classes of service to groups of users.

### Dialplan contexts

The syntax for a context is exactly the same as any other section heading in the configuration files, as explained in [Sections and Settings](#). Simply place the context name in square brackets. For example, here we define an example context called 'users'.

```
[users]
```

- [Dialplan Format](#)
  - [Dialplan contexts](#)
  - [Dialplan extensions](#)
  - [Dialplan priorities](#)
  - [Application calls](#)
- [Dialplan search order](#)

### Dialplan extensions

Within each context, we can define one or more **extensions**. An extension is simply a named set of actions. Asterisk will perform each action, in sequence, when that extension number is dialed. The syntax for an extension is:

```
exten => number,priority,application([parameter[,parameter2...]])
```

Let's look at an example extension.

```
exten => 6001,1,Dial(PJSIP/demo-alice,20)
```

In this case, the extension number is **6001**, the priority number is **1**, the **application** is **Dial()**, and the two parameters to the application are **PJSIP/demo-alice** and **20**.

### Dialplan priorities

Within each extension, there must be one or more *priorities*. A priority is simply a sequence number. The first priority on an extension is executed first. When it finishes, the second priority is executed, and so forth.



#### Priority numbers

Priority numbers must begin with 1, and must increment sequentially. If Asterisk can't find the next priority number, it will terminate the call. We call this *auto-fallthrough*. Consider the example below:

```
exten => 6123,1,do something
exten => 6123,2,do something else
exten => 6123,4,do something different
```

In this case, Asterisk would execute priorities one and two, but would then terminate the call, because it couldn't find priority number three.

#### Priority letter n

Priority numbers can also be simplified by using the letter **n** in place of the priority numbers greater than one. The letter **n** stands for **next**, and when Asterisk sees priority **n** it replaces it in memory with the previous priority number plus one. Note that you must still explicitly declare priority number one.

```
exten => 6123,1,NoOp( )
exten => 6123,n,Verbose("Do something!")
exten => 6123,n,Verbose("Do something different!")
```

### Application calls

You'll notice that each priority is calling a dialplan application (such as NoOp, or Verbose in the example above). That is how we tell Asterisk to "do something" with the [channel](#) that is executing dialplan. See the [Applications](#) section for more detail.

## Priority labels

You can also assign a label (or alias) to a particular priority number by placing the label in parentheses directly after the priority number, as shown below. Labels make it easier to jump back to a particular location within the extension at a later time.

```
exten => 6123,1,NoOp()  
exten => 6123,n(repeat),Verbose("Do something!")  
exten => 6123,n,Verbose("Do something different!")
```

Here, we've assigned a label named **repeat** to the second priority.

Included in the Asterisk 1.6.2 branch (and later) there is a way to avoid having to repeat the extension name/number or pattern using the **same =>** prefix.

```
exten => 6123,1,NoOp()  
same => n(repeat),Verbose("Do something!")  
same => n,Verbose("Do something different!")
```

## Dialplan search order

The order of matching within a context is always exact extensions, [pattern match](#) extensions, [include statements](#), and [switch statements](#). Includes are always processed depth-first. So for example, if you would like a switch "A" to match before context "B", simply put switch "A" in an included context "C", where "C" is included in your original context before "B".

Search order:

- Explicit extensions
- Pattern match extensions
- Includes
- Switches

Make sure to see the [Pattern Matching](#) page for a description of pattern matching order.

# Special Dialplan Extensions

## Special Asterisk Dialplan Extensions

Here we'll list all of the special built-in dialplan extensions and their usage.



Other than special extensions, there is a special context "default" that is used when either a) an extension context is deleted while an extension is in use, or b) a specific starting extension handler has not been defined (unless overridden by the low level channel interface).

### a: Assistant extension

This extension is similar to the **o** extension, only it gets triggered when the caller presses the asterisk (\*) key while recording a voice mail message. This is typically used to reach an assistant.

### e: Exception Catchall extension

This extension will substitute as a catchall for any of the 'i', 't', or 'T' extensions, if any of them do not exist and catching the error in a single routine is desired. The function `EXCEPTION` may be used to query the type of exception or the location where it occurred.

### h: Hangup extension

When a call is hung up, Asterisk executes the **h** extension in the current context. This is typically used for some sort of clean-up after a call has been completed.

### i: Invalid entry extension

If Asterisk can't find an extension in the current context that matches the digits dialed during the **Background()** or **WaitExten()** applications, it will send the call to the **i** extension. You can then handle the call however you see fit.

### o: Operator extension

If a caller presses the zero key on their phone keypad while recording a voice mail message, and the **o** extension exists, the caller will be redirected to the **o** extension. This is typically used so that the caller can press zero to reach an operator.

### s: Start extension

When an analog call comes into Asterisk, the call is sent to the **s** extension. The **s** extension is also used in macros.

Please note that the **s** extension is **not** a catch-all extension. It's simply the location that analog calls and macros begin. In our example above, it simply makes a convenient extension to use that can't be easily dialed from the **Background()** and **WaitExten()** applications.

### t: Response timeout extension

When the caller waits too long before entering a response to the **Background()** or **WaitExten()** applications, and there are no more priorities in the current extension, the call is sent to the **t** extension.

### T: Absolute timeout extension

This is the extension that is executed when the 'absolute' timeout is reached. See "core show function TIMEOUT" for more information on setting timeouts.

## Include Statements

Include statements allow us to split up the functionality in our dialplan into smaller chunks, and then have Asterisk search multiple contexts for a dialed extension. Most commonly, this functionality is used to provide security boundaries between different classes of callers.

It is important to remember that when calls come into the Asterisk dialplan, they get directed to a particular context by the channel driver. Asterisk then begins looking for the dialed extension in the context specified by the channel driver. By using include statements, we can include other contexts in the search for the dialed extension.

Asterisk supports two different types of include statements: regular includes and time-based includes.

## Include Statements Basics

To set the stage for our explanation of include statements, let's say that we want to organize our dialplan and create a new context called **features**. We'll leave our extensions **6001** and **6002** for Alice and Bob in the **users** context, and place extensions such as **6500** in the new **features** context. When calls come into the users context and doesn't find a matching extension, the include statement tells Asterisk to also look in the new **features** context.

The syntax for an include statement is very simple. You simply write **include =>** and then the name of the context you'd like to include from the existing context. If we reorganize our dialplan to add a **features** context, it might look something like this:

```
[users]
include => features

exten => 6001,1,Dial(SIP/demo-alice,20)
    same => n,VoiceMail(6001@vm-demo,u)

exten => 6002,1,Dial(SIP/demo-bob,20)
    same => n,VoiceMail(6002@vm-demo,u)

[features]
exten => 6000,1,Answer(500)
    same => n,Playback(hello-world)
    same => n,Hangup()

exten => 6500,1,Answer(500)
    same => n,VoiceMailMain(@vm-demo)
```



### Location of Include Statements

Please note that in the example above, we placed the include statement before extensions **6001** and **6002**. It could have just as well come after. Asterisk will always try to find a matching extension in the current context first, and only follow the include statement to a new context if there isn't anything that matches in the current context.



## Using Include Statements to Create Classes of Service

Now that we've shown the basic syntax of include statements, let's put some include statements to good use. Include statements are often used to build chains of functionality or classes of service. In this example, we're going to build several different contexts, each with its own type of outbound calling. We'll then use include statements to chain these contexts together.



### Numbering Plans

The examples in this section use patterns designed for the North American Number Plan, and may not fit your individual circumstances. Feel free to use this example as a guide as you build your own dialplan.

In these examples, we're going to assume that a seven-digit number that does not begin with a zero or a one is a local (non-toll) call. Ten-digit numbers (where neither the first or fourth digits begin with zero or one) are also treated as local calls. A one, followed by ten digits (where neither the first or fourth digits begin with zero or one) is considered a long-distance (toll) call. Again, feel free to modify these examples to fit your own particular circumstances.



### Outbound dialing

These examples assume that you have a SIP provider named `provider` configured in `sip.conf`. The examples dial out through this SIP provider using the `SIP/provider/number` syntax.

Obviously, these examples won't work unless you setup a SIP provider for outbound calls, or replace this syntax with some other type of outbound connection.

First, let's create a new context for local calls.

```
[local]
; seven-digit local numbers
exten => _NXXXXXX,1,Dial(SIP/provider/${EXTEN})

; ten-digit local numbers
exten => _NXXXXXXXX,1,Dial(SIP/provider/${EXTEN})

; emergency services (911), and other three-digit services
exten => NXX,1,Dial(SIP/provider/${EXTEN})

; if you don't find a match in this context, look in [users]
include => users
```

Remember that the variable `${EXTEN}` will get replaced with the dialed extension. For example, if Bob dials `5551212` in the `local` context, Asterisk will execute the Dial application with `SIP/provider/5551212` as the first parameter. (This syntax means "Dial out to the account named `provider` using the SIP channel driver, and dial the number `5551212`.)

Next, we'll build a long-distance context, and link it back to the `local` context with an include statement. This way, if you dial a local number and your phone's channel driver sends the call to the `longdistance` context, Asterisk will search the `local` context if it doesn't find a matching pattern in the `longdistance` context.

```
[longdistance]
; 1+ ten digit long-distance numbers
exten => _NXXXXXXXX,1,Dial(SIP/provider/${EXTEN})

; if you don't find a match in this context, look in [local]
include => local
```

Last but not least, let's add an `international` context. In North America, you dial 011 to signify that you're going to dial an international number.

```
[international]
; 1+ ten digit long-distance numbers
exten => _011.,1,Dial(SIP/provider/${EXTEN})

; if you don't find a match in this context, look in [longdistance]
include => longdistance
```

And there we have it -- a simple chain of contexts going from most privileged (international calls) down to least privileged (local calling).

At this point, you may be asking yourself, "What's the big deal? Why did we need to break them up into contexts, if they're all going out the same outbound connection?" That's a great question! The primary reason for breaking the different classes of calls into separate contexts is so that we can enforce some

security boundaries.

Do you remember what we said earlier, that the channel drivers point inbound calls at a particular context? In this case, if we point a phone at the **local** context, it could only make local and internal calls. On the other hand, if we were to point it at the **international** context, it could make international and long-distance and local and internal calls. Essentially, we've created different classes of service by chaining contexts together with include statements, and using the channel driver configuration files to point different phones at different contexts along the chain.

Please take the next few minutes and implement a series of chained contexts into your own dialplan, similar to what we've explained above. You can then change the configuration for Alice and Bob (in **sip.conf**, since they're SIP phones) to point to different contexts, and see what happens when you attempt to make various types of calls from each phone.

# Switch Statements

## Dialplan Switch Statements

The **switch** statement permits a server to share the dialplan with another server. To understand when a switch would be searched for dialplan extensions you should read the [Contexts, Extensions, and Priorities](#) section as it covers Dialplan search order.



Use with care: Reciprocal switch statements are not allowed (e.g. both A -> B and B -> A), and the switched server need to be on-line or else dialing can be severely delayed.

### Basic switch statement

As an example, with remote IAX switching you get transparent access to the remote Asterisk PBX.

```
[iaxprovider]
switch => IAX2/user:password@myserver/mycontext
```

### The lswitch statement

An "lswitch" is like a switch but is literal, in that variable substitution is not performed at load time but is passed to the switch directly (presumably to be substituted in the switch routine itself)

```
lswitch => Loopback/12${EXTEN}@othercontext
```

### The eswitch statement

An "eswitch" is like a switch but the evaluation of variable substitution is performed at runtime before being passed to the switch routine.

```
eswitch => IAX2/context@${CURSERVER}
```

## Variables

Variables are used in most programming and scripting languages. In Asterisk, we can use variables to simplify our dialplan and begin to add logic to the system. A variable is simply a container that has both a name and a value. For example, we can have a variable named **COUNT** which has a value of three. Later on, we'll show you how to route calls based on the value of a variable. Before we do that, however, let's learn a bit more about variables. The names of variables are case-sensitive, so **COUNT** is different than **Count** and **count**. Any channel variables created by Asterisk will have names that are completely upper-case, but for your own channels you can name them however you would like.

In Asterisk, we have two different types of variables: *channel variables* and *global variables*.

## Channel Variables

What's a channel variable? Read on to find out why they're important and how they'll improve your quality of life.

There are two levels of parameter evaluation done in the Asterisk dial plan in extensions.conf.

1. The first, and most frequently used, is the substitution of variable references with their values.
2. Then there are the evaluations of expressions done in `$( .. )`. This will be discussed below.

Asterisk has user-defined variables and standard variables set by various modules in Asterisk. These standard variables are listed at the end of this document.

## Parameter Quoting

```
exten => s,5,BackGround,blabla
```

The parameter (blabla) can be quoted ("blabla"). In this case, a comma does not terminate the field. However, the double quotes will be passed down to the Background command, in this example.

Also, characters special to variable substitution, expression evaluation, etc (see below), can be quoted. For example, to literally use a \$ on the string "\$1231", quote it with a preceding

. Special characters that must be quoted to be used, are [ ] \$ " \. (to write \itself, use a backslash. ).

These Double quotes and escapes are evaluated at the level of the asterisk config file parser.

Double quotes can also be used inside expressions, as discussed below.

## Setting and Substituting Channel Variables

Parameter strings can include variables. Variable names are arbitrary strings. They are stored in the respective channel structure.

To **set** a variable to a particular value, do:

```
exten => 1,2,Set(varname=value)
```

You can **substitute** the value of a variable everywhere using `${variablename}`.

Here is a simple example.

```
exten => 1,1,Set(COUNT=3)
exten => 1,n,SayNumber(${COUNT})
```

In the second line of this example, Asterisk replaces the `${COUNT}` text with the value of the `COUNT` variable, so that it ends up calling `SayNumber(3)`.

For another example, to stringwise append `$varname2` to `$varname3` and store result in `$varname1`, do:

```
exten => 1,2,Set(varname1=${varname2}${varname3})
```

There are two reference modes - reference by value and reference by name. To refer to a variable with its name (as an argument to a function that requires a variable), just write the name. To refer to the variable's value, enclose it inside `${}`. For example, `Set` takes as the first argument (before the `=`) a variable name, so:

```
exten => 1,2,Set(varname1=varname2)
exten => 1,3,Set(${varname1}=value)
```

The above dialplan stores to the variable "varname1" the value "varname2" and to variable "varname2" the value "value".

In fact, everything contained `${here}` is just replaced with the value of the variable "here".

## Variable Inheritance

### Single Inheritance

Variable names which are prefixed by "\_" (one underbar character) will be inherited to channels that are created in the process of servicing the original channel in which the variable was set. When the inheritance takes place, the prefix will be removed in the channel inheriting the variable. Meaning it will not be inherited any further than a single level, that is one child channel.

```
exten = 1234,1,Set(_FOO=bar)
```

### Multiple Inheritance

If the name is prefixed by "\_\_" (two underbar characters) in the channel, then the variable is inherited and the "\_\_" will remain intact in the new channel. Therefore any channels then created by the new channel will also receive the variable with "\_\_", continuing the inheritance indefinitely.

In the Dialplan, all references to these variables refer to the same variable, regardless of having a prefix or not. Note that setting any version of the variable removes any other version of the variable, regardless of prefix.

```
exten = 1234,1,Set(__FOO=bar)
```



## Selecting Characters from Variables

The format for selecting characters from a variable can be expressed as:

```
${variable_name[:offset[:length]]}
```

If you want to select the first N characters from the string assigned to a variable, simply append a colon and the number of characters to skip from the beginning of the string to the variable name.

```
; Remove the first character of extension, save in "number" variable
exten => _9X.,1,Set(number=${EXTEN:1})
```

Assuming we've dialed 918005551234, the value saved to the 'number' variable would be 18005551234. This is useful in situations when we require users to dial a number to access an outside line, but do not wish to pass the first digit.

If you use a negative offset number, Asterisk starts counting from the end of the string and then selects everything after the new position. The following example will save the numbers 1234 to the 'number' variable, still assuming we've dialed 918005551234.

```
; Remove everything before the last four digits of the dialed string
exten => _9X.,1,Set(number=${EXTEN:-4})
```

We can also limit the number of characters from our offset position that we wish to use. This is done by appending a second colon and length value to the variable name. The following example will save the numbers 555 to the 'number' variable.

```
; Only save the middle numbers 555 from the string 918005551234
exten => _9X.,1,Set(number=${EXTEN:5:3})
```

The length value can also be used in conjunction with a negative offset. This may be useful if the length of the string is unknown, but the trailing digits are. The following example will save the numbers 555 to the 'number' variable, even if the string starts with more characters than expected (unlike the previous example).

```
; Save the numbers 555 to the 'number' variable
exten => _9X.,1,Set(number=${EXTEN:-7:3})
```

If a negative length value is entered, Asterisk will remove that many characters from the end of the string.

```
; Set pin to everything but the trailing #.
exten => _XXXX#,1,Set(pin=${EXTEN:0:-1})
```

## Asterisk Standard Channel Variables

There are a number of variables that are defined or read by Asterisk. Here is a listing of them. More information is available in each application's help text. All these variables are in UPPER CASE only.

Variables marked with a \* are builtin functions and can't be set, only read in the dialplan. Writes to such variables are silently ignored.

### Variables present in Asterisk 1.8 and forward:

- `$(CDR(accountcode))` \* - Account code (if specified)
- `$(BLINDTRANSFER)` - The name of the channel on the other side of a blind transfer
- `$(BRIDGEPEER)` - Bridged peer
- `$(BRIDGEPVTCALLID)` - Bridged peer PVT call ID (SIP Call ID if a SIP call)
- `$(CALLERID(ani))` \* - Caller ANI (PRI channels)
- `$(CALLERID(ani2))` \* - ANI2 (Info digits) also called Originating line information or OLI
- `$(CALLERID(all))` \* - Caller ID
- `$(CALLERID(dnid))` \* - Dialed Number Identifier
- `$(CALLERID(name))` \* - Caller ID Name only
- `$(CALLERID(num))` \* - Caller ID Number only
- `$(CALLERID(rdnis))` \* - Redirected Dial Number ID Service
- `$(CALLINGANI2)` \* - Caller ANI2 (PRI channels)
- `$(CALLINGPRES)` \* - Caller ID presentation for incoming calls (PRI channels)
- `$(CALLINGTNS)` \* - Transit Network Selector (PRI channels)
- `$(CALLINGTON)` \* - Caller Type of Number (PRI channels)
- `$(CHANNEL)` \* - Current channel name
- `$(CONTEXT)` \* - Current context
- `$(DATETIME)` \* - Current date time in the format: DDMMYYYY-HH:MM:SS (Deprecated; use `$(STRFTIME($EPOCH,,"%d%m%Y-%H:%M:%S))`)
- `$(DB_RESULT)` - Result value of `DB_EXISTS()` dial plan function
- `$(EPOCH)` \* - Current unix style epoch
- `$(EXTEN)` \* - Current extension
- `$(ENV(VAR))` - Environmental variable VAR
- `$(GOTO_ON_BLINDXFR)` - Transfer to the specified context/extension/priority after a blind transfer (use ^ characters in place of | to separate context/extension/priority when setting this variable from the dialplan)
- `$(HANGUPCAUSE)` \* - Asterisk cause of hangup (inbound/outbound)
- `$(HINT)` \* - Channel hints for this extension
- `$(HINTNAME)` \* - Suggested Caller\*ID name for this extension
- `$(INVALID_EXTEN)` - The invalid called extension (used in the "i" extension)
- `$(LANGUAGE)` \* - Current language (Deprecated; use `$(CHANNEL(language))`)
- `$(LEN(VAR))` - String length of VAR (integer)
- `$(PRIORITY)` \* - Current priority in the dialplan
- `$(PRIREDIRECTREASON)` - Reason for redirect on PRI, if a call was directed
- `$(TIMESTAMP)` \* - Current date time in the format: YYYYMMDD-HHMMSS (Deprecated; use `$(STRFTIME($EPOCH,,"%Y%m%d-%H%M%S))`)
- `$(TRANSFER_CONTEXT)` - Context for transferred calls
- `$(FORWARD_CONTEXT)` - Context for forwarded calls
- `$(DYNAMIC_PEERNAME)` - The name of the channel on the other side when a dynamic feature is used.
- `$(DYNAMIC_FEATURENAME)` The name of the last triggered dynamic feature.
- `$(UNIQUEID)` \* - Current call unique identifier
- `$(SYSTEMNAME)` \* - value of the systemname option of asterisk.conf
- `$(ENTITYID)` \* - Global Entity ID set automatically, or from asterisk.conf

### Variables present in Asterisk 11 and forward:

- `$(AGIEXITONHANGUP)` - set to 1 to force the behavior of a call to AGI to behave as it did in 1.4, where the AGI script would exit immediately on detecting a channel hangup
- `$(CALENDAR_SUCCESS)` \* - Status of the `CALENDAR_WRITE` function. Set to 1 if the function completed successfully; 0 otherwise.
- `$(SIP_RECVADDR)` \* - the address a SIP MESSAGE request was received from
- `$(VOICEMAIL_PLAYBACKSTATUS)` \* - Status of the `VoiceMailPlayMsg` application. `SUCCESS` if the voicemail was played back successfully, `FAILED` otherwise

## Application return values

Many applications return the result in a variable that you read to get the result of the application. These status fields are unique for each application. For the various status values, see each application's help text.

- `#{AGISTATUS} * agi()`
- `#{AQMSTATUS} * addqueuemember()`
- `#{AVAILSTATUS} * chanisavail()`
- `#{CHECKGROUPSTATUS} * checkgroup()`
- `#{CHECKMD5STATUS} * checkmd5()`
- `#{CPLAYBACKSTATUS} * controlplayback()`
- `#{DIALSTATUS} * dial()`
- `#{DBGETSTATUS} * dbget()`
- `#{ENUMSTATUS} * enumlookup()`
- `#{HASVMSTATUS} * hasnewvoicemail()`
- `#{LOOKUPBLSTATUS} * lookupblacklist()`
- `#{OSPAUTHSTATUS} * ospauth()`
- `#{OSPLOOKUPSTATUS} * osplookup()`
- `#{OSPNEXTSTATUS} * ospnext()`
- `#{OSPFINISHSTATUS} * ospfinish()`
- `#{PARKEDAT} * parkandannounce()`
- `#{PLAYBACKSTATUS} * playback()`
- `#{PQMSTATUS} * pausequeuemember()`
- `#{PRIVACYMGRSTATUS} * privacymanager()`
- `#{QUEUESTATUS} * queue()`
- `#{RQMSTATUS} * removequeuemember()`
- `#{SENDIMAGESTATUS} * sendimage()`
- `#{SENDTEXTSTATUS} * sendtext()`
- `#{SENDURLSTATUS} * sendurl()`
- `#{SYSTEMSTATUS} * system()`
- `#{TRANSFERSTATUS} * transfer()`
- `#{TXTCIDNAMESTATUS} * txtcidname()`
- `#{UPQMSTATUS} * unpaqueuemember()`
- `#{VMSTATUS} * voicemail()`
- `#{VMBOXEXISTSSTATUS} * vmboxexists()`
- `#{WAITSTATUS} * waitforsilence()`

## Various application variables

- `${CURL}` - Resulting page content for `CURL()`
- `${ENUM}` - Result of application `EnumLookup()`
- `${EXITCONTEXT}` - Context to exit to in IVR menu (`Background()`) or in the `RetryDial()` application
- `${MONITOR}` - Set to "TRUE" if the channel is/has been monitored (`app monitor()`)
- `${MONITOR_EXEC}` - Application to execute after monitoring a call
- `${MONITOR_EXEC_ARGS}` - Arguments to application
- `${MONITOR_FILENAME}` - File for monitoring (recording) calls in queue
- `${QUEUE_PPIO}` - Queue priority
- `${QUEUE_MAX_PENALTY}` - Maximum member penalty allowed to answer caller
- `${QUEUE_MIN_PENALTY}` - Minimum member penalty allowed to answer caller
- `${QUEUESTATUS}` - Status of the call, one of: (TIMEOUT | FULL | JOINEMPTY | LEAVEEMPTY | JOINUNAVAIL | LEAVEUNAVAIL)
- `${QUEUEPOSITION}` - When a caller is removed from a queue, his current position is logged in this variable. If the value is 0, then this means that the caller was serviced by a queue member. If non-zero, then this was the position in the queue the caller was in when he left.
- `${RECORDED_FILE}` - Recorded file in `record()`
- `${TALK_DETECTED}` - Result from `talkdetect()`
- `${TOUCH_MONITOR}` - The filename base to use with Touch Monitor (auto record)
- `${TOUCH_MONITOR_PREF}` - The prefix for automonitor recording filenames.
- `${TOUCH_MONITOR_FORMAT}` - The audio format to use with Touch Monitor (auto record)
- `${TOUCH_MONITOR_OUTPUT}` - Recorded file from Touch Monitor (auto record)
- `${TOUCH_MONITOR_MESSAGE_START}` - Recorded file to play for both channels at start of monitoring session
- `${TOUCH_MONITOR_MESSAGE_STOP}` - Recorded file to play for both channels at end of monitoring session
- `${TOUCH_MIXMONITOR}` - The filename base to use with Touch MixMonitor (auto record)
- `${TOUCH_MIXMONITOR_FORMAT}` - The audio format to use with Touch MixMonitor (auto record)
- `${TOUCH_MIXMONITOR_OUTPUT}` - Recorded file from Touch MixMonitor (auto record)
- `${TXTCIDNAME}` - Result of application `TXTCIDName`
- `${VPB_GETDTMF}` - `chan_vpb`

## MeetMe Channel Variables

- `${MEETME_RECORDINGFILE}` - Name of file for recording a conference with the "r" option
- `${MEETME_RECORDINGFORMAT}` - Format of file to be recorded
- `${MEETME_EXIT_CONTEXT}` - Context for exit out of meetme meeting
- `${MEETME_AGI_BACKGROUND}` - AGI script for Meetme (DAHDI only)
- `${MEETMESECS}` \* - Number of seconds a user participated in a MeetMe conference
- `${CONF_LIMIT_TIMEOUT_FILE}` - File to play when time is up. Used with the L() option.
- `${CONF_LIMIT_WARNING_FILE}` - File to play as warning if 'y' is defined. The default is to say the time remaining. Used with the L() option.
- `${MEETMEBOOKID}` \* - This variable exposes the bookid column for a realtime configured conference bridge.
- `${MEETME_EXIT_KEY}` - DTMF key that will allow a user to leave a conference

## VoiceMail Channel Variables

- `${VM_CATEGORY}` - Sets voicemail category
- `${VM_NAME}` \* - Full name in voicemail
- `${VM_DUR}` \* - Voicemail duration
- `${VM_MSGNUM}` \* - Number of voicemail message in mailbox
- `${VM_CALLERID}` \* - Voicemail Caller ID (Person leaving vm)
- `${VM_CIDNAME}` \* - Voicemail Caller ID Name
- `${VM_CIDNUM}` \* - Voicemail Caller ID Number
- `${VM_DATE}` \* - Voicemail Date
- `${VM_MESSAGEFILE}` \* - Path to message left by caller

## **VMAuthenticate Channel Variables**

- `${AUTH_MAILBOX} *` - Authenticated mailbox
- `${AUTH_CONTEXT} *` - Authenticated mailbox context

## DUNDiLookup Channel Variables

- `$(DUNDTECH) *` - The Technology of the result from a call to DUNDiLookup()
- `$(DUNDDEST) *` - The Destination of the result from a call to DUNDiLookup()



## chan\_dahdi Channel Variables

- `${ANI2}` \* - The ANI2 Code provided by the network on the incoming call. (ie, Code 29 identifies call as a Prison/Inmate Call)
- `${CALLTYPE}` \* - Type of call (Speech, Digital, etc)
- `${CALLEDTON}` \* - Type of number for incoming PRI extension i.e. 0=unknown, 1=international, 2=domestic, 3=net\_specific, 4=subscriber, 6=abbreviated, 7=reserved
- `${CALLINGSUBADDR}` \* - Caller's PRI Subaddress
- `${FAXEXTEN}` \* - The extension called before being redirected to "fax"
- `${PRIREDIRECTREASON}` \* - Reason for redirect, if a call was directed
- `${SMDI_VM_TYPE}` \* - When an call is received with an SMDI message, the 'type' of message 'b' or 'u'

## chan\_sip Channel Variables

- `$(SIPCALLID)` \* - SIP Call-ID: header verbatim (for logging or CDR matching)
- `$(SIPDOMAIN)` \* - SIP destination domain of an inbound call (if appropriate)
- `$(SIPFROMUSER)` - Set user portion of From header on outbound calls
- `$(SIPFROMDOMAIN)` - Set SIP domain portion of From header on outbound calls
- `$(SIPUSERAGENT)` \* - SIP user agent (deprecated)
- `$(SIPURI)` \* - SIP uri
- `$(SIP_MAX_FORWARDS)` - Set the value of the Max-Forwards header for outbound call
- `$(SIP_CODEEC)` - Set the SIP codec for an inbound call
- `$(SIP_CODEEC_INBOUND)` - Set the SIP codec for an inbound call
- `$(SIP_CODEEC_OUTBOUND)` - Set the SIP codec for an outbound call
- `$(SIP_URI_OPTIONS)` \* - additional options to add to the URI for an outgoing call
- `$(RTPAUDIOQOS)` - RTCP QoS report for the audio of this call
- `$(RTPVIDEOQOS)` - RTCP QoS report for the video of this call

## chan\_agent Channel Variables

- `$(AGENTMAXLOGINTRIES)` - Set the maximum number of failed logins
- `$(AGENTUPDATECDR)` - Whether to update the CDR record with Agent channel data
- `$(AGENTGOODBYE)` - Sound file to use for "Good Bye" when agent logs out
- `$(AGENTACKCALL)` - Whether the agent should acknowledge the incoming call
- `$(AGENTAUTOLOGOFF)` - Auto logging off for an agent
- `$(AGENTWRAPUPTIME)` - Setting the time for wrapup between incoming calls
- `$(AGENTNUMBER) *` - Agent number (username) set at login
- `$(AGENTSTATUS) *` - Status of login ( fail | on | off )
- `$(AGENTEXTEN) *` - Extension for logged in agent

## Dial Channel Variables

- `${DIALEDPEERNAME}` \* - Dialed peer name
- `${DIALEDPEERNUMBER}` \* - Dialed peer number
- `${DIALEDTIME}` \* - Time for the call (seconds). Is only set if call was answered.
- `${ANSWEREDTIME}` \* - Time from answer to hangup (seconds)
- `${DIALSTATUS}` \* - Status of the call, one of: (CHANUNAVAIL | CONGESTION | BUSY | NOANSWER | ANSWER | CANCEL | DONTCALL | TORTURE)
- `${DYNAMIC_FEATURES}` \* - The list of features (from the [applicationmap](#) section of `features.conf`) to activate during the call, with feature names separated by '#' characters
- `${LIMIT_PLAYAUDIO_CALLER}` - Soundfile for call limits
- `${LIMIT_PLAYAUDIO_CALLEE}` - Soundfile for call limits
- `${LIMIT_WARNING_FILE}` - Soundfile for call limits
- `${LIMIT_TIMEOUT_FILE}` - Soundfile for call limits
- `${LIMIT_CONNECT_FILE}` - Soundfile for call limits
- `${OUTBOUND_GROUP}` - Default groups for peer channels (as in `SetGroup`) \* See "show application dial" for more information

## Chanisavail() Channel Variables

- `#{AVAILCHAN}` \* - the name of the available channel if one was found
- `#{AVAILORIGCHAN}` \* - the canonical channel name that was used to create the channel
- `#{AVAILSTATUS}` \* - Status of requested channel

## Dialplan Macros Channel Variables

- `$(MACRO_EXTEN) *` - The calling extensions
- `$(MACRO_CONTEXT) *` - The calling context
- `$(MACRO_PRIORITY) *` - The calling priority
- `$(MACRO_OFFSET)` - Offset to add to priority at return from macro

## ChanSpy Channel Variables

- `#{SPYGROUP} * - A ':'` (colon) separated list of group names. (To be set on spied on channel and matched against the `g(grp)` option)

## Open Settlement Protocol (OSP) Channel Variables

- `$(OSPINHANDLE)` - The inbound call OSP transaction handle.
- `$(OSPINOKEN)` - The inbound OSP token.
- `$(OSPINMELIMIT)` - The inbound call duration limit in seconds.
- `$(OSPINPEERIP)` - The last hop IP address.
- `$(OSPINNETWORKID)` - The inbound source network ID.
- `$(OSPINNPRN)` - The inbound routing number.
- `$(OSPINNPCIC)` - The inbound carrier identification code.
- `$(OSPINNPDI)` - The inbound number portability database dip indicator.
- `$(OSPINSPID)` - The inbound service provider identity.
- `$(OSPINOCN)` - The inbound operator company number.
- `$(OSPINSPN)` - The inbound service provider name.
- `$(OSPINALTSPN)` - The inbound alternate service provider name.
- `$(OSPINMCC)` - The inbound mobile country code.
- `$(OSPINMNC)` - The inbound mobile network code.
- `$(OSPINDIVUSER)` - The inbound Diversion header user part.
- `$(OSPINDIVHOST)` - The inbound Diversion header host part.
- `$(OSPINTOHOST)` - The inbound To header host part.
- `$(OSPINCUSTOMINFOn)` - The inbound custom information. Where n is the index beginning with 1 upto 8.
- `$(OSPOUTHANDLE)` - The outbound call OSP transaction handle.
- `$(OSPOUTOKEN)` - The outbound OSP token.
- `$(OSPOUTMELIMIT)` - The outbound call duration limit in seconds.
- `$(OSPOUTTECH)` - The outbound channel technology.
- `$(OSPOUTCALLIDTYPES)` - The outbound Call-ID types.
- `$(OSPOUTCALLID)` - The outbound Call-ID. Only for H.323.
- `$(OSPDESTINATION)` - The destination IP address.
- `$(OSPDESTREMAILS)` - The number of remained destinations.
- `$(OSPOUTCALLING)` - The outbound calling number.
- `$(OSPOUTCALLED)` - The outbound called number.
- `$(OSPOUTNETWORKID)` - The outbound destination network ID.
- `$(OSPOUTNPRN)` - The outbound routing number.
- `$(OSPOUTNPCIC)` - The outbound carrier identification code.
- `$(OSPOUTNPDI)` - The outbound number portability database dip indicator.
- `$(OSPOUTSPID)` - The outbound service provider identity.
- `$(OSPOUTOCN)` - The outbound operator company number.
- `$(OSPOUTSPN)` - The outbound service provider name.
- `$(OSPOUTALTSPN)` - The outbound alternate service provider name.
- `$(OSPOUTMCC)` - The outbound mobile country code.
- `$(OSPOUTMNC)` - The outbound mobile network code.
- `$(OSPDIALSTR)` - The outbound Dial command string.
- `$(OSPINAUDIOQOS)` - The inbound call leg audio QoS string.
- `$(OSPOUTAUDIOQOS)` - The outbound call leg audio QoS string.



## Digit Manipulation Channel Variables

- `${REDIRECTING_CALLEE_SEND_MACRO}`  
Macro to call before sending a redirecting update to the callee
- `${REDIRECTING_CALLEE_SEND_MACRO_ARGS}`  
Arguments to pass to `${REDIRECTING_CALLEE_SEND_MACRO}`

—

- `${REDIRECTING_CALLER_SEND_MACRO}`  
Macro to call before sending a redirecting update to the caller
- `${REDIRECTING_CALLER_SEND_MACRO_ARGS}`  
Arguments to pass to `${REDIRECTING_CALLER_SEND_MACRO}`

---

- `${CONNECTED_LINE_CALLEE_SEND_MACRO}`  
Macro to call before sending a connected line update to the callee
- `${CONNECTED_LINE_CALLEE_SEND_MACRO_ARGS}`  
Arguments to pass to `${CONNECTED_LINE_CALLEE_SEND_MACRO}`

—

- `${CONNECTED_LINE_CALLER_SEND_MACRO}`  
Macro to call before sending a connected line update to the caller
- `${CONNECTED_LINE_CALLER_SEND_MACRO_ARGS}`  
Arguments to pass to `${CONNECTED_LINE_CALLER_SEND_MACRO}`

## Case Sensitivity

Case sensitivity of channel variables in Asterisk is dependent on the version of Asterisk in use.

### Versions prior to Asterisk 12

This includes versions

- Asterisk 1.0.X
- Asterisk 1.2.X
- Asterisk 1.4.X
- Asterisk 1.6.0.X
- Asterisk 1.6.1.X
- Asterisk 1.6.2.X
- Asterisk 1.8.X
- Asterisk 10.X
- Asterisk 11.X

These versions of Asterisk follow these three rules:

- Variables evaluated in the dialplan are **case-insensitive**
- Variables evaluated within Asterisk's internals are **case-sensitive**
- Built-in variables are **case-sensitive**

This is best illustrated through the following examples

#### Example 1: A user-set variable

In this example, the user retrieves a value from the AstDB and then uses it as the destination for a Dial command.

```
[default]
exten => 1000,1,Set(DEST=${DB(egg/salad)})
same => n,Dial(${DEST},15)
```

Since the DEST variable is set and evaluated in the dialplan, its evaluation is case-insensitive. Thus the following would be equivalent:

```
exten => 1000,1,Set(DEST=${DB(egg/salad)})
same => n,Dial(${dest},15)
```

As would this:

```
exten => 1000,1,Set(DeSt=${DB(egg/salad)})
same => n,Dial(${dEsT},15)
```

#### Example 2: Using a built-in variable

In this example, the user wishes to use a built-in variable in order to determine the destination for a call.

```
exten => _X.,1,Dial(SIP/${EXTEN})
```

Since the variable EXTEN is a built-in variable, the following would **not** be equivalent:

```
exten => _X.,1,Dial(SIP/${exten})
```

The lowercase exten variable would evaluate to an empty string since no previous value was set for exten.

#### Example 3: A variable used internally by Asterisk

In this example, the user wishes to suggest to the SIP channel driver what codec to use on the call.

```
exten => 1000,Set(SIP_CODEC=g729)
same => n,Dial(SIP/1000,15)
```

SIP\_CODEC is set in the dialplan, but it gets evaluated inside of Asterisk, so the evaluation is case-sensitive. Thus the following dialplan would not be equivalent:

```
exten => 1000,Set(sip_codec=g729)
same => n,Dial(SIP/1000,15)
```

This can lead to some rather confusing situations. Consider that a user wrote the following dialplan. He intended to set the variable SIP\_CODEC but instead made a typo:

```
exten => 1000,Set(SIP_CODEc=g729)
same => n,Dial(SIP/1000,15)
```

As has already been discussed, this is not equivalent to using `SIP_CODEc`. The user looks over his dialplan and does not notice the typo. As a way of debugging, he decides to place a `NoOp` in the dialplan:

```
exten => 1000,Set(SIP_CODEc=g729)
same => n,NoOp(${SIP_CODEc})
same => n,Dial(SIP/1000,15)
```

When the user checks the verbose logs, he sees that the second priority has evaluated `SIP_CODEc` to be "g729". This is because the evaluation in the dialplan was done case-insensitively.

### Asterisk 12 and above

Due to potential confusion stemming from the policy, for Asterisk 12, it was proposed that variables should be evaluated consistently. E-mails were sent to the [Asterisk-developers](#) and [Asterisk-users](#) lists about whether variables should be evaluated case-sensitively or case-insensitively. The majority opinion swayed towards case-sensitive evaluation. Thus in Asterisk 12, all variable evaluation, whether done in the dialplan or internally, will be case-sensitive.

For those who are upgrading to Asterisk 12 from a previous version, be absolutely sure that your variables are used consistently throughout your dialplan.

## Global Variables Basics

Global variables are variables that don't live on one particular channel — they pertain to all calls on the system. They have global scope. There are two ways to set a global variable. The first is to declare the variable in the **[globals]** section of **extensions.conf**, like this:

```
[globals]
MYGLOBALVAR=somevalue
```

You can also set global variables from dialplan logic using the **GLOBAL()** dialplan function along with the **Set()** application. Simply use the syntax:

```
exten=>6124,1,Set(GLOBAL(MYGLOBALVAR)=somevalue)
```

To retrieve the value of a global channel variable, use the same syntax as you would if you were retrieving the value of a channel variable.

## Manipulating Variables Basics

It's often useful to do string manipulation on a variable. Let's say, for example, that we have a variable named **NUMBER** which represents a number we'd like to call, and we want to strip off the first digit before dialing the number. Asterisk provides a special syntax for doing just that, which looks like **`\${variable[:skip[docs::length]}`**.

The optional **skip** field tells Asterisk how many digits to strip off the front of the value. For example, if **NUMBER** were set to a value of **98765**, then **`\${NUMBER:2}`** would tell Asterisk to remove the first two digits and return **765**.

If the skip field is negative, Asterisk will instead return the specified number of digits from the end of the number. As an example, if **NUMBER** were set to a value of **98765**, then **`\${NUMBER:-2}`** would tell Asterisk to return the last two digits of the variable, or **65**.

If the optional **length** field is set, Asterisk will return at most the specified number of digits. As an example, if **NUMBER** were set to a value of **98765**, then **`\${NUMBER:0:3}`** would tell Asterisk not to skip any characters in the beginning, but to then return only the three characters from that point, or **987**. By that same token, **`\${NUMBER:1:3}`** would return **876**.

## Using the CONTEXT, EXTEN, PRIORITY, UNIQUEID, and CHANNEL Variables

Now that you've learned a bit about variables, let's look at a few of the variables that Asterisk automatically creates.

Asterisk creates channel variables named **CONTEXT**, **EXTEN**, and **PRIORITY** which contain the current context, extension, and priority. We'll use them in pattern matching (below), as well as when we talk about macros in [Section 308.10. Macros](#). Until then, let's show a trivial example of using **\${EXTEN}** to read back the current extension number.

```
exten=>6123,1,SayNumber(${EXTEN})
```

If you were to add this extension to the **[docs:users]** context of your dialplan and reload the dialplan, you could call extension **6123** and hear Asterisk read back the extension number to you.

Another channel variable that Asterisk automatically creates is the **UNIQUEID** variable. Each channel within Asterisk receives a unique identifier, and that identifier is stored in the **UNIQUEID** variable. The **UNIQUEID** is in the form of **1267568856.11**, where **1267568856** is the Unix epoch, and **11** shows that this is the eleventh call on the Asterisk system since it was last restarted.

Last but not least, we should mention the **CHANNEL** variable. In addition to a unique identifier, each channel is also given a channel name and that channel name is set in the **CHANNEL** variable. A SIP call, for example, might have a channel name that looks like **SIP/george-0000003b**, for example.

## Pattern Matching

The next concept we'll cover is called *pattern matching*. Pattern matching allows us to create extension patterns in our dialplan that match more than one possible dialed number. Pattern matching saves us from having to create an extension in the dialplan for every possible number that might be dialed.

When Alice dials a number on her phone, Asterisk first looks for an extension (in the context specified by the channel driver configuration) that matches exactly what Alice dialed. If there's no exact match, Asterisk then looks for a pattern that matches. After we show the syntax and some basic examples of pattern matching, we'll explain how Asterisk finds the best match if there are two or more patterns which match the dialed number.

## Special Characters Used in Pattern Matching

Pattern matches always begin with an underscore. This is how Asterisk recognizes that the extension is a pattern and not just an extension with a funny name. Within the pattern, we use various letters and characters to represent sets or ranges of numbers. Here are the most common letters:

### X

The letter **X** or **x** represents a single digit from 0 to 9.

### Z

The letter **Z** or **z** represents any digit from 1 to 9.

- [Special Characters Used in Pattern Matching](#)
  - X
  - Z
  - N
- [Character Sets](#)
- [Other Special Characters](#)
- [Order of Pattern Matching](#)
- [Matching on Caller ID](#)

### N

The letter **N** or **n** represents a single digit from 2 to 9.

Now let's look at a sample pattern. If you wanted to match all four-digit numbers that had the first two digits as six and four, you would create an extension that looks like:

```
exten => _64XX,1,SayDigits(${EXTEN})
```

In this example, each **X** represents a single digit, with any value from zero to nine. We're essentially saying "The first digit must be a six, the second digit must be a four, the third digit can be anything from zero to nine, and the fourth digit can be anything from zero to nine".

## Character Sets

If we want to be more specific about a range of numbers, we can put those numbers or number ranges in square brackets to define a character set. For example, what if we wanted the second digit to be either a three or a four? One way would be to create two patterns (**\_64XX** and **\_63XX**), but a more compact method would be to do **\_6[34]XX**. This specifies that the first digit must be a six, the second digit can be either a three or a four, and that the last two digits can be anything from zero to nine.

You can also use ranges within square brackets. For example, **[1-468]** would match a single digit from one through four or six or eight. It does not match any number from one to four hundred sixty-eight!



The X, N, and Z convenience notations mentioned earlier have no special meaning within a set.

The only characters with special meaning within a set are the '-' character, to define a range between two characters, the '\' character to escape a special character available within a set, and the ']' character which closes the set. The treatment of the '\' character in pattern matching is somewhat haphazard and may not escape any special character meaning correctly.

## Other Special Characters

Within Asterisk patterns, we can also use a couple of other characters to represent ranges of numbers. The period character (.) at the end of a pattern matches one or more remaining **characters**. You put it at the end of a pattern when you want to match extensions of an indeterminate length. As an example, the pattern **\_9876.** would match any number that began with **9876** and had at least one more character or digit.

The exclamation mark (!) character is similar to the period and matches zero or more remaining characters. It is used in overlap dialing to dial through Asterisk. For example, **\_9876!** would match any number that began with **9876** including **9876**, and would respond that the number was complete as soon

as there was an unambiguous match.



Asterisk treats a period or exclamation mark as the end of a pattern. If you want a period or exclamation mark in your pattern as a plain character you should put it into a character set: `[.]` or `[!]`.



### Be Careful With Wildcards in Pattern Matches

Please be extremely cautious when using the period and exclamation mark characters in your pattern matches. They match more than just digits. They match on characters. If you're not careful to filter the input from your callers, a malicious caller might try to use these wildcards to bypass security boundaries on your system.

For a more complete explanation of this topic and how you can protect yourself, please refer to the [README-SERIOUSLY.bestpractices.txt](#) file in the Asterisk source code.

## Order of Pattern Matching

Now let's show what happens when there is more than one pattern that matches the dialed number. How does Asterisk know which pattern to choose as the best match?

Asterisk uses a simple set of rules to sort the extensions and patterns so that the best match is found first. The best match is simply the most specific pattern. The sorting rules are:

1. The dash (-) character is ignored in extensions and patterns except when it is used in a pattern to specify a range in a character set. It has no effect in matching or sorting extensions.
2. Non-pattern extensions are sorted in ASCII sort order before patterns.
3. Patterns are sorted by the most constrained character set per digit. By most constrained, we mean the pattern that has the fewest possible matches for a digit. As an example, the **N** character has eight possible matches (two through nine), while **X** has ten possible matches (zero through nine) so **N** sorts first.
4. Character sets that have the same number of characters are sorted in ASCII sort order as if the sets were strings of the set characters. As an example, **X** is **0123456789** and **[a-j]** is **abcdefghij** so **X** sorts first. This sort ordering is important if the character sets overlap as with **[0-4]** and **[4-8]**.
5. The period (.) wildcard sorts after character sets.
6. The exclamation mark (!) wildcard sorts after the period wildcard.

Let's look at an example to better understand how this works. Let's assume Alice dials extension **6421**, and she has the following patterns in her dialplan:

```
exten => _6XX1,1,SayAlpha(A)
exten => _64XX,1,SayAlpha(B)
exten => _640X,1,SayAlpha(C)
exten => _6.,1,SayAlpha(D)
exten => _64NX,1,SayAlpha(E)
exten => _6[45]NX,1,SayAlpha(F)
exten => _6[34]NX,1,SayAlpha(G)
```

Can you tell (without reading ahead) which one would match?

Using the sorting rules explained above, the extensions sort as follows:

```
_640X sorts before _64NX because of rule 3 at position 4. (0 before N)
_64NX sorts before _64XX because of rule 3 at position 4. (N before X)
_64XX sorts before _6[34]NX because of rule 3 at position 3. (4 before [34])
_6[34]NX sorts before _6[45]NX because of rule 4 at position 3. ([34] before [45])
_6[45]NX sorts before _6XX1 because of rule 3 at position 3. ([45] before X)
_6XX1 sorts before _6. because of rule 5 at position 3. (X before .)
```

### Sorted extensions

```
exten => _640X,1,SayAlpha(C)
exten => _64NX,1,SayAlpha(E)
exten => _64XX,1,SayAlpha(B)
exten => _6[34]NX,1,SayAlpha(G)
exten => _6[45]NX,1,SayAlpha(F)
exten => _6XX1,1,SayAlpha(A)
exten => _6.,1,SayAlpha(D)
```

When Alice dials **6421**, Asterisk searches through its list of sorted extensions and uses the first matching extension. In this case **\_64NX** is found.

To verify that Asterisk actually does sort the extensions in the manner that we've shown, add the following extensions to the **[users]** context of your own dialplan.



```

exten => _6XX1,1,SayAlpha(A)
exten => _64XX,1,SayAlpha(B)
exten => _640X,1,SayAlpha(C)
exten => _6.,1,SayAlpha(D)
exten => _64NX,1,SayAlpha(E)
exten => _6[45]NX,1,SayAlpha(F)
exten => _6[34]NX,1,SayAlpha(G)

```

Reload the dialplan, and then type **dialplan show 6421@users** at the Asterisk CLI. Asterisk will show you all extensions that match in the **[users]** context. If you were to dial extension **6421** in the **[users]** context the first found extension will execute.

```

server*CLI> dialplan show 6421@users
[ Context 'users' created by 'pbx_config' ]
'_64NX' =>      1. SayAlpha(E)                [pbx_config]
'_64XX' =>      1. SayAlpha(B)                [pbx_config]
'_6[34]NX' =>   1. SayAlpha(G)                [pbx_config]
'_6[45]NX' =>   1. SayAlpha(F)                [pbx_config]
'_6XX1' =>      1. SayAlpha(A)                [pbx_config]
'_6.' =>        1. SayAlpha(D)                [pbx_config]

-- 6 extensions (6 priorities) in 1 context. --

```

```

server*CLI> dialplan show users
[ Context 'users' created by 'pbx_config' ]
'_640X' =>      1. SayAlpha(C)                [pbx_config]
'_64NX' =>      1. SayAlpha(E)                [pbx_config]
'_64XX' =>      1. SayAlpha(B)                [pbx_config]
'_6[34]NX' =>   1. SayAlpha(G)                [pbx_config]
'_6[45]NX' =>   1. SayAlpha(F)                [pbx_config]
'_6XX1' =>      1. SayAlpha(A)                [pbx_config]
'_6.' =>        1. SayAlpha(D)                [pbx_config]

-- 7 extensions (7 priorities) in 1 context. --

```

You can dial extension **6421** to try it out on your own.



### Be Careful with Pattern Matching

Please be aware that because of the way auto-fallthrough works, if Asterisk can't find the next priority number for the current extension or pattern match, it will also look for that same priority in a less specific pattern match. Consider the following example:

```

exten => 6410,1,SayDigits(987)
exten => _641X,1,SayDigits(12345)
exten => _641X,n,SayDigits(54321)

```

If you were to dial extension **6410**, you'd hear "nine eight seven five four three two one".

We strongly recommend you make the **Hangup()** application be the last priority of any extension to avoid this behaviour, unless you purposely want to fall through to a less specific match.

## Matching on Caller ID

Within an extension handler, it is also possible to match based upon the Caller ID of the incoming channel by appending a forward slash to the dialed extension or pattern, followed by a Caller ID pattern to be matched. Consider the following example, featuring phones with Caller IDs of 101, 102 and 103.

```

exten => 306,1,NoOp()
same => n,Background(goodbye)
same => n,Hangup()

exten => 306/_101,1,NoOp()
same => n,Background(year)
same => n,Hangup()

exten => 306/_102,1,NoOp()
same => n,Background(beep)
same => n,Hangup()

```

The phone with Caller ID 101, when dialing 306, will hear the prompt "year" and will be hung up. The phone with Caller ID 102, when dialing 306, will hear the "beep" sound and will be hung up. The phone with Caller ID 103, or any other caller, when dialing 306, will hear the "goodbye" prompt and will be hung up.

## Subroutines

Subroutines in Asterisk are defined similarly to standard dialplan [contexts](#) and are referred to as [Macros](#) and [Gosubs](#). They are invoked via the `Macro` and `Gosub` applications, but may also be invoked in the context of [Pre-Dial Handlers](#), [Pre-Bridge Handlers](#) and [Hangup Handlers](#) via the use of flags and arguments within other applications.

# GoSub

## Overview

GoSub is a dialplan application. It replaces (is recommended in place of, and deprecates) the Macro application.

GoSub allows you to execute a specific block (context or section) of dialplan as well as pass and return information via arguments to/from the scope of the block. Whereas Macro has issues with nesting, GoSub does not and GoSub should be used wherever you would have used a Macro.

Other dialplan applications, such as Dial and Queue make use of GoSub functionality from within their applications. That means, they allow you to perform actions like calling GoSub on the called party's channel from a Dial, or on a Queue member's channel after they answer. See the [Pre-Dial Handlers](#) and [Pre-Queue Handlers](#) sections for more information.

## Defining a dialplan context for use with GoSub

No special syntax is needed when defining the dialplan code that you want to call with GoSub, **unless** you want to **Return** back to where you called GoSub from. In the case of wanting to return, then you should call the [Return](#) application.

Here is an example of dialplan we could call with GoSub when we don't wish to return.

```
[my-gosub]
exten = s,1,Verbose("Here we are in a subroutine! Let's listen to some weasels")
same = s,n,Playback(tt-weasels)
```

Here is an example where we do wish to return.

```
[my-gosub]
exten = s,1,Verbose("Here we are in a subroutine! Let's listen to some weasels")
same = s,n,Playback(tt-weasels)
same = s,n,Return()
```

## Calling GoSub

GoSub syntax is simple, you only need to specify the priority, and then optionally the context and extension plus any arguments you wish to use.

```
Gosub([[context],[exten],[priority](arg1[,...][,argN]])]
```

Here is an example within Asterisk dialplan.

```
[somecontext]
exten = 7000,1,Verbose("We are going to run a Gosub before Dialing!")
same = n,Gosub(my-gosub,s,1)
same = n,Dial(PJSIP/ALICE)
```

Here we are calling the 'my-gosub' context at extension 's', priority '1'.

## Calling GoSub with arguments

If you want to pass information into your Gosub routine then you need to use arguments.

Here is how we call Gosub with an argument. We are substituting the EXTEN channel variable for the first argument field (ARG1).


```
[somecontext]
exten = 7000,1,Verbose("We are going to run a Gosub before Dialing!")
same = n,Gosub(my-gosub,s,1(${EXTEN}))
same = n,Dial(PJSIP/ALICE)
```

Below we make use of ARG1 in the Verbose message we print during the subroutine execution.

```
[my-gosub]
exten = s,1,Verbose("Here we are in a subroutine! This subroutine was called from
extension ${ARG1}")
same = s,n,Playback(tt-weasels)
same = s,n,Return()
```

To use multiple arguments, simply separate them via commas when defining them in the Gosub call. Then within the Gosub reference them as ARG1, ARG2, ARG3, etc.

# Hangup Handlers

 Hangup Handlers were added in **Asterisk 11**

## Overview

Hangup handlers are subroutines attached to a channel that will execute when that channel hangs up. Unlike the traditional `h` extension, hangup handlers follow the channel. Thus hangup handlers are always run when a channel is hung up, regardless of where in the dialplan a channel is executing.

Multiple hangup handlers can be attached to a single channel. If multiple hangup handlers are attached to a channel, the hangup handlers will be executed in the order of most recently added first.

### NOTES

- Please note that when the hangup handlers execute in relation to the `h` extension is not defined. They could execute before or after the `h` extension.
- Call transfers, call pickup, and call parking can result in channels on both sides of a bridge containing hangup handlers.
- Hangup handlers can be attached to any call leg using [pre-dial handlers](#).

### WARNINGS

- As hangup handlers are subroutines, they must be terminated with a call to `Return`.
- Adding a hangup handler in the `h` extension or during a hangup handler execution is undefined behaviour.
- As always, hangup handlers, like the `h` extension, need to execute quickly because they are in the hangup sequence path of the call leg. Specific channel driver protocols like ISDN and SIP may not be able to handle excessive delays completing the hangup sequence.

## Dialplan Applications and Functions

All manipulation of a channel's hangup handlers are done using the `CHANNEL` function. All values manipulated for hangup handlers are write-only.

### *hangup\_handler\_push*

Used to push a hangup handler onto a channel.

```
same => n,Set(CHANNEL(hangup_handler_push)=[context,]exten,]priority[(arg1[,...[,argN]])]);
```

### *hangup\_handler\_pop*

Used to pop a hangup handler off a channel. Optionally, a replacement hangup handler can be added to the channel.

```
same => n,Set(CHANNEL(hangup_handler_pop)=[context,]exten,]priority[(arg1[,...[,argN]])]);
```

### *hangup\_handler\_wipe*

Remove all hangup handlers on the channel. Optionally, a new hangup handler can be pushed onto the channel.

```
same => n,Set(CHANNEL(hangup_handler_wipe)=[context,]exten,]priority[(arg1[,...[,argN]])]);
```

## Examples

### *Adding hangup handlers to a channel*

In this example, three hangup handlers are added to a channel: `hdlr3`, `hdlr2`, and `hdlr1`. When the channel is hung up, they will be executed in the order of most recently added first - so `hdlr1` will execute first, followed by `hdlr2`, then `hdlr3`.

```

; Some dialplan extension
same => n,Set(CHANNEL(hangup_handler_push)=hdlr3,s,1(args));
same => n,Set(CHANNEL(hangup_handler_push)=hdlr2,s,1(args));
same => n,Set(CHANNEL(hangup_handler_push)=hdlr1,s,1(args));
; Continuing in some dialplan extension

[hdlr1]

exten => s,1,Verbose(0, Executed First)
same => n,Return()

[hdlr2]

exten => s,1,Verbose(0, Executed Second)
same => n,Return()

[hdlr3]

exten => s,1,Verbose(0, Executed Third)
same => n,Return()

```

### ***Removing and replacing hangup handlers***

In this example, three hangup handlers are added to a channel: hdlr3, hdlr2, and hdlr1. Using the `CHANNEL` function's `hangup_handler_pop` value, hdlr1 is removed from the stack of hangup handlers. Then, using the `hangup_handler_pop` value again, hdlr2 is replaced with hdlr4. When the channel is hung up, hdlr4 will be executed, followed by hdlr3.

```

; Some dialplan extension
same => n,Set(CHANNEL(hangup_handler_push)=hdlr3,s,1(args));
same => n,Set(CHANNEL(hangup_handler_push)=hdlr2,s,1(args));
same => n,Set(CHANNEL(hangup_handler_push)=hdlr1,s,1(args));
; Remove hdlr1
same => n,Set(CHANNEL(hangup_handler_pop)=)
; Replace hdlr2 with hdlr4
same => n,Set(CHANNEL(hangup_handler_pop)=hdlr4,s,1(args));

; Continuing in some dialplan extension

[hdlr1]

exten => s,1,Verbose(0, Not Executed)
same => n,Return()

[hdlr2]

exten => s,1,Verbose(0, Not Executed)
same => n,Return()

[hdlr3]

exten => s,1,Verbose(0, Executed Second)
same => n,Return()

[hdlr4]

exten => s,1,Verbose(0, Executed First)
same => n,Return()

```

## CLI Commands

### Single channel

```
core show hanguphandlers <chan>
```

### Output

```

Channel      Handler
<chan-name>  <first handler to execute>
              <second handler to execute>
              <third handler to execute>

```

### All channels

```
core show hanguphandlers all
```

### Output

```

Channel      Handler
<chan1-name> <first handler to execute>
              <second handler to execute>
              <third handler to execute>
<chan2-name> <first handler to execute>
<chan3-name> <first handler to execute>
              <second handler to execute>

```

# Macros

## Overview



Macros are very similar in function to the [Gosub](#) application **which deprecates Macro**. This information is here for historical purposes and you should really use Gosub wherever you would have previously used Macro.

Macro is a dialplan application that facilitates code-reuse within the dialplan. That is, a macro, once defined can be called from almost anywhere else within the dialplan using the Macro application or else via flags and arguments for other applications that allow calling macros.

Other dialplan applications, such as [Dial](#) and [Queue](#) make use of Macro functionality from within their applications. That means, they allow you to perform actions like calling Macro (or Gosub) on the called party's channel from a Dial, or on a Queue member's channel after they answer. See the [Pre-Dial Handlers](#) and [Pre-Bridge Handlers](#) sections for more information.

## Variables and arguments available within a Macro

The calling extension, context, and priority are stored in **MACRO\_EXTEN**, **MACRO\_CONTEXT** and **MACRO\_PRIORITY** respectively. Arguments become **ARG1**, **ARG2**, etc in the macro context. If you Goto out of the Macro context, the Macro will terminate and control will be returned at the location of the Goto. If **MACRO\_OFFSET** is set at termination, Macro will attempt to continue at priority **MACRO\_OFFSET + N + 1** if such a step exists, and **N + 1** otherwise.

## Defining a dialplan context for use with Macro

Macros look like a typical dialplan context, except for two factors:

- Macros must be named with the 'macro-' prefix.
- Macros must use the 's' extension.

```
[macro-announcement]
exten = s,1,NoOp()
same = n,Playback(tt-weasels)
```

## Calling a Macro

Macro syntax is simple, you only need to specify the priority, and then optionally the context and extension plus any arguments you wish to use.

```
Macro([[context],[exten],[priority]([arg1],[...],[argN]])]
```

Here is an example within Asterisk dialplan.

```
[somecontext]
exten = 7000,1,Verbose("We are going to run a Macro before Dialing!")
same = n,Macro(announcement,s,1)
same = n,Dial(PJSIP/ALICE)
```

As you can see we are calling the 'announcement' macro at context 'macro-announcement', extension 's', priority '1'.

## Calling Macro with arguments

Other than the predefined variables mentioned earlier on this page, if you want to pass information into your Macro routine then you need to use arguments.

Here is how we call Macro with an argument. We are substituting the EXTEN channel variable for the first argument field (ARG1).

```
[somecontext]
exten = 7000,1,Verbose("We are going to run a Macro before Dialing!")
same = n,Gosub(announcement,s,1(${EXTEN}))
same = n,Dial(PJSIP/ALICE)
```

Below notice that make use of ARG1 in the Verbose message we print during the subroutine execution.



```
[macro-announcement]
exten = s,l,Verbose("Here we are in a subroutine! This subroutine was called from
extension ${ARG1}")
same = s,n,Playback(tt-weasels)
same = s,n,Return()
```

To use multiple arguments, simply separate them via commas when defining them in the Macro call. Then within the Macro reference them as ARG1, ARG2, ARG3, etc.

# Party ID Interception Macros and Routines

## Interception routines



As Interception routines are implemented internally using the `Gosub` application, all routines should end with an explicit call to the `Return` application.

The interception routines give the administrator an opportunity to alter [connected line and redirecting information](#) before the channel driver is given the information. If the routine does not change a value then that is what is going to be passed to the channel driver.

The tag string available in `CALLERID`, `CONNECTEDLINE`, and `REDIRECTING` is useful for the interception routines to provide some information about where the information originally came from.

The 'i' option of the `CONNECTEDLINE` dialplan function should always be used in the `CONNECTED_LINE` interception routines. The interception routine always passes the connected line information on to the channel driver when the routine returns. Similarly, the 'i' option of the `REDIRECTING` dialplan function should always be used in the `REDIRECTING` interception routines.



Note that Interception routines do not attempt to draw a distinction between caller/callee. As it turned out, it was not a good thing to distinguish since transfers make a mockery of caller/callee.

- `$(REDIRECTING_SEND_SUB)`  
Subroutine to call before sending a redirecting update to the party.
- `$(REDIRECTING_SEND_SUB_ARGS)`  
Arguments to pass to `$(REDIRECTING_CALLEE_SEND_SUB)`.
- `$(CONNECTED_LINE_SEND_SUB)`  
Subroutine to call before sending a connected line update to the party.
- `$(CONNECTED_LINE_SEND_SUB_ARGS)`  
Arguments to pass to `$(CONNECTED_LINE_SEND_SUB)`.

## Interception macros



### WARNING

Interception macros have been deprecated in Asterisk 11 due to deprecation of [Macro](#). Users of the interception functionality should plan to migrate to [Interception routines](#).

The interception macros give the administrator an opportunity to alter [connected line and redirecting information](#) before the channel driver is given the information. If the macro does not change a value then that is what is going to be passed to the channel driver.

The tag string available in `CALLERID`, `CONNECTEDLINE`, and `REDIRECTING` is useful for the interception macros to provide some information about where the information originally came from.

The 'i' option of the `CONNECTEDLINE` dialplan function should always be used in the `CONNECTED_LINE` interception macros. The interception macro always passes the connected line information on to the channel driver when the macro exits. Similarly, the 'i' option of the `REDIRECTING` dialplan function should always be used in the `REDIRECTING` interception macros.

- `$(REDIRECTING_CALLEE_SEND_MACRO)`  
Macro to call before sending a redirecting update to the callee. This macro may never be needed since the redirecting updates should only go from the callee to the caller direction. It is available for completeness.
- `$(REDIRECTING_CALLEE_SEND_MACRO_ARGS)`  
Arguments to pass to `$(REDIRECTING_CALLEE_SEND_MACRO)`.
- `$(REDIRECTING_CALLER_SEND_MACRO)`  
Macro to call before sending a redirecting update to the caller.
- `$(REDIRECTING_CALLER_SEND_MACRO_ARGS)`  
Arguments to pass to `$(REDIRECTING_CALLER_SEND_MACRO)`.
- `$(CONNECTED_LINE_CALLEE_SEND_MACRO)`  
Macro to call before sending a connected line update to the callee.
- `$(CONNECTED_LINE_CALLEE_SEND_MACRO_ARGS)`  
Arguments to pass to `$(CONNECTED_LINE_CALLEE_SEND_MACRO)`.
- `$(CONNECTED_LINE_CALLER_SEND_MACRO)`  
Macro to call before sending a connected line update to the caller.

- `$(CONNECTED_LINE_CALLER_SEND_MACRO_ARGS)`  
Arguments to pass to `$(CONNECTED_LINE_CALLER_SEND_MACRO)`.

# Pre-Bridge Handlers

## Overview

Pre-bridge handlers allow you to execute dialplan subroutines on a channel after the call has been initiated and the channels have been created, but before connecting the caller to the callee. Handlers for the `Dial` and `queue` applications allow both the older `macro` and the newer `gosub` routines to be executed. These handlers are executed on the **called party** channel, after it is **answered**, but **pre-bridge** before the calling and called party are connected.

If you want to execute routines earlier in the call lifetime then check out the [Pre-Dial Handlers](#) section.

Pre-bridge handlers are invoked using flags or arguments for a particular dialplan application. The dialplan application help documentation within Asterisk goes into detail on the various arguments, options and flags, however we will provide some examples below. You should always check the CLI or wiki application docs for any updates.

## Dial application

There are two flags for the Dial application, **M** and **U**.

### M flag

The M flag allows a `macro` and arguments to be specified. You must specify the macro name, leaving off the 'macro-' prefix.

```
M(macro[ ^arg[ ^... ] ])
```

The variable `MACRO_RESULT` can be set with certain options inside the specified macro to determine behavior when the macro finishes. The options are documented in the [Dial application documentation](#).

- [Overview](#)
  - [Dial application](#)
  - [Queue application](#)
  - [Examples](#)

### U flag

The U flag allows a `gosub` and arguments to be specified. You must remember to call `Return` inside the `gosub`.

```
U(x[ ^arg[ ^... ] ])
```

The variable `GOSUB_RESULT` can be set within certain options inside the specified `gosub` to determine behavior when the `gosub` returns. The options are documented in the [Dial application documentation](#).

## Queue application

The Queue application, similar to Dial, has two options for handling pre-bridge subroutines. For Queue, both arguments have the same syntax.

```
Queue(queueName[ ,options[ ,URL[ ,announceoverride[ ,timeout[ ,AGI[ ,macro[ ,gosub[ ,rule[ ,position]]]]]]]]]
```

`macro` and `gosub` can both be populated with the name of a macro or `gosub` routine to execute on the called party channel as described in the overview.

## Examples

### Example 1 - Executing a pre-bridge macro handler from Dial

BOB(6002) dials ALICE(6001) and Playback is executed from within the subroutine on the called party's channel after they answer.

Dialplan

```
[from-internal]
exten = 6001,1,Dial(PJSIP/ALICE,30,M(announcement))

[macro-announcement]
exten = s,1,NoOp()
same = n,Playback(tt-weasels)
same = n,Hangup()
```

CLI output

```

-- Executing [6001@from-internal:1] Dial("PJSIP/BOB-00000014", "PJSIP/ALICE,30,M(announcement)") in new stack
-- Called PJSIP/ALICE
-- PJSIP/ALICE-00000015 is ringing
-- PJSIP/ALICE-00000015 answered PJSIP/BOB-00000014
-- Executing [s@macro-announcement:1] NoOp("PJSIP/ALICE-00000015", "") in new stack
-- Executing [s@macro-announcement:2] Playback("PJSIP/ALICE-00000015", "tt-weasels") in new stack
-- <PJSIP/ALICE-00000015> Playing 'tt-weasels.gsm' (language 'en')
-- Channel PJSIP/BOB-00000014 joined 'simple_bridge' basic-bridge <612c2313-98bf-48ce-89b1-d530b06e44d7>
-- Channel PJSIP/ALICE-00000015 joined 'simple_bridge' basic-bridge <612c2313-98bf-48ce-89b1-d530b06e44d7>
-- Channel PJSIP/BOB-00000014 left 'native_rtp' basic-bridge <612c2313-98bf-48ce-89b1-d530b06e44d7>
-- Channel PJSIP/ALICE-00000015 left 'native_rtp' basic-bridge <612c2313-98bf-48ce-89b1-d530b06e44d7>
== Spawn extension (from-internal, 6001, 1) exited non-zero on 'PJSIP/BOB-00000014'

```

### Example 2 - Executing a pre-bridge gosub handler from Dial

ALICE(6001) dials BOB(6002) and Playback is executed from within the subroutine on the called party's channel after they answer. Notice that since this subroutine is a gosub, we need to call Return.

Dialplan

```

[from-internal]
exten = 6002,1,Dial(PJSIP/BOB,30,U(sub-announcement))

[sub-announcement]
exten = s,1,NoOp()
same = n,Playback(tt-weasels)
same = n,Return()

```

CLI output

```

-- Executing [6002@from-internal:1] Dial("PJSIP/ALICE-00000016", "PJSIP/BOB,30,U(sub-announcement)") in new stack
-- Called PJSIP/BOB
-- PJSIP/BOB-00000017 is ringing
-- PJSIP/BOB-00000017 answered PJSIP/ALICE-00000016
-- PJSIP/BOB-00000017 Internal Gosub(sub-announcement,s,1) start
-- Executing [s@sub-announcement:1] NoOp("PJSIP/BOB-00000017", "") in new stack
-- Executing [s@sub-announcement:2] Playback("PJSIP/BOB-00000017", "tt-weasels") in new stack
-- <PJSIP/BOB-00000017> Playing 'tt-weasels.gsm' (language 'en')
-- Executing [s@sub-announcement:3] Return("PJSIP/BOB-00000017", "") in new stack
== Spawn extension (from-internal, , 1) exited non-zero on 'PJSIP/BOB-00000017'
-- PJSIP/BOB-00000017 Internal Gosub(sub-announcement,s,1) complete GOSUB_RETVAL=
-- Channel PJSIP/ALICE-00000016 joined 'simple_bridge' basic-bridge <16e76a40-4a24-441d-a2b2-5c9ddfb21d7a>
-- Channel PJSIP/BOB-00000017 joined 'simple_bridge' basic-bridge <16e76a40-4a24-441d-a2b2-5c9ddfb21d7a>
-- Channel PJSIP/BOB-00000017 left 'native_rtp' basic-bridge <16e76a40-4a24-441d-a2b2-5c9ddfb21d7a>
-- Channel PJSIP/ALICE-00000016 left 'native_rtp' basic-bridge <16e76a40-4a24-441d-a2b2-5c9ddfb21d7a>
== Spawn extension (from-internal, 6002, 1) exited non-zero on 'PJSIP/ALICE-00000016'

```

### Example 3 - Executing a pre-bridge gosub handler from Queue

ALICE(6001) dials Queue 'sales' where BOB(6002) is a member. Once BOB answers the queue call, the Playback is executed from within the gosub.

Dialplan

```

[sub-announcement]
exten = s,1,NoOp()
same = n,Playback(tt-weasels)
same = n,Return()

[queues]
exten => 7002,1,Verbose(2,${CALLERID(all)} entering the sales queue)
same => n,Queue(sales,,,,,sub-announcement)
same => n,Hangup()

```

CLI output

```

-- Executing [7002@from-internal:1] Verbose("PJSIP/ALICE-00000009", "2,"Alice" <ALICE> entering the sales queue") in new stack
== "Alice" <ALICE> entering the sales queue
-- Executing [7002@from-internal:2] Queue("PJSIP/ALICE-00000009", "sales,,,,,,,,sub-announcement") in new stack
-- Started music on hold, class 'default', on channel 'PJSIP/ALICE-00000009'
-- Called PJSIP/BOB
-- PJSIP/BOB-0000000a is ringing
> 0x7f74d4039840 -- Probation passed - setting RTP source address to 10.24.18.138:4042
-- PJSIP/BOB-0000000a answered PJSIP/ALICE-00000009
-- Stopped music on hold on PJSIP/ALICE-00000009
-- PJSIP/BOB-0000000a Internal Gosub(sub-announcement,s,1) start
-- Executing [s@sub-announcement:1] NoOp("PJSIP/BOB-0000000a", "") in new stack
-- Executing [s@sub-announcement:2] Playback("PJSIP/BOB-0000000a", "tt-weasels") in new stack
-- <PJSIP/BOB-0000000a> Playing 'tt-weasels.gsm' (language 'en')
-- Executing [s@sub-announcement:3] Return("PJSIP/BOB-0000000a", "") in new stack
== Spawn extension (from-internal, 7002, 1) exited non-zero on 'PJSIP/BOB-0000000a'
-- PJSIP/BOB-0000000a Internal Gosub(sub-announcement,s,1) complete GOSUB_RETVAL=
-- Channel PJSIP/ALICE-00000009 joined 'simple_bridge' basic-bridge <cbc54ed6-1f51-4b10-be99-4994f52d851f>
-- Channel PJSIP/BOB-0000000a joined 'simple_bridge' basic-bridge <cbc54ed6-1f51-4b10-be99-4994f52d851f>
> Bridge cbc54ed6-1f51-4b10-be99-4994f52d851f: switching from simple_bridge technology to native_rtp
> Remotely bridged 'PJSIP/BOB-0000000a' and 'PJSIP/ALICE-00000009' - media will flow directly between them
> Remotely bridged 'PJSIP/BOB-0000000a' and 'PJSIP/ALICE-00000009' - media will flow directly between them
> 0x7f74d400c620 -- Probation passed - setting RTP source address to 10.24.18.16:4040
-- Channel PJSIP/BOB-0000000a left 'native_rtp' basic-bridge <cbc54ed6-1f51-4b10-be99-4994f52d851f>
-- Channel PJSIP/ALICE-00000009 left 'native_rtp' basic-bridge <cbc54ed6-1f51-4b10-be99-4994f52d851f>
== Spawn extension (from-internal, 7002, 2) exited non-zero on 'PJSIP/ALICE-00000009'

```

#### Example 4 - Executing a pre-bridge macro handler from Queue

BOB(6002) calls the queue 'support' where ALICE(6001) is a member. Once ALICE answers the queue call, the Playback is executed from within the macro.

Dialplan

```

[macro-announcement]
exten = s,1,NoOp()
same = n,Playback(tt-weasels)

[queues]
exten => 7001,1,Verbose(2,${CALLERID(all)} entering the support queue)
same => n,Queue(support,,,,,,,,announcement)
same => n,Hangup()

```


CLI output

```

-- Executing [7001@from-internal:1] Verbose("PJSIP/BOB-00000004", "2,"Bob" <BOB> entering the support queue") in new stack
== "Bob" <BOB> entering the support queue
-- Executing [7001@from-internal:2] Queue("PJSIP/BOB-00000004", "support,,,,,,,,announcement") in new stack
-- Started music on hold, class 'default', on channel 'PJSIP/BOB-00000004'
-- Called PJSIP/ALICE
-- PJSIP/ALICE-00000005 is ringing
> 0x7f8450039d40 -- Probation passed - setting RTP source address to 10.24.18.16:4048
-- PJSIP/ALICE-00000005 answered PJSIP/BOB-00000004
-- Stopped music on hold on PJSIP/BOB-00000004
-- Executing [s@macro-announcement:1] NoOp("PJSIP/ALICE-00000005", "") in new stack
-- Executing [s@macro-announcement:2] Playback("PJSIP/ALICE-00000005", "tt-weasels") in new stack
-- <PJSIP/ALICE-00000005> Playing 'tt-weasels.gsm' (language 'en')
-- Channel PJSIP/BOB-00000004 joined 'simple_bridge' basic-bridge <8283212f-b12d-4571-9653-0c8484e88980>
-- Channel PJSIP/ALICE-00000005 joined 'simple_bridge' basic-bridge <8283212f-b12d-4571-9653-0c8484e88980>
> Bridge 8283212f-b12d-4571-9653-0c8484e88980: switching from simple_bridge technology to native_rtp
> Remotely bridged 'PJSIP/ALICE-00000005' and 'PJSIP/BOB-00000004' - media will flow directly between them
> Remotely bridged 'PJSIP/ALICE-00000005' and 'PJSIP/BOB-00000004' - media will flow directly between them
> 0x7f84500145d0 -- Probation passed - setting RTP source address to 10.24.18.138:4050
-- Channel PJSIP/ALICE-00000005 left 'native_rtp' basic-bridge <8283212f-b12d-4571-9653-0c8484e88980>
-- Channel PJSIP/BOB-00000004 left 'native_rtp' basic-bridge <8283212f-b12d-4571-9653-0c8484e88980>
== Spawn extension (from-internal, 7001, 2) exited non-zero on 'PJSIP/BOB-00000004'

```

## Pre-Dial Handlers

 Pre-Dial Handlers were added in **Asterisk 11**

### Overview

Pre-dial handlers allow you to execute a dialplan subroutine on a channel before a call is placed but after the application performing a dial action is invoked. This means that the handlers are executed after the creation of the caller/callee channels, but before any actions have been taken to actually dial the callee channels. You can execute a dialplan subroutine on the caller channel and on each callee channel dialed.

There are two ways in which a pre-dial handler can be invoked:

- The '**B**' option in an application executes a dialplan subroutine on the caller channel before any callee channels are created.
- The '**b**' option in an application executes a dialplan subroutine on each callee channel after it is created but before the call is placed to the end-device.

Pre-dial handlers are supported in the [Dial](#) application and the [FollowMe](#) application.


### WARNINGS

- As pre-dial handlers are implemented using [Gosub](#) subroutines, they must be terminated with a call to [Return](#).
- Taking actions in pre-dial handlers that would put the caller/callee channels into other applications will result in undefined behaviour. Pre-dial handlers should be short routines that do not impact the state that the dialing application assumes the channel will be in.

### Syntax

For [Dial](#) or [FollowMe](#), handlers are invoked using similar nomenclature as other options (such as **M** or **U** in [Dial](#)) that cause some portion of the dialplan to execute.

```
b([[context^]exten^]priority[(arg1[^[^...][^argN]])])
B([[context^]exten^]priority[(arg1[^[^...][^argN]])])
```

 If context or exten are not supplied then the current values from the caller channel are used.

### Examples

The examples illustrated below use the following channels:

- *SIP/foo* is calling either *SIP/bar*, *SIP/baz*, or both
- *SIP/foo* is the caller
- *SIP/bar* is a callee
- *SIP/baz* is another callee

#### Example 1 - Executing a pre-dial handler on the caller channel

```
[default]

exten => s,1,NoOp()
same => n,Dial(SIP/bar,,B(default^caller_handler^1))
same => n,Hangup()

exten => caller_handler,1,NoOp()
same => n,Verbose(0, In caller pre-dial handler!)
same => n,Return()
```

#### Example 1 CLI Output

```
<SIP/foo-123> Dial(SIP/bar,,B(default^caller_handler^1))
<SIP/foo-123> Executing default,caller_handler,1
<SIP/foo-123> In caller pre-dial handler!
<SIP/foo-123> calling SIP/bar-124
```

### Example 2 - Executing a pre-dial handler on a callee channel

```
[default]

exten => s,1,NoOp()
same => n,Dial(SIP/bar,,b(default^callee_handler^1))
same => n,Hangup()

exten => callee_handler,1,NoOp()
same => n,Verbose(0, In callee pre-dial handler!)
same => n,Return()
```

#### Example 2 CLI Output

```
<SIP/foo-123> Dial(SIP/bar,,b(default^callee_handler^1))
<SIP/bar-124> Executing default,callee_handler,1
<SIP/bar-124> In callee pre-dial handler!
<SIP/foo-123> calling SIP/bar-124
```

### Example 3 - Executing a pre-dial handler on multiple callee channels

```
[default]

exten => s,1,NoOp()
same => n,Dial(SIP/bar&SIP/baz,,b(default^callee_handler^1))
same => n,Hangup()

exten => callee_handler,1,NoOp()
same => n,Verbose(0, In callee pre-dial handler!)
same => n,Return()
```

#### Example 3 CLI Output

```
<SIP/foo-123> Dial(SIP/bar&SIP/baz,,b(default^callee_handler^1))
<SIP/bar-124> Executing default,callee_handler,1
<SIP/bar-124> In callee pre-dial handler!
<SIP/baz-125> Executing default,callee_handler,1
<SIP/baz-125> In callee pre-dial handler!
<SIP/foo-123> calling SIP/bar-124
<SIP/foo-123> calling SIP/baz-125
```



## Expressions

Everything contained inside a bracket pair prefixed by a \$ (like `$(this)`) is considered as an expression and it is evaluated. Evaluation works similar to (but is done on a later stage than) variable substitution: the expression (including the square brackets) is replaced by the result of the expression evaluation.

For example, after the sequence:

```
exten => 1,1,Set(lala=${1 + 2}) exten => 1,2,Set(koko=${2 * ${lala}})
```

the value of variable koko is "6".

and, further:

```
exten => 1,1,Set,(lala=${ 1 + 2 });
```

will parse as intended. Extra spaces are ignored.

## Spaces Inside Variables Values

If the variable being evaluated contains spaces, there can be problems.

For these cases, double quotes around text that may contain spaces will force the surrounded text to be evaluated as a single token. The double quotes will be counted as part of that lexical token.

As an example:

```
exten => s,6,GotoIf($[ "${CALLERID(name)}" : "Privacy Manager" ]?callerid-liar,s,1:s,7)
```

The variable CALLERID(name) could evaluate to "DELOREAN MOTORS" (with a space) but the above will evaluate to:

- "DELOREAN MOTORS" : "Privacy Manager"

and will evaluate to 0.

The above without double quotes would have evaluated to:

- DELOREAN MOTORS : Privacy Manager

and will result in syntax errors, because token DELOREAN is immediately followed by token MOTORS and the expression parser will not know how to evaluate this expression, because it does not match its grammar.

## Operators

Operators are listed below in order of increasing precedence. Operators with equal precedence are grouped within { } symbols.

- `expr1 | expr2`  
Return the evaluation of `expr1` if it is neither an empty string nor zero; otherwise, returns the evaluation of `expr2`.
- `expr1 & expr2`  
Return the evaluation of `expr1` if neither expression evaluates to an empty string or zero; otherwise, returns zero.
- `expr1 {=, >, >=, <, <=, !=} expr2`  
Return the results of floating point comparison if both arguments are numbers; otherwise, returns the results of string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relation is true, or 0 if the relation is false.
- `expr1 {+, -} expr2`  
Return the results of addition or subtraction of floating point-valued arguments.
- `expr1 {/, %} expr2*`  
Return the results of multiplication, floating point division, or remainder of arguments.
- `- expr1`  
Return the result of subtracting `expr1` from 0. This, the unary minus operator, is right associative, and has the same precedence as the `!` operator.
- `! expr1`  
Return the result of a logical complement of `expr1`. In other words, if `expr1` is null, 0, an empty string, or the string "0", return a 1. Otherwise, return a 0. It has the same precedence as the unary minus operator, and is also right associative.
- `expr1 : expr2`  
The ``:` operator matches `expr1` against `expr2`, which must be a regular expression. The regular expression is anchored to the beginning of the string with an implicit ```.

If the match succeeds and the pattern contains at least one regular expression subexpression ```, the string corresponding to ``1` is returned; otherwise the matching operator returns the number of characters matched. If the match fails and the pattern contains a regular expression subexpression the null string is returned; otherwise 0.

Normally, the double quotes wrapping a string are left as part of the string. This is disastrous to the `:` operator. Therefore, before the regex match is made, beginning and ending double quote characters are stripped from both the pattern and the string.

- `expr1 =~ expr2`  
Exactly the same as the `:` operator, except that the match is not anchored to the beginning of the string. Pardon any similarity to seemingly similar operators in other programming languages! The `:"` and `"=` operators share the same precedence.
- `expr1 ? expr2 :: expr3`  
Traditional Conditional operator. If `expr1` is a number that evaluates to 0 (false), `expr3` is result of the this expression evaluation. Otherwise, `expr2` is the result. If `expr1` is a string, and evaluates to an empty string, or the two characters (""), then `expr3` is the result. Otherwise, `expr2` is the result. In Asterisk, all 3 exprs will be "evaluated"; if `expr1` is "true", `expr2` will be the result of the "evaluation" of this expression. `expr3` will be the result otherwise. This operator has the lowest precedence.
- `expr1 ~~ expr2`  
Concatenation operator. The two exprs are evaluated and turned into strings, stripped of surrounding double quotes, and are turned into a single string with no intervening spaces. This operator is new to trunk after 1.6.0; it is not needed in existing `extensions.conf` code. Because of the way asterisk evaluates `[]` constructs (recursively, bottom- up), no is ever present when the contents of a `[]` is evaluated. Thus, tokens are usually already merged at evaluation time. But, in AEL, various exprs are evaluated raw, and `[]` are gathered and treated as tokens. And in AEL, no two tokens can sit side by side without an intervening operator. So, in AEL, concatenation must be explicitly specified in expressions. This new operator will play well into future plans, where expressions ( constructs) are merged into a single grammar.

Parentheses are used for grouping in the usual manner.

Operator precedence is applied as one would expect in any of the C or C derived languages.

## Floating Point Numbers

In 1.6 and above, we shifted the `$[...]` expressions to be calculated via floating point numbers instead of integers. We use 'long double' numbers when possible, which provide around 16 digits of precision with 12 byte numbers.

To specify a floating point constant, the number has to have this format: `D.D`, where `D` is a string of base 10 digits. So, you can say `0.10`, but you can't say `.10` or `20.` - we hope this is not an excessive restriction!

Floating point numbers are turned into strings via the `'%g'/'%Lg'` format of the `printf` function set. This allows numbers to still 'look' like integers to those counting on integer behavior. If you were counting on `1/4` evaluating to `0`, you need to now say `TRUNC(1/4)`. For a list of all the truncation/rounding capabilities, see the next section.

## Expr2 Built-in Functions

In 1.6 and above, we upgraded the `[$]` expressions to handle floating point numbers. Because of this, folks counting on integer behavior would be disrupted. To make the same results possible, some rounding and integer truncation functions have been added to the core of the Expr2 parser. Indeed, dialplan functions can be called from `[$...]` expressions without the `#{...}` operators. The only trouble might be in the fact that the arguments to these functions must be specified with a comma. If you try to call the MATH function, for example, and try to say `3 + MATH(7*8)`, the expression parser will evaluate `7*8` for you into 56, and the MATH function will most likely complain that its input doesn't make any sense.

We also provide access to most of the floating point functions in the C library. (but not all of them).

While we don't expect someone to want to do Fourier analysis in the dialplan, we don't want to preclude it, either.

Here is a list of the 'builtin' functions in Expr2. All other dialplan functions are available by simply calling them (read-only). In other words, you don't need to surround function calls in `[$...]` expressions with `#{...}`. Don't jump to conclusions, though! - you still need to wrap variable names in curly braces!

- `COS(x)` x is in radians. Results vary from -1 to 1.
- `SIN(x)` x is in radians. Results vary from -1 to 1.
- `TAN(x)` x is in radians.
- `ACOS(x)` x should be a value between -1 and 1.
- `ASIN(x)` x should be a value between -1 and 1.
- `ATAN(x)` returns the arc tangent in radians; between  $-\pi/2$  and  $\pi/2$ .
- `ATAN2(x,y)` returns a result resembling `y/x`, except that the signs of both args are used to determine the quadrant of the result. Its result is in radians, between  $-\pi$  and  $\pi$ .
- `POW(x,y)` returns the value of x raised to the power of y.
- `SQRT(x)` returns the square root of x.
- `FLOOR(x)` rounds x down to the nearest integer.
- `CEIL(x)` rounds x up to the nearest integer.
- `ROUND(x)` rounds x to the nearest integer, but round halfway cases away from zero.
- `RINT(x)` rounds x to the nearest integer, rounding halfway cases to the nearest even integer.
- `TRUNC(x)` rounds x to the nearest integer not larger in absolute value.
- `REMAINDER(x,y)` computes the remainder of dividing x by y. The return value is  $x - n*y$ , where n is the value `x/y`, rounded to the nearest integer. If this quotient is `1/2`, it is rounded to the nearest even number.
- `EXP(x)` returns e to the x power.
- `EXP2(x)` returns 2 to the x power.
- `LOG(x)` returns the natural logarithm of x.
- `LOG2(x)` returns the base 2 log of x.
- `LOG10(x)` returns the base 10 log of x.

## Expressions Examples

\*"One Thousand Five Hundred" =~ "T[Expressions Examples^ ]"

returns: Thousand

- "One Thousand Five Hundred" =~ "T[Expressions Examples^ ]"  
returns: 8
- "One Thousand Five Hundred" : "T[Expressions Examples^ ]"  
returns: 0
- "8015551212" : "(...)"  
returns: 801
- "3075551212" : "...(...)"  
returns: 555
- ! "One Thousand Five Hundred" =~ "T[Expressions Examples^ ]"  
returns: 0  
(because it applies to the string, which is non-null, which it turns to "0", and then looks for the pattern in the "0", and doesn't find it)
- !( "One Thousand Five Hundred" : "T[Expressions Examples^ ]+" )  
returns: 1  
(because the string doesn't start with a word starting with T, so the match evals to 0, and the ! operator inverts it to 1 )
- 2 + 8 / 2  
returns: 6  
(because of operator precedence; the division is done first, then the addition)
- 2+8/2  
returns: 6  
Spaces aren't necessary
- (2+8)/2  
returns: 5  
of course
- (3+8)/2  
returns: 5.5
- TRUNC((3+8)/2)  
returns: 5
- FLOOR(2.5)  
returns: 2
- FLOOR(-2.5)  
returns: -3
- CEIL(2.5)  
returns: 3
- CEIL(-2.5)  
returns: -2
- ROUND(2.5)  
returns: 3
- ROUND(3.5)  
returns: 4
- ROUND(-2.5)  
returns: -3
- RINT(2.5)  
returns: 2
- RINT(3.5)  
returns: 4
- RINT(-2.5)  
returns: -2
- RINT(-3.5)  
returns: -4
- TRUNC(2.5)

returns: 2

- TRUNC(3.5)  
returns: 3
- TRUNC(-3.5)  
returns: -3

Of course, all of the above examples use constants, but would work the same if any of the numeric or string constants were replaced with a variable reference `$(CALLERID(num))`, for instance.

## Numbers Vs. Strings

Tokens consisting only of numbers are converted to 'long double' if possible, which are from 80 bits to 128 bits depending on the OS, compiler, and hardware. This means that overflows can occur when the numbers get above 18 digits (depending on the number of bits involved). Warnings will appear in the logs in this case.



## Expression Parsing Errors

Syntax errors are now output with 3 lines.

If the extensions.conf file contains a line like:

```
exten => s,6,GotoIf($[ "${CALLERID(num)}" = "3071234567" & & "${CALLERID(name)}" :  
"Privacy Manager" ]?callerid-liar,s,1:s,7)
```

You may see an error in /var/log/asterisk/messages like this:

```
Jul 15 21:27:49 WARNING[1251240752]: ast_yyerror(): syntax error: parse error, unexpected TOK_AND, expecting TOK_M  
INUS or TOK_LP or TOKEN; Input:  
"3072312154" = "3071234567" & & "Steves Extension" : "Privacy Manager"  
      ^
```

The log line tells you that a syntax error was encountered. It now also tells you (in grand standard bison format) that it hit an "AND" (&) token unexpectedly, and that was hoping for a MINUS , LP (left parenthesis), or a plain token (a string or number).

The next line shows the evaluated expression, and the line after that, the position of the parser in the expression when it became confused, marked with the "^" character.

## NULL Strings

Testing to see if a string is null can be done in one of two different ways:

```
exten => _XX.,1,GotoIf("${calledid}" != "")?3)
```

or:

```
exten => _XX.,1,GotoIf([foo${calledid} != foo]?3)
```

The second example above is the way suggested by the WIKI. It will work as long as there are no spaces in the evaluated value.

The first way should work in all cases, and indeed, might now be the safest way to handle this situation.

## Warnings about Expressions

If you need to do complicated things with strings, asterisk expressions is most likely NOT the best way to go about it. AGI scripts are an excellent option to this need, and make available the full power of whatever language you desire, be it Perl, C, C++, Cobol, RPG, Java, Snobol, PL/I, Scheme, Common Lisp, Shell scripts, Tcl, Forth, Modula, Pascal, APL, assembler, etc.

## Expression Parser Incompatibilities

The asterisk expression parser has undergone some evolution. It is hoped that the changes will be viewed as positive.

The "original" expression parser had a simple, hand-written scanner, and a simple bison grammar. This was upgraded to a more involved bison grammar, and a hand-written scanner upgraded to allow extra spaces, and to generate better error diagnostics. This upgrade required bison 1.85, and part of the user community felt the pain of having to upgrade their bison version.

The next upgrade included new bison and flex input files, and the makefile was upgraded to detect current version of both flex and bison, conditionally compiling and linking the new files if the versions of flex and bison would allow it.

If you have not touched your extensions.conf files in a year or so, the above upgrades may cause you some heartburn in certain circumstances, as several changes have been made, and these will affect asterisk's behavior on legacy extension.conf constructs. The changes have been engineered to minimize these conflicts, but there are bound to be problems.

The following list gives some (and most likely, not all) of areas of possible concern with "legacy" extension.conf files:

1. Tokens separated by space(s). Previously, tokens were separated by spaces. Thus, ' 1 + 1 ' would evaluate to the value '2', but '1+1' would evaluate to the string '1+1'. If this behavior was depended on, then the expression evaluation will break. '1+1' will now evaluate to '2', and something is not going to work right. To keep such strings from being evaluated, simply wrap them in double quotes: ' "1+1" '
2. The colon operator. In versions previous to double quoting, the colon operator takes the right hand string, and using it as a regex pattern, looks for it in the left hand string. It is given an implicit operator at the beginning, meaning the pattern will match only at the beginning of the left hand string. If the pattern or the matching string had double quotes around them, these could get in the way of the pattern match. Now, the wrapping double quotes are stripped from both the pattern and the left hand string before applying the pattern. This was done because it recognized that the new way of scanning the expression doesn't use spaces to separate tokens, and the average regex expression is full of operators that the scanner will recognize as expression operators. Thus, unless the pattern is wrapped in double quotes, there will be trouble. For instance, `_${VAR1} : (WhoWhat)+` may have worked before, but unless you wrap the pattern in double quotes now, look out for trouble! This is better: `"_${VAR1}" : "(WhoWhat*)+"` and should work as previous.\*
3. Variables and Double Quotes Before these changes, if a variable's value contained one or more double quotes, it was no reason for concern. It is now !
4. LE, GE, NE operators removed. The code supported these operators, but they were not documented. The symbolic operators, =, =, and != should be used instead.
5. Added the unary '-' operator. So you can `3+ -4` and get `-1`.
6. Added the unary '!' operator, which is a logical complement. Basically, if the string or number is null, empty, or '0', a '1' is returned. Otherwise a '0' is returned.
7. Added the '=~' operator, just in case someone is just looking for match anywhere in the string. The only diff with the '!' is that match doesn't have to be anchored to the beginning of the string.
8. Added the conditional operator 'expr1 ? true\_expr :: false\_expr' First, all 3 exprs are evaluated, and if expr1 is false, the 'false\_expr' is returned as the result. See above for details.
9. Unary operators '-' and '!' were made right associative.

## Expression Debugging Hints

There are two utilities you can build to help debug the `[$]` in your `extensions.conf` file.

The first, and most simplistic, is to issue the command:

```
make testexpr2
```

in the top level asterisk source directory. This will build a small executable, that is able to take the first command line argument, and run it thru the expression parser. No variable substitutions will be performed. It might be safest to wrap the expression in single quotes...

```
testexpr2 '2*2+2/2'
```

is an example.

And, in the `utils` directory, you can say:

```
make check_expr
```

and a small program will be built, that will check the file mentioned in the first command line argument, for any expressions that might have problems when you move to flex-2.5.31. It was originally designed to help spot possible incompatibilities when moving from the pre-2.5.31 world to the upgraded version of the lexer.

But one more capability has been added to `check_expr`, that might make it more generally useful. It now does a simple minded evaluation of all variables, and then passes the `[$]` exprs to the parser. If there are any parse errors, they will be reported in the log file. You can use `check_expr` to do a quick sanity check of the expressions in your `extensions.conf` file, to see if they pass a crude syntax check.

The "simple-minded" variable substitution replaces `$(varname)` variable references with '555'. You can override the 555 for variable values, by entering in `var=val` arguments after the filename on the command line. So...

```
check_expr /etc/asterisk/extensions.conf CALLERID(num)=3075551212 DIALSTATUS=TORTURE  
EXTEN=121
```

will substitute any `$(CALLERID(num))` variable references with 3075551212, any `$(DIALSTATUS)` variable references with 'TORTURE', and any `$(EXTEN)` references with '121'. If there is any fancy stuff going on in the reference, like `$(EXTEN:2)`, then the override will not work. Everything in the `$(...)` has to match. So, to substitute `$(EXTEN:2)` references, you'd best say:

```
check_expr /etc/asterisk/extensions.conf CALLERID(num)=3075551212 DIALSTATUS=TORTURE  
EXTEN:2=121
```

on stdout, you will see something like:

```
OK - $[ "${DIALSTATUS}" = "TORTURE" | "${DIALSTATUS}" = "DONTCALL" ] at line 416
```

In the `expr2_log` file that is generated, you will see:

```
line 416, evaluation of $[ "TORTURE" = "TORTURE" | "TORTURE" = "DONTCALL" ] result: 1
```

`check_expr` is a very simplistic algorithm, and it is far from being guaranteed to work in all cases, but it is hoped that it will be useful.

## Conditional Applications

There is one conditional application - the conditional goto :

```
exten => 1,2,GotoIf(condition?label1:label2)
```

If condition is true go to label1, else go to label2. Labels are interpreted exactly as in the normal goto command.

"condition" is just a string. If the string is empty or "0", the condition is considered to be false, if it's anything else, the condition is true. This is designed to be used together with the expression syntax described above, eg :

```
exten => 1,2,GotoIf(${${CALLERID(all)} = 123456}?2,1:3,1)
```

Example of use :

```
exten => s,2,Set(vara=1)
exten => s,3,Set(varb=${${vara} + 2})
exten => s,4,Set(varc=${${varb} * 2})
exten => s,5,GotoIf(${${varc} = 6}?99,1:s,6)
```

# Features

The Asterisk core provides a set of features that once enabled can be activated through DTMF codes (also known as feature codes).

Features are configured in features.conf and most require additional configuration via arguments or options to applications that invoke channel creation.



Versions of Asterisk older than 12 included parking configuration inside features.conf. In Asterisk 12 parking configuration was moved out into res\_parking.conf.

The core features discussed in this section are:

- [Feature Code Call Transfers](#)
  - Blind transfers
  - Attended transfers and variations.
- [One-Touch Features](#)
  - This includes instructions for call recording, disconnect and quick parking.
- [Call Pickup](#)
  - Feature code call pickup as well as dialplan application-based call pickup.
- [Built-in Dynamic Features](#)
  - How to use a couple of functions to set built-in feature codes on a per-channel basis.
- [Custom Dynamic Features](#)
  - How to define custom features and set them on a per-channel basis using a channel variable.
- [Call Parking](#)
  - Instructions for how to implement parking lots (with examples).

The only features discussed in this section are those that have some relation to features.conf. Features in a broader sense - that is features that your application built with Asterisk may have - are implemented through usage of [Applications](#), [Functions](#) and [Interfaces](#) or [Dialplan](#).

# Feature Code Call Transfers

## Overview of Feature Code Call Transfers

A call transfer is when one party of a call directs Asterisk to connect the other party to a new location on the system.

Transfer types supported by the Asterisk core:

- Blind transfer
- Attended transfer
  - Variations on attended transfer behavior

Transfer features provided by the Asterisk core are configured in `features.conf` and accessed with feature codes.

Channel driver technologies such as `chan_sip` and `chan_pjsip` have native capability for various transfer types. That native transfer functionality is independent of this core transfer functionality. The core feature code transfer functionality is channel agnostic.

### Blind transfer

A blind or unsupervised transfer is where the initiating party is blind to what is happening after initiating the transfer. They are removed from the process as soon as they initiate the transfer. It is a sort of "fire and forget" transfer.

### Attended transfer

An attended or supervised transfer happens when one party transfers another party to a new location by first dialing the transfer destination and only completing the transfer when ready. The initiating party is attending or supervising the transfer process by contacting the destination before completing the transfer. This is helpful if the transfer initiator wants to make sure someone answers or is ready at the destination.

## Configuring Transfer Features

There are three primary requirements for the use of core transfer functionality.

- The transfer type must be enabled and assigned a DTMF digit string in `features.conf` or per channel - see [\(\(Dynamic DTMF Features\)\)](#)
- The channel must allow the type of transfer attempted. This can be configured via the Application invoking the channel such as `Dial` or `Queue`.
- The channels involved must be answered and bridged.

### Enabling blind or attended transfers

In `features.conf` you must configure the `blindxfer` or `atxfer` options in the `featuremap` section. The options are configured with the DTMF character string you want to use for accessing the feature.

```
[featuremap]
blindxfer = #1
atxfer = *2
```

Now that you have the feature enabled you need to configure the dialplan such that a particular channel will be allowed to use the feature.

As an example if you want to allow transfers via the `Dial` application you can use two options, "t" or "T".

- t - Allow the called party to transfer the calling party by sending the DTMF sequence defined in `features.conf`. This setting does not perform policy enforcement on transfers initiated by other methods
- T - Allow the calling party to transfer the called party by sending the DTMF sequence defined in `features.conf`. This setting does not perform policy enforcement on transfers initiated by other methods.

Setting these options for `Dial` in `extensions.conf` would look similar to the following:

```
exten = 102,1,Dial(PJSIP/BOB,30,T)
```

Asterisk should be restarted or relevant modules should be reloaded for changes to take effect.



The same arguments ("t" and "T") work for the `Queue` and `Dial` applications!

### On this Page

- Overview of Feature Code Call Transfers
  - Blind transfer
  - Attended transfer
- Configuring Transfer Features
  - Enabling blind or attended transfers
  - Feature codes for attended transfer control
  - Configuring attended transfer callbacks
- Behavior Options
- Basic Transfer Examples
  - Making a blind transfer
  - Making an attended transfer



## Feature codes for attended transfer control

There are a few additional feature codes related to attended transfers. These features allow you to vary the behavior of an attended transfer on command. They are all configured in the 'general' section of features.conf

### Aborting an attended transfer

Dialing the **atxferabort** code aborts an attended transfer. Otherwise there is no way to abort an attended transfer.

### Completing an attended transfer

Dialing the **atxfercomplete** code completes an attended transfer and drops out of the call without having to hang up.

### Completing an attended transfer as a three-way bridge

Dialing the **atxferthreeway** code completes an attended transfer and enters a bridge with both of the other parties.

### Swapping between the transferee and destination

Dialing the **atxferswap** code swaps you between bridges with either party before the transfer is complete. This allows you to talk to either party one at a time before finalizing the attended transfer.

### Example configuration

```
[general]
atxferabort = *3
atxfercomplete = *4
atxferthreeway = *5
atxferswap = *6
```

## Configuring attended transfer callbacks

By default Asterisk will call back the initiator of the transfer if they hang up before the target answers and the answer timeout is reached. There are a few options for configuring this behavior.

### No answer timeout

**atxfernoanswertimeout** allows you to define the timeout for attended transfers. This is the amount of time (in seconds) Asterisk will attempt to ring the target before giving up.

### Dropped call behavior

**atxferdropcall** allows you to change the default callback behavior. The default is 'no' which results in Asterisk calling back the initiator of a transfer when they hang up early and the attended transfer times out. If set to 'yes' then the transfer target channel will be immediately transferred to the channel being transferred as soon as the initiator hangs up.

### Loop delay timing

**atxferloopdelay** sets the number of seconds to wait between callback retries. This option is only relevant when atxferdropcall=no (or is undefined).

### Number of retries for callbacks

**atxfercallbackretries** sets the number of times Asterisk will try to send a failed attended transfer back to the initiator. The default is 2.

### Example Configuration

```
[general]
atxfernoanswertimeout = 15
atxferdropcall = no
atxferloopdelay = 10
atxfercallbackretries = 2
```

## Behavior Options

These options are configured in the "[general]" section of features.conf

## General transfer options

```
;transferdigittimeout = 3 ; Number of seconds to wait between digits when transferring a call  
; (default is 3 seconds)
```

## Attended transfer options

```
;xfersound = beep ; to indicate an attended transfer is complete  
;xferfailsound = beeperr ; to indicate a failed transfer  
;transferdialattempts = 3 ; Number of times that a transferer may attempt to dial an extension before  
; being kicked back to the original call.  
;transferretrysound = "beep" ; Sound to play when a transferer fails to dial a valid extension.  
;transferinvalidsound = "beeperr" ; Sound to play when a transferer fails to dial a valid extension and is out of retries.
```

## Basic Transfer Examples

In the previous section we configured #1 and \*2 as our features codes. We also passed the "T" argument in the Dial application at 102 to allow transfers by the calling party.

Our hypothetical example includes a few devices:

- PJSIP/ALICE at extension 101
- PJSIP/BOB at extension 102
- PJSIP/CATHY at extension 103

### Making a blind transfer

For blind transfers we configured the #1 feature code.

An example call flow:

- ALICE dials extension 102 to call BOB
- ALICE decides to transfer BOB to extension 103, so she dials #1. Asterisk will play the audio prompt "transfer".
- ALICE enters the digits 103 for the destination extension.
- Asterisk immediately hangs up the channel between ALICE and BOB. Asterisk creates a new channel for BOB that is dialing extension 103.

### Making an attended transfer

For attended transfers we configured \*2 as our feature code.

An example call flow:

- ALICE dials extension 102 to call BOB and BOB answers.
- ALICE decides to transfer BOB to extension 103, so she dials \*2. Asterisk plays the audio prompt "transfer".
- ALICE enters the digits 103 for the destination extension. Asterisk places BOB on hold and creates a channel for ALICE to dial CATHY.
- CATHY answers - ALICE and CATHY talk. ALICE decides to complete the transfer and hangs up the phone.
- Asterisk immediately hangs up the channel between ALICE and BOB. Asterisk plays a short beep tone to CATHY and then bridges the channels for BOB and CATHY.

# One-Touch Features

## Overview

Once configured these features can be activated with only a few or even one keypress on a user's phone. They are often called "one-touch" or "one-step" features.

All of the features are configured via options in the featuremap section of features.conf and require arguments to be passed to the applications invoking the target channel.

## Available Features

- **automon** - (One-touch Recording) Asterisk will invoke Monitor on the current channel.
- **automixmon** - (One-touch Recording) Has the same behavior as automon, but uses MixMonitor instead of Monitor.
- **disconnect** - (One-touch Disconnect) When this code is detected on a channel that channel will be immediately hung up.
- **parkcall** - (One-touch Parking) Sets a feature code for quickly parking a call.
  - Most parking options and behavior are configured in res\_parking.conf in Asterisk 12 and newer.

## Enabling the Features

### Configuration of features.conf

The options are configured in features.conf in the featuremap section. They use typical Asterisk [configuration file syntax](#).

```
features.conf

[featuremap]
automon = *1
automixmon = *3
disconnect = *0
parkcall = #72
```

Assign each option the DTMF character string that you want users to enter for invoking the feature.

### Dialplan application options

For each feature there are a pair of options that can be set in the [Dial](#) or [Queue](#) applications. The two options enable the feature on either the calling party channel or the called party channel.



If neither option of a pair are set then you will not be able to use the related feature on the channel.

#### automon

- W - Allow the calling party to enable recording of the call.
- w - Allow the called party to enable recording of the call.

#### automixmon

- X - Allow the calling party to enable recording of the call.
- x - Allow the called party to enable recording of the call.

#### disconnect

- H - Allow the calling party to hang up the channel.
- h - Allow the called party to hang up the channel.

#### parkcall

- K - Allow the calling party to enable parking of the call.
- k - Allow the called party to enable parking of the call.

### Example usage

Set the option as you would any [application](#) option.

## extensions.conf

```
exten = 101,1,Dial(PJSIP/ALICE,30,X)
```

This would allow the calling party, the party dialing PJSIP/ALICE, to invoke recording on the channel.

## Using the Feature

Once you have configured features.conf and the options in the application then you only have to enter the feature code on your phone's keypad during a call!

# Call Pickup



Call pickup support added in Asterisk 11

## Overview

Call pickup allows you to answer a call while it is ringing another phone or group of phones (other than the phone you are sitting at).

Requesting to pickup a call is done by two basic methods.

1. by `dialplan` using the `Pickup` or `PickupChan` applications.
2. by dialing the extension defined for `pickupexten` configured in `features.conf`.

Which calls can be picked up is determined by configuration and `dialplan`.

## Dialplan Applications and Functions

### Pickup Application

The `Pickup` application has three ways to select calls for pickup.

### PickupChan Application

The `PickupChan` application tries to pickup the specified channels given to it.

### CHANNEL Function

The `CHANNEL` function allows the pickup groups set on a channel to be changed from the defaults set by the channel driver when the channel was created.

#### `callgroup/namedcallgroup`

The `CHANNEL(callgroup)` option specifies which numeric pickup groups that this channel is a member.

```
same => n, Set (CHANNEL(callgroup)=1,5-7)
```

The `CHANNEL(namedcallgroup)` option specifies which named pickup groups that this channel is a member.

```
same => n, Set (CHANNEL(namedcallgroup)=engineering,sales)
```



For this option to be effective, you must set it on the outgoing channel. There are a couple of ways:

- You can use the `setvar` option available with several channel driver configuration files to set the pickup groups.
- You can use a pre-dial handler.

#### `pickupgroup/namedpickupgroup`

The `CHANNEL(pickupgroup)` option specifies which numeric pickup groups this channel can pickup.

```
same => n, Set (CHANNEL(pickupgroup)=1,6-8)
```

The `CHANNEL(namedpickupgroup)` option specifies which named pickup groups this channel can pickup.

```
same => n, Set (CHANNEL(namedpickupgroup)=engineering,sales)
```



For this option to be effective, you must set it on the channel before executing the `Pickup` application or calling the `pickupexten`.

- You can use the `setvar` option available with several channel driver configuration files to set the pickup groups.

## Configuration Options

The pickupexten request method selects calls using the numeric and named call groups. The ringing channels have the callgroup assigned when the channel is created by the channel driver or set by the CHANNEL(callgroup) or CHANNEL(namedcallgroup) dialplan function.

Calls picked up using pickupexten can hear an optional sound file for success and failure.



The current channel drivers that support calling the pickupexten to pickup a call are: chan\_dahdi/analog, chan\_mgcp, chan\_misdn, chan\_sip, chan\_unistim and chan\_pjsip.

### features.conf

```
pickupexten = *8           ; Configure the pickup extension. (default is *8)
pickupsound = beep         ; to indicate a successful pickup (default: no sound)
pickupfailsound = beeperr  ; to indicate that the pickup failed (default: no sound)
```

## Numeric call pickup groups

A numeric callgroup and pickupgroup can be set to a comma separated list of ranges (e.g., 1-4) or numbers that can have a value of 0 to 63. There can be a maximum of 64 numeric groups.

### SYNTAX

```
callgroup=[number[-number][,number[-number][, ...]]]
pickupgroup=[number[-number][,number[-number][, ...]]]
```

- callgroup - specifies which numeric pickup groups that this channel is a member.
- pickupgroup - specifies which numeric pickup groups this channel can pickup.

### Configuration example

```
callgroup=1,5-7
pickupgroup=1
```

Configuration should be supported in several channel drivers, including:

- chan\_dahdi.conf
- misdn.conf
- mgcp.conf
- sip.conf
- unistim.conf
- pjsip.conf

pjsip.conf uses snake case:

### Configuration in pjsip.conf

```
call_group=1,5-7
pickup_group=1
```

## Named call pickup groups

A named callgroup and pickupgroup can be set to a comma separated list of case sensitive name strings. The number of named groups is unlimited. The number of named groups you can specify at once is limited by the line length supported.

### SYNTAX

```
namedcallgroup=[name[,name[, ...]]]
namedpickupgroup=[name[,name[, ...]]]
```

- namedcallgroup - specifies which named pickup groups that this channel is a member.
- namedpickupgroup - specifies which named pickup groups this channel can pickup.

## Configuration Example

```
namedcallgroup=engineering,sales,netgroup,protgroup  
namedpickupgroup=sales
```

Configuration should be supported in several channel drivers, including:

- chan\_dahdi.conf
- misdn.conf
- sip.conf
- pjsip.conf

pjsip.conf uses snake case:

```
named_call_group=engineering,sales,netgroup,protgroup  
named_pickup_group=sales
```



You can use named pickup groups in parallel with numeric pickup groups. For example, the named pickup group '4' is not the same as the numeric pickup group '4'.

Named pickup groups are new with Asterisk 11.

## Built-in Dynamic Features

The `FEATURE` and `FEATUREMAP` dialplan functions allow you to set some `features.conf` options on a per channel basis.



To see what options are currently supported, look at the `FEATURE` and `FEATUREMAP` function descriptions. **These functions were added in Asterisk 11.**

At this time the functions do not work with custom features. Those are set with a channel variable as described in the [Custom Dynamic Features](#) section.

### Set the parking time of this channel to be 100 seconds if it is parked.

```
exten => s,1,Set(FEATURE(parkingtime)=100)
same => n,Dial(SIP/100)
same => n,Hangup()
```

### Set the DTMF sequence for attended transfer on this channel to \*9.

```
exten => s,1,Set(FEATUREMAP(atxfer)=*9)
same => n,Dial(SIP/100,,T)
same => n,Hangup()
```



# Custom Dynamic Features

## Overview

Asterisk allows you to define custom features mapped to Asterisk applications. You can then enable these features dynamically, on a per-channel basis by using a channel variable.

## Defining the Features

Custom features are defined in the **applicationmap** section of the features.conf file.

Syntax:

```
[applicationmap]
<FeatureName> = <DTMF_sequence>,<ActivateOn>[/<ActivatedBy>],<Application>[,<AppArguments>[,MOH_Class]]
<FeatureName> = <DTMF_sequence>,<ActivateOn>[/<ActivatedBy>],<Application>[, "<AppArguments>" [,MOH_Class]]
<FeatureName> = <DTMF_sequence>,<ActivateOn>[/<ActivatedBy>],<Application>({<AppArguments>})[,MOH_Class]
```

Syntax Fields:

Field Name	Description
FeatureName	This is the name of the feature used when setting the DYNAMIC_FEATURES variable to enable usage of this feature.
DTMF_sequence	This is the key sequence used to activate this feature.
ActivateOn	This is the channel of the call that the application will be executed on. Valid values are "self" and "peer". "self" means run the application on the same channel that activated the feature. "peer" means run the application on the opposite channel from the one that has activated the feature.
ActivatedBy	ActivatedBy is no longer honored. The feature is activated by which channel DYNAMIC_FEATURES includes the feature is on. Use predial to set different values of DYNAMIC_FEATURES on the channels. Historic values are: "caller", "callee", and "both".
Application	This is the application to execute.
AppArguments	These are the arguments to be passed into the application. If you need commas in your arguments, you should use either the second or third syntax, above.
MOH_Class	This is the music on hold class to play while the idle channel waits for the feature to complete. If left blank, no music will be played.

## Application Mapping

The applicationmap is not intended to be used for all Asterisk applications. When applications are used in extensions.conf, they are executed by the PBX core. In this case, these applications are executed outside of the PBX core, so it does \*not\* make sense to use any application which has any concept of dialplan flow. Examples of this would be things like Goto, Background, WaitExten, and many more. The exceptions to this are Gosub and Macro routines which must complete for the call to continue.

Enabling these features means that the PBX needs to stay in the media flow and media will not be re-directed if DTMF is sent in the media stream.

## Example Feature Definitions:

Here we have defined a few custom features to give you an idea of how the configuration looks.

### features.conf

```
[applicationmap]
playmonkeys => #9,peer,Playback,tt-monkeys
retrieveinfo => #8,peer,Set(ARRAY(CDR(mark),CDR(name))=${ODBC_FOO(${CALLERID(num)}}))
pauseMonitor => #1,self/callee,PauseMonitor
unpauseMonitor => #3,self/callee,UnPauseMonitor
```

Example feature descriptions:

- playmonkeys - Allow both the caller and callee to play tt-monkeys to the bridged channel.
- retrieveinfo - Set arbitrary channel variables, based upon CALLERID number (Note that the application argument contains commas)
- pauseMonitor - Allow the callee to pause monitoring on their channel.

- unpauseMonitor - Allow the callee to unpause monitoring on their channel.

## Enabling Features

After you define a custom feature in features.conf you must enable it on a channel by setting the DYNAMIC\_FEATURES channel variable.

DYNAMIC\_FEATURES accepts as an argument a list of hash-sign delimited feature names.

Example Usage:

### extensions.conf

```
Set (__DYNAMIC_FEATURES=playmonkeys#pauseMonitor#unpauseMonitor)
```



#### Tip: Variable Inheritance

The two leading underscores allow these feature settings to be set on the outbound channels, as well. Otherwise, only the original channel will have access to these features.

# Call Parking

## Overview

Some organizations have the need to facilitate calls to employees who move around the office a lot or who don't necessarily sit at a desk all day. In Asterisk, it is possible to allow a call to be put on hold at one location and then picked up from a different location such that the conversation can be continued from a device other than the one from which call was originally answered. This concept is known as call parking.

Call parking is a feature that allows a participant in a call to put the other participants on hold while they themselves hang up. When parking, the party that initiates the park will be told a parking space, which under standard configuration doubles as an extension. This extension, or parking space, serves as the conduit for accessing the parked call. At this point, as long as the parking space is known, the parked call can be retrieved from a different location/device from where it was originally answered.

## Call Parking Configuration Files and Module

In versions of Asterisk prior to Asterisk 12, call parking was considered an Asterisk core feature and was configured using `features.conf`. However, Asterisk 12 underwent vast architectural changes, several of which were directed at call parking support. Because the amount of changes introduced in Asterisk 12 was quite extensive, they have been omitted from this document. For reference, you can find a comprehensive list of these changes here: [New in 12](#).

In a nutshell, Asterisk 12 relocated its support for call parking from the Asterisk core into a separate, loadable module, `res_parking`. As a result, configuration for call parking was also moved to `res_parking.conf`. Configuration for call parking through `features.conf` for versions of Asterisk 12 and beyond, is no longer supported. Additionally, support for the `ParkAndAnnounce` application was relocated to the `res_parking` module and the `app_parkandannounce` module was removed.

Before we move any further, there is one more rather important detail to address regarding configuration for `res_parking`:



`res_parking` uses the configuration framework. If an invalid configuration is supplied, `res_parking` will fail to load or fail to reload. Previously, invalid configurations would generally be accepted, with certain errors resulting in individually disabled parking lots.

Now that we've covered all of that, let's look at some examples of how all this works.

### On This Page

- [Overview](#)
- [Call Parking Configuration Files and Module](#)
- [Example Configurations](#)
  - [Basic Call Parking/Retrieval Scenario](#)
  - [Basic Handling for Call Parking Timeouts](#)
  - [Custom Handling for Call Parking Timeouts](#)

## Example Configurations

### Basic Call Parking/Retrieval Scenario

This is a basic scenario that only requires minimal adjustments to the following configuration files: `res_parking.conf`, `features.conf`, and `extensions.conf`.

In this scenario, our dialplan contains an extension to accept calls from the outside. Let's assume that the extension the caller dialed was: `5555001`. The handler will then attempt to dial the `alice` extension, using the `x` option.

Sadly for our caller, the `alice` extension answers the call and immediately after saying, "Hello world!", sends the DTMF digits to invoke the call parking feature without giving the caller a chance to speak. The `alice` extension quickly redeems itself by using the `GOTo` application to navigate to the `701` extension in the `parkedcalls` context to retrieve the parked call. But, since the next thing the `alice` extension does is hangup on our caller, I am beginning to think the `alice` extension doesn't want to be bothered.

**In summary:**

- Outside caller dials `5555001`
- Alice picks up the device and says "Hello World!"
- Alice presses the one touch parking DTMF combination
- Alice then dials the extension that the call was parked to ( `701` ) to retrieve the call
- Alice says, "Goodbye", and disconnects the caller

## res\_parking.conf

```
[general]
parkext => 700 ; Sets the default extension used to park calls.
Note: This option ; can take any alphanumeric string.

parkpos => 701-709 ; Sets the range of extensions used as the
parking lot. Parked calls ; may be retrieved by dialing the numbers in
this range. Note: These ; need to be numeric, as Asterisk starts from
the start position and ; increments with one for the next parked call.

context => parkedcalls ; Sets the default dialplan context where the
parking extension and ; the parking lot extensions are created. These
will be automatically ; generated since we have specified a value for
the 'parkext' option ; above. If you need to use this in your
dialplan (extensions.conf), ; just include it like: include => parkedcalls.

parkingtime => 300 ; Specifies the number of seconds a call will
wait in the parking ; lot before timing out. In this example, a
parked call will time out ; if it is not un-parked before 300 seconds (5
minutes) elapses.

findslot => next ; Configures the parking slot selection
behavior. For this example, ; the next free slot will be selected when a
call is parked.
```

## features.conf

```
[featuremap]
parkcall => #72 ; Parks the call (one-step parking). For this
example, a call will be ; automatically parked when an allowed party
presses the DTMF digits, ; #.7.2. A party is able to make use of this
when the the K/k options ; are used when invoking the Dial() application.
For convenience, the ; values of this option are defined below:
of the call. ; K - Allow the calling party to enable parking
of the call. ; k - Allow the called party to enable parking
```

## extensions.conf

```
[globals]
; Extension Maps
5001=alice                                ; Maps 5001 to a local extension that will
emulate                                   ; a party pressing DTMF digits from a device.
                                           ; What a realistic mapping for the alice device
                                           ; would look like.

; Realistically, 'alice' would map to a channel for a local device that would receive the
call, therefore
; rendering this extension unnecessary. However, for the purposes of this demonstration,
the extension is
; presented to you to show that sending the sequence of DTMF digits defined in the
'parkcall' option in
; 'features.conf' is the trigger that invokes the one-step parking feature.

[parking-example]
include => parkedcalls

exten => alice,1,NoOp(Handles calls to alice.)
    same => n,Answer()
    same => n,Playback(hello-world)
    same => n,SendDTMF(#72w)
    same => n,Goto(parkedcalls,701,1)
    same => n,Playback(vm-goodbye)
    same => n,Hangup()

[from-outside]
exten => 5555001,1,NoOp(Route to a local extension.)
    ; Dials the device that is mapped to the local resource, alice, giving the recipient of
the call the ability
    ; to park it. Assuming the value of LocalExtension is 5001, the Dial() command will
look like: Dial(alice,,k)
    same => n,Dial(PJSIP/alice)
    same => n,Hangup()
```

## Basic Handling for Call Parking Timeouts

Next we will move on to explain how to handle situations where a call is parked but is not retrieved before the value specified as the `parkingtime` option elapses. Just like the scenario above, this is a basic scenario that only requires minimal adjustments to the following configuration files: `res_parking.conf`, `features.conf`, and `extensions.conf`.

Like before, our dialplan contains an extension to accept calls from the outside. Again, let's assume that the extension the caller dialed was: `5555001`. The handler will then attempt to dial the `alice` extension, using the `k` option.

Sadly for our caller, the `alice` extension answers the call and immediately sends the DTMF digits to invoke the call parking feature without giving the caller a chance to speak. Unlike in the previous scenario, however, the `alice` extension does not retrieve the parked call. Our sad caller is now even more sad.

After a period of `300 seconds`, or `5 minutes` (as defined in the `parkingtime` option in `res_parking.conf`), the call will time out. Because we told Asterisk to return a timed-out parked call to the party that originally parked the call (`comebacktoorigin=yes`), Asterisk will attempt to call `alice` using an extension automatically created in the special context, `park-dial`.

Unfortunately, the `alice` extension has no time to be bothered with us at this moment, so the call is not answered. After a period of `20 seconds` elapses (the value specified for the `comebackdialtime` option in `res_parking.conf`), Asterisk finally gives up and the `t` extension in the `park-dial` context is executed. Our caller is then told "Goodbye" before being disconnected.

### In summary:

- Outside caller dials `5555001`
- Alice picks up the device and says "Hello World!"

- Alice presses the one touch parking DTMF combination
- The parked call times out after 300 seconds
- Asterisk sends the call to the origin, or the `alice` extension
- A period of 20 seconds elapses without an answer
- Asterisk sends the call to `t` extension in the `park-dial` context
- Our caller hears, "Goodbye", before being disconnected

## res\_parking.conf

```
[general]
parkext => 700 ; Sets the default extension used to park calls.
Note: This option ; can take any alphanumeric string.

parkpos => 701-709 ; Sets the range of extensions used as the
parking lot. Parked calls ; may be retrieved by dialing the numbers in
this range. Note: These ; need to be numeric, as Asterisk starts from
the start position and ; increments with one for the next parked call.

context => parkedcalls ; Sets the default dialplan context where the
parking extension and ; the parking lot extensions are created. These
will be automatically ; generated since we have specified a value for
the 'parkext' option ; above. If you need to use this in your
dialplan (extensions.conf), ; just include it like: include => parkedcalls.

parkingtime => 300 ; Specifies the number of seconds a call will
wait in the parking ; lot before timing out. In this example, a
parked call will time out ; if it is not un-parked before 300 seconds (5
minutes) elapses.

findslot => next ; Configures the parking slot selection
behavior. For this example, ; the next free slot will be selected when a
call is parked.

comebackdialtime=20 ; When a parked call times out, this is the
number of seconds to dial ; the device that originally parked the call, or
the PARKER ; channel variable. The value of
'comebackdialtime' is available as ; the channel variable 'COMEBACKDIALTIME' after
a parked call has ; timed out. For this example, when a parked
call times out, Asterisk ; will attempt to call the PARKER for 20
seconds, using an extension ; it will automatically create in the
'park-dial' context. If the ; party does not answer the call during this
period, Asterisk will
```

```
the dialplan.                                ; continue executing any remaining priorities in

comebacktoorigin=yes                          ; Determines what should be done with a parked
call if it is not                             ; retrieved before the time specified in the
'parkingtime' option                          ; elapses. In the case of this example where
'comebacktoorigin=yes',                      ; Asterisk will attempt to return the parked
call to the party that                       ; originally parked the call, or the PARKER
channel variable, using                      ; an extension it will automatically create in
```

```
the 'park-dial'
; context.
```

### features.conf

```
[featuremap]
parkcall => #72 ; Parks the call (one-step parking). For this
example, a call will be ; automatically parked when an allowed party
presses the DTMF digits, ; #·7·2. A party is able to make use of this
when the the K/k options ; are used when invoking the Dial() application.
For convenience, the ; values of this option are defined below:
of the call. ; K - Allow the calling party to enable parking
of the call. ; k - Allow the called party to enable parking
```



## extensions.conf

```
[globals]
; Extension Maps
5001=alice                                ; Maps 5001 to a local extension that will
emulate                                   ; a party pressing DTMF digits from a device.
                                           ; What a realistic mapping for the alice device
                                           ; would look like.

; Realistically, 'alice' would map to a channel for a local device that would receive the
call, therefore
; rendering this extension unnecessary. However, for the purposes of this demonstration,
the extension is
; presented to you to show that sending the sequence of DTMF digits defined in the
'parkcall' option in
; 'features.conf' is the trigger that invokes the one-step parking feature.

[parking-example]
include => parkedcalls

exten => alice,1,NoOp(Handles calls to alice.)
    same => n,Answer()
    same => n,Playback(hello-world)
    same => n,SendDTMF(#72w)
    same => n,Wait(300)
    same => n,Hangup()

[from-outside]
exten => 5555001,1,NoOp(Route to a local extension.)
    ; Dials the device that is mapped to the local resource, alice, giving the recipient of
the call the ability
    ; to park it. Assuming the value of LocalExtension is 5001, the Dial() command will
look like: Dial(alice,,k)
    same => n,Dial(PJSIP/alice)
    same => n,Hangup()

[park-dial]
; Route here if the party that initiated the call parking cannot be reached after a
period of time equaling the
; value specified in the 'comebackdialtime' option elapses.
exten => t,1,NoOp(End of the line for a timed-out parked call.)
    same => n,Playback(vm-goodbye)
    same => n,Hangup()
```

## Custom Handling for Call Parking Timeouts

Finally, we will move on to explain how to handle situations where upon a parked call session timing out, it is not desired to return to the parked call to the device from where the call was originally parked. (This might be handy for situations where you have a dedicated receptionist or service desk extension to handle incoming call traffic.) Just like the previous two examples, this is a basic scenario that only requires minimal adjustments to the following configuration files: `res_parking.conf`, `features.conf`, and `extensions.conf`.

Like before, our dialplan contains an extension to accept calls from the outside. Again, let's assume that the extension the caller dialed was: `5555001`. The handler will then attempt to dial the `alice` extension, using the `k` option.

Sadly for our caller, the `alice` extension answers the call and immediately sends the DTMF digits to invoke the call parking feature without giving the caller a chance to speak. Just like in the previous scenario, the `alice` extension does not retrieve the parked call. Maybe the `alice` extension is having a bad day.

After a period of `300 seconds`, or `5 minutes` (as defined in the `parkingtime` option in `res_parking.conf`), the call will time out. Because we told Asterisk to send a timed-out parked call to the `parkedcalltimeout` context (`comebacktoorigin=no`), we are able to bypass the default logic

that directs Asterisk to returning the call to the person who initiated the park. In our example, when a parked call enters our `s` extension in our `parkedcallsttimeout` context, we only play a sound file to the caller and hangup the call, but this is where you could do any custom logic like returning the call to a different extension, or performing a lookup of some sort.

**In summary:**

- Outside caller dials 5555001
- Alice picks up the device and says "Hello World!"
- Alice presses the one touch parking DTMF combination
- The parked call times out after 300 seconds
- Asterisk sends the call to the `s` extension in our `parkedcallsttimeout`
- Our caller hears, "Goodbye", before being disconnected

```
res_parking.conf

[general]

[default]
parkext => 700 ; Sets the default extension used to park calls.
Note: This option ; can take any alphanumeric string.

parkpos => 701-709 ; Sets the range of extensions used as the
parking lot. Parked calls ; may be retrieved by dialing the numbers in
this range. Note: These ; need to be numeric, as Asterisk starts from
the start position and ; increments with one for the next parked call.

context => parkedcalls ; Sets the default dialplan context where the
parking extension and ; the parking lot extensions are created. These
will be automatically ; generated since we have specified a value for
the 'parkext' option ; above. If you need to use this in your
dialplan (extensions.conf), ; just include it like: include => parkedcalls.

parkingtime => 300 ; Specifies the number of seconds a call will
wait in the parking ; lot before timing out. In this example, a
parked call will time out ; if it is not un-parked before 300 seconds (5
minutes) elapses.

findslot => next ; Configures the parking slot selection
behavior. For this example, ; the next free slot will be selected when a
call is parked.

comebacktoorigin=no ; Determines what should be done with a parked
call if it is not ; retrieved before the time specified in the
'parkingtime' option ; elapses.
;
; Setting 'comebacktoorigin=no' (like in this
example) is for cases ; when you want to perform custom dialplan logic
```

to gracefully handle  
out.  
comebackcontext=parkedcallstimeout  
to in the event it  
route the call to an  
flattened peer name. If  
attempt to route the  
Note: If you set  
but do not define this

```
; the remainder of the parked call when it times  
; The context that a parked call will be routed  
; times out. Asterisk will first attempt to  
; extension in this context that matches the  
; no such extension exists, Asterisk will next  
; call to the 's' extension in this context.  
; 'comebacktoorigin=no' in your configuration  
; value, Asterisk will route the call to the 's'
```

extension in the

; default context.

### features.conf

[featuremap]

parkcall => #72

example, a call will be

presses the DTMF digits,

when the the K/k options

For convenience, the

of the call.

of the call.

; Parks the call (one-step parking). For this

; automatically parked when an allowed party

; #.7.2. A party is able to make use of this

; are used when invoking the Dial() application.

; values of this option are defined below:

; K - Allow the calling party to enable parking

; k - Allow the called party to enable parking

## extensions.conf

```
[globals]
; Extension Maps
5001=alice                               ; Maps 5001 to a local extension that will
emulate                                  ; a party pressing DTMF digits from a device.
                                           ; What a realistic mapping for the alice device
                                           ; would look like.

; Realistically, 'alice' would map to a channel for a local device that would receive the
call, therefore
; rendering this extension unnecessary. However, for the purposes of this demonstration,
the extension is
; presented to you to show that sending the sequence of DTMF digits defined in the
'parkcall' option in
; 'features.conf' is the trigger that invokes the one-step parking feature.

[parking-example]
include => parkedcalls

exten => alice,1,NoOp(Handles calls to alice.)
    same => n,Answer()
    same => n,Playback(hello-world)
    same => n,SendDTMF(#72w)
    same => n,Wait(300)
    same => n,Hangup()

[from-outside]
exten => 5555001,1,NoOp(Route to a local extension.)
    ; Dials the device that is mapped to the local resource, alice, giving the recipient of
the call the ability
    ; to park it. Assuming the value of LocalExtension is 5001, the Dial() command will
look like: Dial(alice,,k)
    same => n,Dial(PJSIP/alice)
    same => n,Hangup()

[parkedcallstimeout]
exten => s,1,NoOp(This is all that happens to parked calls if they time out.)
    same => n,Playback(vm-goodbye)
    same => n,Hangup()
```

# Applications

## Asterisk Dialplan Applications

In this section we'll discuss Asterisk dialplan applications. The concept of Asterisk applications shouldn't be confused with the more general idea of building an "application" with Asterisk, where one is talking about what Asterisk is doing in relation to the outside world.

For example in the general sense; an application of Asterisk could be a voicemail server or PBX. In contrast, an **Asterisk dialplan application** is a unit of functionality provided via an [Asterisk module](#) that can be called via dialplan or one of Asterisk's APIs. Dialplan applications do some work on the [channel](#), such as answering a call or playing back a sound prompt. There are a wide variety of dialplan applications available for your use.

The term **application** in Asterisk documentation and on Asterisk discussion forums is usually referring to dialplan applications.

## Available applications

Many applications come with Asterisk by default. For a complete list of the dialplan applications available to your installation of Asterisk, type **core show applications** at the Asterisk [CLI](#). Not all applications are compiled with Asterisk by default so if you have the source available then you may want to browse the applications listed in [menuselect](#).

Since anyone can write an Asterisk application they can also be obtained from sources outside the Asterisk distribution. Pre-packaged community or commercial Asterisk distributions that have a special purpose may include a custom application or two.

## General application syntax

Each extension priority in the [dialplan](#) calls an application. Most applications take one or more parameters, which provide additional information to the application or change its behavior. Parameters should be separated by commas.

You can find examples of Asterisk extensions and priorities in the [Contexts](#), [Extensions](#), and [Priorities](#) section. You'll notice that most extensions look similar to this example:

```
exten =>
6123,1,ApplicationName(ParamOne,ParamTwo,ParamThree)
```

ApplicationName is where you put the name of an application you want to call.

ParamOne, two and three are the parameters passed to the application. Each is separated by a comma. Some applications don't require any parameters, but most do.



Using the pipe character or vertical bar character (|) to delimit parameters is deprecated syntax. Use commas instead!

## Help for specific applications

The wiki section [CLI Syntax and Help Commands](#) details how to use the CLI-accessible documentation. This will allow you to access the syntax and usage info for each application including detail on all the options for each application.

Wikibot also publishes the same documentation on the wiki. You can find applications docs in the version specific top level sections; such as [Asterisk 13 Dialplan Applications](#).

## Commonly used applications

There are many Asterisk applications, but several are very commonly used and essential to almost every Asterisk system. The sub-pages in the section will provide some further detail and usage of these common Asterisk applications.

### On This Page

- [Asterisk Dialplan Applications](#)
  - [Available applications](#)
  - [General application syntax](#)
  - [Help for specific applications](#)
  - [Commonly used applications](#)

### Topics

- [Answer, Playback, and Hangup Applications](#)
- [Bridge Application](#)
- [Dial Application](#)
- [Directory Application](#)
- [Early Media and the Progress Application](#)
- [External IVR Interface](#)
- [MacroExclusive](#)
- [The Read Application](#)
- [The Verbose and NoOp Applications](#)
- [SMS](#)
- [Voicemail](#)
- [Conferencing Applications](#)

## Answer, Playback, and Hangup Applications

As its name suggests, the **Answer()** application answers an incoming call. The **Answer()** application takes a delay (in milliseconds) as its first parameter. Adding a short delay is often useful for ensuring that the remote endpoint has time to begin processing audio before you play a sound prompt. Otherwise, you may not hear the very beginning of the prompt.

### Knowing When to Answer a Call

When you're first learning your way around the Asterisk dialplan, it may be a bit confusing knowing when to use the **Answer()** application, and when not to.

If Asterisk is simply going to pass the call off to another device using the **Dial()** application, you probably don't want to call the answer the call first. If, on the other hand, you want Asterisk to play sound prompts or gather input from the caller, it's probably a good idea to call the **Answer()** application before doing anything else.

The **Playback()** application loads a sound prompt from disk and plays it to the caller, ignoring any touch tone input from the caller. The first parameter to the dialplan application is the filename of the sound prompt you wish to play, without a file extension. If the channel has not already been answered, **Playback()** will answer the call before playing back the sound prompt, unless you pass **noanswer** as the second parameter.

To avoid the first few milliseconds of a prompt from being cut off you can play a second of silence. For example, if the prompt you wanted to play was hello-world which would look like this in the dialplan:

```
exten => 1234,1,Playback(hello-world)
```

You could avoid the first few seconds of the prompt from being cut off by playing the silence/1 file:

```
exten => 1234,1,Playback(silence/1)
exten => 1234,n,Playback(hello-world)
```

Alternatively this could all be done on the same line by separating the filenames with an ampersand (&):

```
exten => 1234,1,Playback(silence/1&hello-world)
```

The **Hangup()** application hangs up the current call. While not strictly necessary due to auto-fallthrough (see the note on Priority numbers above), in general we recommend you add the **Hangup()** application as the last priority in any extension.

Now let's put **Answer()**, **Playback()**, and **Hangup()** together to play a sample sound file.

```
exten => 6000,1,Answer(500)
exten => 6000,n,Playback(hello-world)
exten => 6000,n,Hangup()
```

# Bridge Application

## Overview of the Bridge Application

The Bridge application takes two channels and attempts to put them into a [Bridge](#). Both channels and bridges are very common elements of Asterisk operation, so this is a really useful application to learn.

For Bridge to work, two channels are required to exist. That is the channel executing the Bridge application and a target channel that you want to bridge with. If the operation is successful the two channels will be bridged and media exchanged among the channels.

To help out let's contrast Dial and Bridge:

- When a channel executes Dial a new channel is created for the target device. If the new channel is answered then both channels are bridged.
- When a channel executes Bridge then Asterisk will attempt to bridge the two existing channels; the executing channel and the target channel.

Note that bridge provides several options to tweak behavior and upon completion Bridge will return status on a channel variable. These are detailed in the [app documentation](#).

## Using the Bridge application

Read the [Bridge documentation](#) for your version of Asterisk (e.g. Asterisk 13 - [Asterisk 13 Application\\_Bridge](#))

and the [Key Concepts](#) section on [Bridges](#) to get a good start.

## See Also

- [Bridging Modules](#)
- [Pre-Bridge Handlers](#)
- [Introduction to ARI and Bridges](#)
- [Asterisk 13 Application\\_BridgeWait](#)
- [Conferencing Applications](#)



# Dial Application

## Overview of the Dial Application

The Dial application is probably the most well known and crucial Asterisk application. Asterisk is often used to interface between communication devices and technologies, and Dial is a simple way to establish a connection from the dialplan. When a channel executes Dial then Asterisk will attempt to contact or "dial" all devices passed to the application. If an answer is received then the two channels will be bridged. Dial provides many options to control behavior and will return results and status of the dial operation on a few channel variables.

## Using the Dial application

Here is a few ways to learn the usage of the Dial application.

- Read the detailed documentation for your Asterisk version: e.g. for Asterisk 13 - [Asterisk 13 Application\\_Dial](#)
- See how to use Dial for a specific channel driver: [Dialing PJSIP Channels](#)
- Note how Dial is used in a simple Asterisk dialplan: [Creating Dialplan Extensions](#)

## See Also

- [Pre-Dial Handlers](#)
- [Hangup Handlers](#)

# Directory Application

The next application we'll cover is named **Directory()**, because it presents the callers with a dial-by-name directory. It asks the caller to enter the first few digits of the person's name, and then attempts to find matching names in the specified voice mail context in **voicemail.conf**. If the matching mailboxes have a recorded name greeting, Asterisk will play that greeting. Otherwise, Asterisk will spell out the person's name letter by letter.

```
Directory([voicemail_context],[dialplan_context],[options]])
```

The **Directory()** application takes three parameters:

## ***voicemail\_context***

This is the context within **voicemail.conf** in which to search for a matching directory entry. If not specified, the **[docs:default]** context will be searched.

## ***dialplan\_context***

When the caller finds the directory entry they are looking for, Asterisk will dial the extension matching their mailbox in this context.

## ***options***

A set of options for controlling the dial-by-name directory. Common options include **f** for searching based on first name instead of last name and **e** to read the extension number as well as the name.



### **Directory() Options**

To see the complete list of options for the Directory() application, type **core show application Directory** at the Asterisk CLI.

Let's add a dial-by-name directory to our dialplan. Simply add this line to your **[docs:users]** context in **extensions.conf**:

```
exten => 6501,1,Directory(vm-demo,users,ef)
```

Now you should be able to dial extension **6501** to test your dial-by-name directory.

## Early Media and the Progress Application

Many dialplan applications within Asterisk support a common VOIP feature known as early media. Early Media is most frequently associated with the SIP channel, but it is also a feature of other channel drivers such as H323. In simple situations, any call in Asterisk that is going to involve audio should invoke either `Progress()` or `Answer()`.

By making use of the progress application, a phone call can be made to play audio before answering a call or even without ever even intending to answer the full call.

Simple Example involving playback:

```
exten => 500,1,Progress()  
exten => 500,n,Wait(1)  
exten => 500,n,Playback(WeAreClosedGoAway,noanswer)  
exten => 500,n,Hangup()
```

In the example above, we start an early media call which waits for a second and then plays a rather rudely named message indicating that the requested service has closed for whatever reason before hanging up. It is worth observing that the Playback application is called with the 'noanswer' argument. Without that argument, Playback would automatically answer the call and then we would no longer be in early media mode.

Strictly speaking, Asterisk will send audio via RTP to any device that calls in regardless of whether Asterisk ever answers or progresses the call. It is possible to make early media calls to some devices without ever sending the progress message, however this is improper and can lead to a myriad of nasty issues that vary from device to device. For instance, in internal testing, there was a problem reported against the Queue application involving the following extension:

```
exten => 500,1,Queue(queueName)
```

This is certainly a brief example. The queue application does not perform any sort of automatic answering, so at this point Asterisk will be sending the phone audio packets, but it will not have formally answered the call or have sent a progress indication. At this point, different phones will behave differently. In the case of the internal test, our Polycom Soundpoint IP 330 phone played nothing while our SNOM360 phone played audio until approximately one minute into the call before it started ceaselessly generating a ring-back indication. There is nothing wrong with either of these phones... they are simply reacting to an oddly formed SIP dialog. Obviously though, neither of these is ideal for a queue and the problem wouldn't have existed had Queue been started after using the Progress application like below:

```
exten => 500,1,Progress()  
exten => 500,n,Queue(queueName)
```

Getting the hang of when to use Progress and Answer can be a little tricky, and it varies greatly from application to application. If you want to be safe, you can always just answer the calls and keep things simple, but there are a number of use cases where it is more appropriate to use early media, and most people who actually need this feature will probably be aware of when it is necessary.

Applications which can use early media and do not automatically answer (incomplete list, please contribute):

- SayAlpha/SayDigits/SayNumber/etc
- Playback (conditionally)
- MP3
- MixMonitor
- MorseCode
- Echo
- Queue
- MusicOnHold

# External IVR Interface

## Asterisk External IVR Interface

If you load `app_externalivr.so` in your Asterisk instance, you will have an `ExternalIVR` application available in your dialplan. This application implements a simple protocol for bidirectional communication with an external process, while simultaneously playing audio files to the connected channel (without interruption or blocking).

There are two ways to use `ExternalIVR`; you can execute an application on the local system or you can establish a socket connection to a TCP/IP socket server.

To execute a local application use the form:

```
ExternalIVR(/full/path/to/application[(arguments)],options)
```

The arguments are optional, however if they exist they must be enclosed in parentheses. The external application will be executed in a child process, with its standard file handles connected to the Asterisk process as follows:

- `stdin (0)` - Events will be received on this handle
- `stdout (1)` - Commands can be sent on this handle
- `stderr (2)` - Messages can be sent on this handle



Use of `stderr` for message communication is discouraged because it is not supported by a socket connection.

To create a socket connection use the form:

```
ExternalIVR(ivr://host[:port][(arguments)],options)
```

The host can be a fully qualified domain name or an IP address (both IPv4 and IPv6 are supported). The port is optional and, if not specified, is 2949 by default. The `ExternalIVR` application will connect to the specified socket server and establish a bidirectional socket connection, where events will be sent to the TCP/IP server and commands received from it.

The specific `ExternalIVR` options, `#events` and `#commands` are detailed below.

Upon execution, if not specifically prevented by an option, the `ExternalIVR` application will answer the channel (if it's not already answered), create an audio generator, and start playing silence. When your application wants to send audio to the channel, it can send a `command` to add a file to the generator's playlist. The generator will then work its way through the list, playing each file in turn until it either runs out of files to play, the channel is hung up, or a command is received to clear the list and start with a new file. At any time, more files can be added to the list and the generator will play them in sequence.

While the generator is playing audio (or silence), any DTMF `#events` received on the channel will be sent to the child process. Note that this can happen at any time, since the generator, the child process and the channel thread are all executing independently. It is very important that your external application be ready to receive events from Asterisk at all times (without blocking), or you could cause the channel to become non-responsive.

If the child process dies, or the remote server disconnects, `ExternalIVR` will notice this and hang up the channel immediately (and also send a message to the log).

### ExternalIVR Options

- `n` - 'n'oanswer, don't answer an otherwise unanswered channel.
- `i` - 'i'gnore\_hangup, instead of sending an `H` event and exiting `ExternalIVR` upon channel hangup, it instead sends an `I` event and expects the external application to exit the process.
- `d` - 'd'ead, allows the operation of `ExternalIVR` on channels that have already been hung up.

### Events

All events are be newline-terminated strings and are sent in the following format:

```
tag,timestamp[,data]
```

The tag can be one of the following characters:

- `0-9` - DTMF event for keys 0 through 9
- `A-D` - DTMF event for keys A through D
- `*` - DTMF event for key \*

- # - DTMF event for key #
- H - The channel was hung up by the connected party
- E - The script requested an exit
- Z - The previous command was unable to be executed. There may be a data element if appropriate, see specific commands below for details
- T - The play list was interrupted (see [S command](#))
- D - A file was dropped from the play list due to interruption (the data element will be the dropped file name) NOTE: this tag conflicts with the [D](#) DTMF event tag. The existence of the data element is used to differentiate between the two cases
- F - A file has finished playing (the data element will be the file name)
- P - A response to the [P command](#)
- G - A response to the [G command](#)
- I - A Inform message, meant to "inform" the client that something has occurred. (see Inform Messages below)

The timestamp will be a decimal representation of the standard Unix epoch-based timestamp, e.g., 284654100.

## Commands

All commands are newline-terminated (`\n`) strings.

The child process can send one of the following commands:

- `S, filename`
- `A, filename`
- `I, TIMESTAMP`
- `H, message`
- `E, message`
- `O, option`
- `V, name=value [ , name=value [ , name=value ] ]`
- `G, name [ , name [ , name ] ]`
- `L, log_message`
- `P, TIMESTAMP`
- `T, TIMESTAMP`

The `S` command checks to see if there is a playable audio file with the specified name, and if so, clears the generator's playlist and places the file onto the list. Note that the playability check does not take into account transcoding requirements, so it is possible for the file to not be played even though it was found. If the file does not exist it sends a `Z` response with the data element set to the file requested. If the generator is not currently playing silence, then `T` and `D` events will be sent to signal the playlist interruption and notify it of the files that will not be played.

The `A` command checks to see if there is a playable audio file with the specified name, and if so, appends it to the generator's playlist. The same playability and exception rules apply as for the `S` command.

The `I` command stops any audio currently playing and clears the generator's playlist. The `I` command was added in Asterisk 11.

The `E` command logs the supplied message to the Asterisk log, stops the generator and terminates the `ExternalIVR` application, but continues execution in the dialplan.

The `H` command logs the supplied message to the Asterisk log, stops the generator, hangs up the channel and terminates the `ExternalIVR` application.

The `O` command allows the child to set/clear options in the `ExternalIVR()` application. The supported options are:

- `(no)autoclear` - Automatically interrupt and clear the playlist upon reception of DTMF input.

The `T` command will answer an unanswered channel. If it fails either answering the channel or starting the generator it sends a `Z` response of `Z, TIMESTAMP`, `P, ANSWER_FAILED` or `Z, TIMESTAMP, GENERATOR_FAILED` respectively.

The `V` command sets the specified channel variable(s) to the specified value(s).

The `G` command gets the specified channel variable(s). Multiple variables are separated by commas. Response is in `name=value` format.

The `P` command gets the parameters passed into `ExternalIVR` minus the options to `ExternalIVR` itself:


If `ExternalIVR` is executed as:

```
ExternalIVR(/usr/bin/foo(arg1, arg2), n)
```


The response to the `P` command would be:

```
P, TIMESTAMP, /usr/bin/foo, arg1, arg2
```



 This is the only way for a TCP/IP server to be able to get retrieve the arguments.

The `L` command puts a message into the Asterisk log.

 This is preferred to using `stderr` and is the only way for a TCP/IP server to log a message.

### **Inform Messages**

The only inform message that currently exists is a `HANGUP` message, in the form `I, TIMESTAMP, HANGUP` and is used to inform of a hangup when the `i` option is specified.

### **Errors**

Any newline-terminated (`\n`) output generated by the child process on its `stderr` handle will be copied into the Asterisk log.

# MacroExclusive

## About the MacroExclusive application

By: Steve Davies <steve@connection-telecom.com>

The MacroExclusive application was added to solve the problem of synchronization between calls running at the same time.

This is usually an issue when you have calls manipulating global variables or the Asterisk database, but may be useful elsewhere.

Consider this example macro, intended to return a "next" number - each caller is intended to get a different number:

```
[macro-next]
exten => s,1,Set(RESET={COUNT})
exten => s,n,SetGlobalVar(COUNT=${${COUNT} + 1})
```

The problem is that in a box with high activity, you can be sure that two calls will come along together - both will get the same "RESULT", or the "COUNT" value will get mangled.

Calling this Macro via MacroExclusive will use a mutex to make sure that only one call executes in the Macro at a time. This ensures that the two lines execute as a unit.

Note that even the s,2 line above has its own race problem. Two calls running that line at once will step on each other and the count will end up as +1 rather than +2.

I've also been able to use MacroExclusive where I have two Macros that need to be mutually exclusive.

Here's the example:

```
[macro-push]
; push value ${ARG2} onto stack ${ARG1}
exten => s,1,Set(DB(STACK/${ARG1})=${ARG2}^${DB(STACK/${ARG1})})

[macro-pop]
; pop top value from stack ${ARG1}
exten => s,1,Set(RESULT=${DB(STACK/${ARG1})})
exten => s,n,Set(DB(STACK/${ARG1})=${CUT(RESULT,^,2)})
exten => s,n,Set(RESULT=${CUT(RESULT,^,1)})
```

All that futzing with the STACK/\${ARG1} in the astdb needs protecting if this is to work. But neither push nor pop can run together.

So add this "pattern":

```
[macro-stack]
exten => Macro(${ARG1},${ARG2},${ARG3})
```

... and use it like so:

```
exten => s,1,MacroExclusive(stack,push,MYSTACK,bananas)
exten => s,n,MacroExclusive(stack,push,MYSTACK,apples)
exten => s,n,MacroExclusive(stack,push,MYSTACK,guavas)
exten => s,n,MacroExclusive(stack,push,MYSTACK,pawpaws)
exten => s,n,MacroExclusive(stack,pop,MYSTACK) ; RESULT gets pawpaws (yum)
exten => s,n,MacroExclusive(stack,pop,MYSTACK) ; RESULT gets guavas
exten => s,n,MacroExclusive(stack,pop,MYSTACK) ; RESULT gets apples
exten => s,n,MacroExclusive(stack,pop,MYSTACK) ; RESULT gets bananas
```

We get to the push and pop macros "via" the stack macro. But only one call can execute the stack macro at a time; ergo, only one of push OR pop can run at a time.

Hope people find this useful.

Lastly, its worth pointing out that only Macros that access shared data will require this MacroExclusive protection. And Macro's that you call with macroExclusive should run quickly or you will clog up your Asterisk system.

## The Read Application

The **Read()** application allows you to play a sound prompt to the caller and retrieve DTMF input from the caller, and save that input in a variable. The first parameter to the **Read()** application is the name of the variable to create, and the second is the sound prompt or prompts to play. (If you want multiple prompts, simply concatenate them together with ampersands, just like you would with the **Background()** application.) There are some additional parameters that you can pass to the **Read()** application to control the number of digits, timeouts, and so forth. You can get a complete list by running the core show application read command at the Asterisk CLI. If no timeout is specified, **Read()** will finish when the caller presses the hash key (#) on their keypad.

```
exten=>6123,1,Read(Digits,enter-ext-of-person)
exten=>6123,n,Playback(you-entered)
exten=>6123,n,SayNumber(${Digits})
```

In this example, the **Read()** application plays a sound prompt which says "Please enter the extension of the person you are looking for", and saves the captured digits in a variable called **Digits**. It then plays a sound prompt which says "You entered" and then reads back the value of the **Digits** variable.



## The Verbose and NoOp Applications

Asterisk has a convenient dialplan applications for printing information to the command-line interface, called **Verbose()**. The **Verbose()** application takes two parameters: the first parameter is the minimum verbosity level at which to print the message, and the second parameter is the message to print. This extension would print the current channel identifier and unique identifier for the current call, if the verbosity level is two or higher.

```
exten=>6123,1,Verbose(2,The channel name is ${CHANNEL})
exten=>6123,n,Verbose(2,The unique id is ${UNIQUEID})
```

The **NoOp()** application stands for "No Operation". In other words, it does nothing. Because of the way Asterisk prints everything to the console if your verbosity level is three or higher, however, the **NoOp()** application is often used to print debugging information to the console like the **Verbose()** does. While you'll probably come across examples of the **NoOp()** application in other examples, we recommend you use the **Verbose()** application instead.

# SMS

## The SMS application

SMS() is an application to handles calls to/from text message capable phones and message centres using ETSI ES 201 912 protocol 1 FSK messaging over analog calls.

Basically it allows sending and receiving of text messages over the PSTN. It is compatible with BT Text service in the UK and works on ISDN and PSTN lines. It is designed to connect to an ISDN or DAHDI interface directly and uses FSK so would probably not work over any sort of compressed link (like a VoIP call using GSM codec).

Typical applications include:-

1. Connection to a message centre to send text messages - probably initiated via the manager interface or "outgoing" directory
2. Connection to a POTS line with an SMS capable phone to send messages - probably initiated via the manager interface or "outgoing" directory
3. Acceptance of calls from the message centre (based on CLI) and storage of received messages
4. Acceptance of calls from a POTS line with an SMS capable phone and storage of received messages

### Arguments to sms():

- First argument is queue name
- Second is options:
  - a: SMS() is to act as the answering side, and so send the initial FSK frame
  - s: SMS() is to act as a service centre side rather than as terminal equipment
- If a third argument is specified, then SMS does not handle the call at all, but takes the third argument as a destination number to send an SMS to
- The forth argument onward is a message to be queued to the number in the third argument. All this does is create the file in the me-sc directory.
- If 's' is set then the number is the source address and the message placed in the sc-me directory.

All text messages are stored in /var/spool/asterisk/sms

A log is recorded in /var/log/asterisk/sms

There are two subdirectories called sc-me.<queue name> holding all messages from service centre to phone, and me-sc.<queue name> holding all messages from phone to service centre.

In each directory are messages in files, one per file, using any filename not starting with a dot.

When connected as a service centre, SMS(s) will send all messages waiting in the sc-me-<queue name> directory, deleting the files as it goes. Any received in this mode are placed in the me-sc-<queue name> directory.

When connected as a client, SMS() will send all messages waiting in the me-sc-<queue name> directory, deleting the files as it goes. Any received in this mode are placed in the sc-me-<queue name> directory.

Message files created by SMS() use a time stamp/reference based filename.

The format of the sms file is lines that have the form of key=value

Keys are :

- oa - Originating Address. Telephone number, national number if just digits. Telephone number starting with + then digits for international. Ignored on sending messages to service centre (CLI used)
- da - Destination Address. Telephone number, national number if just digits. Telephone number starting with + then digits for international.
- scts - Service Centre Time Stamp in the format YYYY-MM-DD HH:MM:SS
- pid - Protocol Identifier (decimal octet value)
- dcs - Data coding scheme (decimal octet value)
- mr - Message reference (decimal octet value)
- ud - The message (see escaping below)
- srr - 0/1 Status Report Request
- rp - 0/1 Return Path
- vp - mins validity period

Omitted fields have default values.

Note that there is special format for ud, ud# instead of ud= which is followed by raw hex (2 characters per octet). This is used in output where characters other than 10,13,32-126,128-255 are included in the data. In this case a comment (line starting ;) is added showing the printable characters

When generating files to send to a service centre, only da and ud need be specified. oa is ignored.

When generating files to send to a phone, only oa and ud need be specified. da is ignored.

When receiving a message as a service centre, only the destination address is sent, so the originating address is set to the callerid.

## EXAMPLES

The following are examples of use within the UK using BT Text SMS/landline service.

This is a context to use with a manager script.

```
[smsdial]
; create and send a text message, expects number+message and
; connect to 17094009
exten => _X.,1,SMS(${CALLERIDNUM},,${EXTEN},${CALLERIDNAME})
exten => _X.,n,SMS(${CALLERIDNUM})
exten => _X.,n,Hangup
```

The script sends

```
action: originate
callerid: message <from>
exten: to
channel: Local/17094009
context: smsdial
priority: 1
```

You put the message as the name of the caller ID (messy, I know), the originating number and hence queue name as the number of the caller ID and the exten as the number to which the sms is to be sent. The context uses SMS to create the message in the queue and then SMS to communicate with 17094009 to actually send the message.

Note that the 9 on the end of 17094009 is the sub address 9 meaning no sub address (BT specific). If a different digit is used then that is the sub address for the sending message source address (appended to the outgoing CLI by BT).

For incoming calls you can use a context like this :-

```
[incoming]
exten => _XXXXXX/_8005875290,1,SMS(${EXTEN:3},a)
exten => _XXXXXX/_8005875290,n,System(/usr/lib/asterisk/smsin ${EXTEN:3})
exten => _XXXXXX/_80058752[0-8]0,1,SMS(${EXTEN:3}${CALLERIDNUM:8:1},a)
exten => _XXXXXX/_80058752[0-8]0,n,System(/usr/lib/asterisk/smsin
${EXTEN>:3}${CALLERIDNUM:8:1})
exten => _XXXXXX/_80058752[0-8]0,n,Hangup
```

In this case the called number we get from BT is 6 digits (XXXXXX) and we are using the last 3 digits as the queue name.

Priority 1 causes the SMS to be received and processed for the incoming call. It is from 080058752X0. The two versions handle the queue name as 3 digits (no sub address) or 4 digits (with sub address). In both cases, after the call a script (smsin) is run - this is optional, but is useful to actually process the received queued SMS. In our case we email them based on the target number. Priority 3 hangs up.

If using the CAPI drivers they send the right CLI and so the \_800... would be \_0800...

# Voicemail

The Asterisk voicemail module provides two key applications for dealing with voice mail.

The **VoiceMail()** application takes two parameters:

1. **Mailbox**

- This parameter specifies the mailbox in which the voice mail message should be left. It should be a mailbox number and a voice mail context concatenated with an at-sign (@), like **6001@default**. (Voice mail boxes are divided out into various voice mail context, similar to the way that extensions are broken up into dialplan contexts.) If the voice mail context is omitted, it will default to the **default** voice mail context.

2. **Options**

- One or more options for controlling the mailbox greetings. The most popular options include the **u** option to play the unavailable message, the **b** option to play the busy message, and the **s** option to skip the system-generated instructions.

The **VoiceMailMain()** application allows the owner of a voice mail box to retrieve their messages, as well as set mailbox options such as greetings and their PIN number. The **VoiceMailMain()** application takes two parameters:

1. **Mailbox** - This parameter specifies the mailbox to log into. It should be a mailbox number and a voice mail context, concatenated with an at-sign (@), like **6001@default**. If the voice mail context is omitted, it will default to the default voice mail context. If the mailbox number is omitted, the system will prompt the caller for the mailbox number.
2. **Options** - One or more options for controlling the voicemail system. The most popular option is the **s** option, which skips asking for the PIN number



### Direct Access to Voicemail

Please exercise extreme caution when using the **s** option! With this option set, anyone which has access to this extension can retrieve voicemail messages without entering the mailbox passcode.

## ODBC Voicemail Storage

ODBC Storage allows you to store voicemail messages within a database instead of using a file. This is not a full realtime engine and only supports ODBC. The table description for the voicemessages table is as follows:

Field	Type	Null	Key	Default	Extra
msgnum	int(11)	Yes		NULL	
dir	varchar(80)	Yes	MUL		NULL
context	varchar(80)	Yes		NULL	
macrocontext	varchar(80)	Yes		NULL	
callerid	varchar(40)	Yes		NULL	
origtime	varchar(40)	Yes		NULL	
duration	varchar(20)	Yes		NULL	
flag	varchar(8)	Yes		NULL	
mailboxuser	varchar(80)	Yes		NULL	
mailboxcontext	varchar(80)	Yes		NULL	
recording	longblob	Yes		NULL	
msg_id	varchar(40)	Yes		NULL	(See Note)



### Upgrade Notice

The msg\_id column is new in Asterisk 11. Existing installations should add this column to their schema when upgrading to Asterisk 11. Existing voicemail messages will have this value populated when the messages are initially manipulated by app\_voicemail in Asterisk 11.

The database name (from /etc/asterisk/res\_odbc.conf) is in the odbctest variable in the general section of voicemail.conf.

You may modify the voicemessages table name by using odbctest=table\_name in voicemail.conf.

## IMAP Voicemail Storage

By enabling IMAP Storage, Asterisk will use native IMAP as the storage mechanism for voicemail messages instead of using the standard file structure.

Tighter integration of Asterisk voicemail and IMAP email services allows additional voicemail functionality, including:

- Listening to a voicemail on the phone will set its state to "read" in a user's mailbox automatically.
- Deleting a voicemail on the phone will delete it from the user's mailbox automatically.
- Accessing a voicemail recording email message will turn off the message waiting indicator (MWI) on the user's phone.
- Deleting a voicemail recording email will also turn off the message waiting indicator, and delete the message from the voicemail system.

## IMAP VM Storage Installation Notes

### University of Washington IMAP C-Client

If you do not have the University of Washington's IMAP c-client installed on your system, you will need to download the c-client source distribution (<http://www.washington.edu/imap/>) and compile it. Asterisk supports the 2007 version of c-client as there appears to be issues with older versions which cause Asterisk to crash in certain scenarios. It is highly recommended that you utilize a current version of the c-client libraries. Additionally, `mail_expunge_full` is enabled in the 2006 and later versions.

Note that Asterisk only uses the 'c-client' portion of the UW IMAP toolkit, but building it also builds an IMAP server and various other utilities. Because of this, the build instructions for the IMAP toolkit are somewhat complicated and can lead to confusion about what is needed.

If you are going to be connecting Asterisk to an existing IMAP server, then you don't need to care about the server or utilities in the IMAP toolkit at all. If you want to also install the UW IMAPD server, that is outside the scope of this document.

Building the c-client library is fairly straightforward; for example, on a Debian system there are two possibilities:

If you will not be using SSL to connect to the IMAP server:

```
$ make slx SSLTYPE=none
```

If you will be using SSL to connect to the IMAP server:

```
$ make slx EXTRACFLAGS="-I/usr/include/openssl"
```

Additionally, you may wish to build on a 64-bit machine, in which case you need to add `-fPIC` to `EXTRACFLAGS`. So, building on a 64-bit machine with SSL support would look something like:

```
$ make slx EXTRACFLAGS="-fPIC -I/usr/include/openssl"
```

Or without SSL support:

```
$ make slx SSLTYPE=none EXTRACFLAGS=-fPIC
```

Once this completes you can proceed with the Asterisk build; there is no need to run 'make install'.

### Compiling Asterisk

Configure with `./configure -with-imap=/usr/src/imap` or wherever you built the UWashington IMAP Toolkit. This directory will be searched for a source installation. If no source installation is found there, then a package installation of the IMAP c-client will be searched for in this directory. If one is not found, then configure will fail.

A second configure option is to not specify a directory (i.e. `./configure -with-imap`). This will assume that you have the `imap-2007e` source installed in the `./imap` directory relative to the Asterisk source. If you do not have this source, then configure will default to the "system" option defined in the next paragraph.

A third option is `./configure -with-imap=system`. This will assume that you have installed a dynamically linked version of the c-client library (most likely via a package provided by your distro). This will attempt to link against `-lc-client` and will search for c-client headers in your include path starting with the `imap` directory, and upon failure, in the c-client directory.

When you run 'make menuselect', choose 'Voicemail Build Options' and the `IMAP_STORAGE` option should be available for selection.

After selecting the `IMAP_STORAGE` option, use the 'x' key to exit menuselect and save your changes, and the build/install Asterisk normally.

## IMAP VM Storage Voicemail.conf Modifications

The following directives have been added to voicemail.conf:

- `imapserver=<name or IP address of IMAP mail server>`
- `imapport=<IMAP port, defaults to 143>`
- `imapflags=<IMAP flags, "novalidate-cert" for example>`
- `imapfolder=<IMAP folder to store messages to>`
- `imapgreetings=<yes or no>`
- `greetingsfolder=<IMAP folder to store greetings in if imapgreetings is enabled>`
- `expungeonhangup=<yes or no>`
- `authuser=<username>`
- `authpassword=<password>`
- `opentimeout=<TCP open timeout in seconds>`
- `closetimeout=<TCP close timeout in seconds>`
- `readtimeout=<TCP read timeout in seconds>`
- `writetimeout=<TCP write timeout in seconds>`

The "imapfolder" can be used to specify an alternative folder on your IMAP server to store voicemails in. If not specified, the default folder 'INBOX' will be used.

The "imapgreetings" parameter can be enabled in order to store voicemail greetings on the IMAP server. If disabled, then they will be stored on the local file system as normal.

The "greetingsfolder" can be set to store greetings on the IMAP server when "imapgreetings" is enabled in an alternative folder than that set by "imapfolder" or the default folder for voicemails.

The "expungeonhangup" flag is used to determine if the voicemail system should expunge all messages marked for deletion when the user hangs up the phone.

Each mailbox definition should also have `imapuser=imap username`. For example:

```
4123=>4123,James Rothenberger, jar@onebiztone.com,,attach=yes|imapuser=jar
```

The directives "authuser" and "authpassword" are not needed when using Kerberos. They are defined to allow Asterisk to authenticate as a single user that has access to all mailboxes as an alternative to Kerberos.



## **Voicemail and IMAP Folders**

Besides INBOX, users should create "Old", "Work", "Family" and "Friends" IMAP folders at the same level of hierarchy as the INBOX. These will be used as alternate folders for storing voicemail messages to mimic the behavior of the current (file-based) voicemail system.

Please note that it is not recommended to store your voicemails in the top level folder where your users will keep their emails, especially if there are a large number of emails. A large number of emails in the same folder(s) that you're storing your voicemails could cause a large delay as Asterisk must parse through all the emails. For example a mailbox with 100 emails in it could take up to 60 seconds to receive a response.

## Separate vs. Shared E-mail Accounts

As administrator you will have to decide if you want to send the voicemail messages to a separate IMAP account or use each user's existing IMAP mailbox for voicemail storage. The IMAP storage mechanism will work either way.

By implementing a single IMAP mailbox, the user will see voicemail messages appear in the same INBOX as other messages. The disadvantage of this method is that if the IMAP server does NOT support UIDPLUS, Asterisk voicemail will expunge ALL messages marked for deletion when the user exits the voicemail system, not just the VOICEMAIL messages marked for deletion.

By implementing separate IMAP mailboxes for voicemail and email, voicemail expunges will not remove regular email flagged for deletion.

## IMAP Server Implementations

There are various IMAP server implementations, each supports a potentially different set of features.  
UW IMAP-2005 or earlier

UIDPLUS is currently NOT supported on these versions of UW-IMAP. Please note that without UID\_EXPUNGE, Asterisk voicemail will expunge ALL messages marked for deletion when a user exits the voicemail system (hangs up the phone).

This version is **not recommended for Asterisk**.  
UW IMAP-2006

This version supports UIDPLUS, which allows UID\_EXPUNGE capabilities. This feature allow the system to expunge ONLY pertinent messages, instead of the default behavior, which is to expunge ALL messages marked for deletion when EXPUNGE is called. The IMAP storage mechanism is this version of Asterisk will check if the UID\_EXPUNGE feature is supported by the server, and use it if possible.

This version is **not recommended for Asterisk**.  
UW IMAP-2007

This is the currently recommended version for use with Asterisk.  
Cyrus IMAP

Cyrus IMAP server v2.3.3 has been tested using a hierarchy delimiter of '/'.

## IMAP Voicemail Quota Support

If the IMAP server supports quotas, Asterisk will check the quota when accessing voicemail. Currently only a warning is given to the user that their quota is exceeded.

## IMAP Voicemail Application Notes

Since the primary storage mechanism is IMAP, all message information that was previously stored in an associated text file, AND the recording itself, is now stored in a single email message. This means that the .gsm recording will ALWAYS be attached to the message (along with the user's preference of recording format if different - ie. .WAV). The voicemail message information is stored in the email message headers. These headers include:

- X-Asterisk-VM-Message-Num
- X-Asterisk-VM-Server-Name
- X-Asterisk-VM-Context
- X-Asterisk-VM-Extension
- X-Asterisk-VM-Priority
- X-Asterisk-VM-Caller-channel
- X-Asterisk-VM-Caller-ID-Num
- X-Asterisk-VM-Caller-ID-Name
- X-Asterisk-VM-Duration
- X-Asterisk-VM-Category
- X-Asterisk-VM-Orig-date
- X-Asterisk-VM-Orig-time
- X-Asterisk-VM-Message-ID



### Upgrade Notice

The X-Asterisk-VM-Message-ID header is new in Asterisk 11. Existing voicemail messages from older versions of Asterisk will have this header added to the message when the messages are manipulated by app\_voicemail in Asterisk 11.

# Message Waiting Indication

## What is MWI?

This page explains the resources available for Message Waiting Indicator(or Indication) functionality in Asterisk and how to configure.

### Documentation Under Construction

## Configuring MWI

Here we talk about configuring Asterisk to provide MWI to endpoints or other systems.

### Providing MWI to chan\_pjsip endpoints

Providing MWI to a chan\_pjsip endpoint requires configuring the "**mailboxes**" option in either the **endpoint** type config section, or the **aor** section.

See the descriptions linked below which explain when to use the option in each section.

[Description of "mailboxes" option for the Endpoint section](#)

[Description of "mailboxes" option for the AOR section.](#)

## Configuring External MWI

Let's look at configuring Asterisk to receive MWI from other systems.

Depending on your Asterisk version and configuration, there are a few different ways to configure receiving MWI from external sources.

1. **chan\_sip**: outbound MWI subscriptions and receiving unsolicited MWI NOTIFY messages
2. **res\_external\_mwi**: A module providing an API for other systems to communicate MWI state to Asterisk



**res\_pjsip**: The functionality for outbound SIP subscription or handling of unsolicited NOTIFYs inbound to Asterisk is not available in res\_pjsip yet. Internal infrastructure is built that would allow it, so if this is something you want to work on, please contact the [Asterisk development community](#).

### Outbound MWI subscription with chan\_sip

Asterisk can subscribe to receive MWI from another SIP server and store it locally for retrieval by other phones. At this time, you can only subscribe using UDP as the transport. Format for the MWI register statement is:

```
[general]
;mwi => user[:secret[:authuser]]@host[:port]/mailbox
;
; Examples:
;mwi => 1234:password@mysipprovider.example.com/1234
;mwi => 1234:password@myportprovider.example.com:6969/1234
;mwi => 1234:password:authuser@myauthprovider.example.com/1234
;mwi => 1234:password:authuser@myauthportprovider.example.com:6969/1234
```

MWI received will be stored in the 1234 mailbox of the SIP\_Remote context. It can be used by other phones by setting their SIP peers "mailbox" option to the <mailbox\_number>@SIP\_Remote. e.g. mailbox=1234@SIP\_Remote

### Reception of unsolicited MWI NOTIFY with chan\_sip

A chan\_sip peer can be configured to receive unsolicited MWI NOTIFY messages and associate them with a particular mailbox.

```
[somesippeer]
;unsolicited_mailbox=123456789
```

If the remote SIP server sends an unsolicited MWI NOTIFY message the new/old message count will be stored in the configured virtual mailbox. It can be used by any device supporting MWI by specifying mailbox=<configured value>@SIP\_Remote as the mailbox for the desired SIP peer.

### res\_external\_mwi

External sources can use the API provided by `res_external_mwi` to communicate MWI and mailbox state.

### **Documentation Under Construction**

[Asterisk 12 Configuration\\_res\\_mwi\\_external](#)



`res_external_mwi.so` is mutually exclusive with `app_voicemail.so`. You'll have to load only the one you want to use.

# Conferencing Applications

## Conferencing with Asterisk

Up until about Asterisk 1.6; app\_meetme was the main application providing conferencing style features. In Asterisk 1.6.2 the ConfBridge module was added and then rewritten in Asterisk 10.

Both MeetMe and ConfBridge still exist in the latest Asterisk versions and provide different feature sets, but with plenty of overlap. Development attention is primarily given to ConfBridge these days and it is the recommended option for modern deployments when using a pre-built application.

There is a detailed [description of ConfBridge functionality](#) on the wiki as well as [MeetMe application](#) usage notes.

## Building your own conferencing application

Conferencing needs can be very specific to your business application. The conferencing applications included with Asterisk provide basic features that will work for many users.

If the included applications don't work for you then you might consider building your own application using the [Asterisk REST Interface](#). This will give you access to all the communication primitives needed and then you can write the logic you need in a language you are comfortable with.



# ConfBridge

## Overview

Asterisk, since its early days, has offered a conferencing application called MeetMe ([app\\_meetme.so](http://app.meetme.so)). MeetMe provides DAHDI-mixed software-based bridges for multi-party audio conferencing. MeetMe is used by nearly all Asterisk implementations - small office, call center, large office, feature-server, third-party application, etc. It has been extremely successful as an audio bridge.

Over time, several significant limitations of MeetMe have been encountered by its users. Among these are two of distinction: MeetMe requires DAHDI for mixing, and is thus limited to 8kHz (PSTN) audio sampling rates; and MeetMe is delivered in a fairly static form, it does not provide extensive configuration options.

To address these limitations, a new conferencing application, based upon the ConfBridge application introduced in Asterisk 1.6.0, is now available with Asterisk 10. This new ConfBridge application replaces the older ConfBridge application. It is not intended to be a direct replacement for MeetMe, it will not provide feature parity with the MeetMe application. Instead, the new ConfBridge application delivers a completely redesigned set of functionality that most users will find more than sufficient, and in many ways better, for their conferencing needs.

## ConfBridge Concepts

ConfBridge provides four internal concepts:

1. Conference Number
2. Bridge Profile
3. User Profile
4. Conference Menu

A **Conference Number** is a numerical representation for an instance of the bridge. Callers joined to the same conference number will be in the same conference bridge; they're connected. Callers joined to different conference numbers are not in the same conference bridge; they're separated. Conference Numbers are assigned in the dialplan. Unlike MeetMe, they're not pre-reserved.

A **Bridge Profile** is a named set of options that control the behavior of a particular conference bridge. Each bridge must have its own profile. A single bridge cannot have more than one Bridge Profile.

A **User Profile** is a named set of options that control the user's experience as a member of a particular bridge. Each user participating in a bridge can have their own individual User Profile.

A **Conference Menu** is a named set of options that are provided to a user when they present DTMF keys while connected to the bridge. Each user participating in a bridge can have their own individual Conference Menu.

## ConfBridge Application Syntax

The ConfBridge application syntax and usage can be found at [Asterisk 13 Application\\_ConfBridge](#)

## ConfBridge Application Examples

### Example 1

In this example, callers will be joined to conference number 1234, using the default Bridge Profile, the default User Profile, and no Conference Menu.

```
exten => 1,1,Answer()  
exten => 1,n,ConfBridge(1234)
```

### Example 2

In this example, callers will be joined to conference number 1234, with the default Bridge Profile, a User Profile called "1234\_participants" and a Conference Menu called "1234\_menu."

```
exten => 1,1,Answer()  
exten => 1,n,ConfBridge(1234,,1234_participants,1234_menu)
```

## Usage Notes, FAQ and Other

There are many points to consider when using the new ConfBridge application. Some will be examined here.

### Video Conferencing

### On This Page

- Overview
- ConfBridge Concepts
- ConfBridge Application Syntax
- ConfBridge Application Examples
- Usage Notes, FAQ and Other

### In This Section

- ConfBridge AMI Actions
- ConfBridge AMI Events
- ConfBridge CLI Commands
- ConfBridge Configuration
- ConfBridge Functions
- ConfBridge Sound Prompts

It is imperative that a video conference not have participants using disparate video codecs or encoding profiles. Everyone **must** use the same codec and profile. Otherwise, the video sessions won't work - you'll likely experience frozen video as the conference switches from one video stream using a codec your client has negotiated, to a video stream using a codec your client hasn't negotiated or doesn't support.

### Video Endpoints

ConfBridge has been tested against a number of video-capable SIP endpoints. Success, and your mileage will vary.

Endpoints that work:

- Jitsi - Jitsi works well for both H.264 and H.263+1998 video calling on Mac, Linux and Windows machines. Currently, Jitsi seems to be the best-working, free, H.264-capable SIP video client.
- Linphone - Linphone works well for H.263+1998 and H.263 video calling on Linux - the Mac port and mobile ports do not support video. Currently, Linphone seems to be the best-working, free, H.263-capable SIP video client, when Jitsi or H.263+1998 aren't an option.
- Empathy - Empathy works for H.264 calling, but is amazingly difficult to configure (why one has to make two SIP accounts just to make a SIP call is a mystery).
- Lifesize - The Lifesize client supports H.264 and runs on Windows only. It works very well, but it isn't free.
- Polycom VVX 1500 - The Polycom VVX 1500 works well for H.264 calling. If you're connecting it to Jitsi, you may have to configure Jitsi to use the Baseline H.264 profile instead of the Main profile.

Endpoints that don't or weren't tested:

- Xlite - Xlite works in some cases, but also seems to crash, regardless of operating system, at odd times. In other cases, Xlite isn't able to decode video from clients.
- Ekiga - Ekiga wasn't tested, because our test camera wasn't supported by the client. The same camera was supported by other soft clients.
- SIPDroid - SIPDroid doesn't seem to work.
- OfficeSIP Messenger - OfficeSIP Messenger didn't seem capable of performing a SIP registration. On this basis alone, no one should recommend its use.

### Mixing Interval

The mixing interval for a conference is defined in its Bridge Profile. The allowable options are 10, 20, 40, and 80, all in milliseconds. Usage of 80ms mixing intervals is only supported for conferences that are sampled at 8, 12, 16, 24, 32, and 48kHz. Usage of 40ms intervals includes all of the aforementioned sampling rates as well as 96kHz. 192kHz sampled conferences are only supported at 10 and 20ms mixing intervals. These limitations are imposed because higher mixing intervals at the higher sampling rates causes large increases in memory consumption. Adventurous users may, through changing of the MAX\_DATALEN define in bridge\_softmix.c allow 96kHz and 192kHz sampled conferences to operate at longer intervals - set to 16192 for 96kHz at 80ms or 32384 for 192kHz at 80ms, recompile, and restart.

### Maximizing Performance

In order to maximize the performance of a given machine for ConfBridge purposes, there are several steps one should take.

- Enable `dsp_drop_silence` is enabled in the User Profile.
  - This is the **single most** important step one can take when trying to increase the number of bridge participants that a single machine can handle. Enabling this means that the audio of users that aren't speaking isn't mixed in with the bridge.
- Lengthen `mixing_interval` in the Bridge Profile.
  - The default interval is 20ms. Other options are 10, 40, and 80ms. Lower values provide a "tighter" sound, but require substantially more CPU. Higher values provide a "looser" sound, and consume substantially less CPU. Setting the value to 80 provides the highest number of possible participants.
- Connect clients at the same sampling rate.
  - Requiring the bridge to resample between clients that use codecs with different sampling rates is an expensive operation. If all clients are dialed in to the bridge at the same sampling rate, and the bridge operates at that same rate, e.g. 16kHz, then the number of possible clients will be maximized.
- Run Asterisk with a higher priority.
  - By default, Asterisk operates at a relatively normal priority, as compared to other processes on the system. To maximize the number of possible clients, Asterisk should be started using the `-p` (realtime) flag. If the load becomes too large, this can negatively impact the performance of other processes, including the console itself - making it difficult to remotely administer a fully loaded system.

As the number of clients approaches the maximum possible on the given machine, given its processing capabilities, audio quality will suffer. Following the above guidelines will increase the number of connected clients before audio quality suffers.

## ConfBridge AMI Actions

### ***ConfbridgeList***

Lists all users in a particular ConfBridge conference. ConfbridgeList will follow as separate events, followed by a final event called ConfbridgeListComplete

#### **Example**

```
Action: ConfbridgeList
Conference: 1111

Response: Success
EventList: start
Message: Confbridge user list will follow

Event: ConfbridgeList
Conference: 1111
CallerIDNum: malcolm
CallerIDName: malcolm
Channel: SIP/malcolm-00000000
Admin: No
MarkedUser: No

Event: ConfbridgeListComplete
EventList: Complete
ListItems: 1
```

### **On This Page**

- [ConfbridgeList](#)
- [ConfbridgeListRooms](#)
- [ConfbridgeMute](#)
- [ConfbridgeUnmute](#)
- [ConfbridgeKick](#)
- [ConfbridgeLock](#)
- [ConfbridgeUnlock](#)
- [ConfbridgeStartRecord](#)
- [ConfbridgeStopRecord](#)
- [ConfbridgeSetSingleVideoSrc](#)

### ***ConfbridgeListRooms***

Lists data about all active conferences. ConfbridgeListRooms will follow as separate events, followed by a final event called ConfbridgeListRoomsComplete.

#### **Example**

```
Action: ConfbridgeListRooms

Response: Success
EventList: start
Message: Confbridge conferences will follow

Event: ConfbridgeListRooms
Conference: 1111
Parties: 1
Marked: 0
Locked: No

Event: ConfbridgeListRoomsComplete
EventList: Complete
ListItems: 1
```

### ***ConfbridgeMute***

Mutes a specified user in a specified conference.

#### **Example**

```
Action: ConfbridgeMute
Conference: 1111
Channel: SIP/mypeer-00000001

Response: Success
Message: User muted
```

### ***ConfbridgeUnmute***

Unmutes a specified user in a specified conference.

#### **Example**

```
Action: ConfbridgeUnmute
Conference: 1111
Channel: SIP/mypeer-00000001

Response: Success
Message: User unmuted
```

## ***ConfbridgeKick***

Removes a specified user from a specified conference.

### **Example**

```
Action: ConfbridgeKick
Conference: 1111
Channel: SIP/mypeer-00000001

Response: Success
Message: User kicked
```

## ***ConfbridgeLock***

Locks a specified conference.

### **Example**

```
Action: ConfbridgeLock
Conference: 1111

Response: Success
Message: Conference locked
```

## ***ConfbridgeUnlock***

Unlocks a specified conference.

### **Example**

```
Action: ConfbridgeUnlock
Conference: 1111

Response: Success
Message: Conference unlocked
```

## ***ConfbridgeStartRecord***

Starts recording a specified conference, with an optional filename. If recording is already in progress, an error will be returned. If RecordFile is not provided, the default record\_file as specified in the conferences Bridge Profile will be used. If record\_file is not specified, a file will automatically be generated in Asterisk's monitor directory.

### **Example**

```
Action: ConfbridgeStartRecord
Conference: 1111

Response: Success
Message: Conference Recording Started.

Event: VarSet
Privilege: dialplan,all
Channel: ConfBridgeRecorder/conf-1111-uid-1653801660
Variable: MIXMONITOR_FILENAME
Value: /var/spool/asterisk/monitor/confbridge-1111-1303309869.wav
Uniqueid: 1303309869.6
```

## ***ConfbridgeStopRecord***

Stops recording a specified conference.

### **Example**

```
Action: ConfbridgeStopRecord
Conference: 1111

Response: Success
Message: Conference Recording Stopped.

Event: Hangup
Privilege: call,all
Channel: ConfBridgeRecorder/conf-1111-uid-1653801660
Uniqueid: 1303309869.6
CallerIDNum: <unknown>
CallerIDName: <unknown>
Cause: 0
Cause-txt: Unknown
```

### ***ConfbridgeSetSingleVideoSrc***

This action sets a conference user as the single video source distributed to all other video-capable participants.

#### **Example**

```
Action: ConfbridgeSetSingleVideoSrc
Conference: 1111
Channel: SIP/mypeer-00000001

Response: Success
Message: Conference single video source set.
```

## ConfBridge AMI Events

### **ConfbridgeStart**

This event is sent when the first user requests a conference and it is instantiated

#### **Example**

```
Event: ConfbridgeStart
Privilege: call,all
Conference: 1111
```

### **On This Page**

- [ConfbridgeStart](#)
- [ConfbridgeJoin](#)
- [ConfbridgeLeave](#)
- [ConfbridgeEnd](#)
- [ConfbridgeTalking](#)

### **ConfbridgeJoin**

This event is sent when a user joins a conference - either one already in progress or as the first user to join a newly instantiated bridge.

#### **Example**

```
Event: ConfbridgeJoin
Privilege: call,all
Channel: SIP/mypeer-00000001
Uniqueid: 1303309562.3
Conference: 1111
CallerIDnum: 1234
CallerIDname: mypeer
```

### **ConfbridgeLeave**

This event is sent when a user leaves a conference.

#### **Example**

```
Event: ConfbridgeLeave
Privilege: call,all
Channel: SIP/mypeer-00000001
Uniqueid: 1303308745.0
Conference: 1111
CallerIDnum: 1234
CallerIDname: mypeer
```

### **ConfbridgeEnd**

This event is sent when the last user leaves a conference and it is torn down.

#### **Example**

```
Event: ConfbridgeEnd
Privilege: call,all
Conference: 1111
```

### **ConfbridgeTalking**

This event is sent when the conference detects that a user has either begin or stopped talking.

#### **Start talking Example**

```
Event: ConfbridgeTalking
Privilege: call, all
Channel: SIP/mypeer-00000001
Uniqueid: 1303308745.0
Conference: 1111
TalkingStatus: on
```

#### **Stop talking Example**

```
Event: ConfbridgeTalking
Privilege: call, all
Channel: SIP/mypeer-00000001
Uniqueid: 1303308745.0
Conference: 1111
TalkingStatus: off
```

## ConfBridge CLI Commands

### ConfBridge CLI Commands

ConfBridge offers several commands that may be invoked from the Asterisk CLI.

#### ***confbridge kick <conference> <channel>***

Removes the specified channel from the conference, e.g.:

```
*CLI> confbridge kick 1111 SIP/mypeer-00000000
Kicking SIP/mypeer-00000000 from confbridge 1111
```

### On This Page

- ConfBridge CLI Commands
  - confbridge kick <conference> <channel>
  - confbridge list
  - confbridge list <conference>
  - confbridge lock <conference>
  - confbridge unlock <conference>
  - confbridge mute <conference> <channel>
  - confbridge unmute <conference> <channel>
  - confbridge record start <conference> <file>
  - confbridge record stop <conference>
  - confbridge show menus
  - confbridge show menu <menu name>
  - confbridge show profile bridges
  - confbridge show profile bridge <bridge>
  - confbridge show profile users
  - confbridge show profile user <user>

#### ***confbridge list***

Shows a summary listing of all bridges, e.g.:

```
*CLI> confbridge list
Conference Bridge Name      Users  Marked  Locked?
=====
1111                        1      0      unlocked
```

#### ***confbridge list <conference>***

Shows a detailed listing of participants in a specified conference, e.g.:

```
*CLI> confbridge list 1111
Channel      User Profile      Bridge Profile      Menu
=====
SIP/mypeer-00000001  default_user      1111                sample_user_menu
```

#### ***confbridge lock <conference>***

Locks a specified conference so that only Admin users can join, e.g.:

```
*CLI> confbridge lock 1111
Conference 1111 is locked.
```

#### ***confbridge unlock <conference>***

Unlocks a specified conference so that only Admin users can join, e.g.:

```
*CLI> confbridge unlock 1111
Conference 1111 is unlocked.
```

#### ***confbridge mute <conference> <channel>***

Mutes a specified user in a specified conference, e.g.:

```
*CLI> confbridge mute 1111 SIP/mypeer-00000001
Muting SIP/mypeer-00000001 from confbridge 1111
```

#### ***confbridge unmute <conference> <channel>***

Unmutes a specified user in a specified conference, e.g.:

```
*CLI> confbridge unmute 1111 SIP/mypeer-00000001
Unmuting SIP/mypeer-00000001 from confbridge 1111
```

### ***confbridge record start <conference> <file>***

Begins recording a conference. If "file" is specified, it will be used, otherwise, the Bridge Profile record\_file will be used. If the Bridge Profile does not specify a record\_file, one will be automatically generated in Asterisk's monitor directory. Usage:

```
*CLI> confbridge record start 1111
Recording started
*CLI> == Begin MixMonitor Recording ConfBridgeRecorder/conf-1111-uid-618880445
```

### ***confbridge record stop <conference>***

Stops recording the specified conference, e.g.:

```
*CLI> confbridge record stop 1111
Recording stopped.
*CLI> == MixMonitor close filestream (mixed)
== End MixMonitor Recording ConfBridgeRecorder/conf-1111-uid-618880445
```

### ***confbridge show menus***

Shows a listing of Conference Menus as defined in confbridge.conf, e.g.:

```
*CLI> confbridge show menus
----- Menu -----
sample_admin_menu
sample_user_menu
```

### ***confbridge show menu <menu name>***

Shows a detailed listing of a named Conference Menu, e.g.:

```
*CLI> confbridge show menu sample_admin_menu
Name: sample_admin_menu
*9=increase_talking_volume
*8=no_op
*7=decrease_talking_volume
*6=increase_listening_volume
*4=decrease_listening_volume
*3=admin_kick_last
*2=admin_toggle_conference_lock
*1=toggle_mute
*=playback_and_continue(conf-adminmenu)
```

### ***confbridge show profile bridges***

Shows a listing of Bridge Profiles as defined in confbridge.conf, e.g.:

```
*CLI> confbridge show profile bridges
----- Bridge Profiles -----
1111
default_bridge
```

### ***confbridge show profile bridge <bridge>***

Shows a detailed listing of a named Bridge Profile, e.g.:



```
*CLI> confbridge show profile bridge 1111
-----
Name:                1111
Internal Sample Rate: 16000
Mixing Interval:     10
Record Conference:   no
Record File:         Auto Generated
Max Members:         No Limit
sound_only_person:   conf-onlyperson
sound_has_joined:    conf-hasjoin
sound_has_left:      conf-hasleft
sound_kicked:        conf-kicked
sound_muted:         conf-muted
sound_unmuted:       conf-unmuted
sound_there_are:     conf-thereare
sound_other_in_party: conf-otherinparty
sound_place_into_conference: conf-placeintoconf
sound_wait_for_leader: conf-waitforleader
sound_get_pin:       conf-getpin
sound_invalid_pin:   conf-invalidpin
sound_locked:        conf-locked
sound_unlocked_now:  conf-unlockednow
sound_lockednow:     conf-lockednow
sound_error_menu:    conf-errormenu
```

### ***confbridge show profile users***

Shows a listing of User Profiles as defined in confbridge.conf, e.g.:

```
*CLI> confbridge show profile users
----- User Profiles -----
awesomeusers
default_user
```

### ***confbridge show profile user <user>***

Shows a detailed listing of a named Bridge Profile, e.g.:

```
*CLI> confbridge show profile user default_user
-----
Name:                default_user
Admin:               false
Marked User:         false
Start Muted:         false
MOH When Empty:      enabled
MOH Class:           default
Quiet:               disabled
Wait Marked:         disabled
END Marked:          disabled
Drop_silence:        enabled
Silence Threshold:   2500ms
Talking Threshold:   160ms
Denoise:             disabled
Talk Detect Events:  disabled
DTMF Pass Through:   disabled
PIN:                 None
Announce User Count: enabled
Announce join/leave: enabled
Announce User Count all: enabled
```

## ConfBridge Configuration

### ConfBridge Configuration

ConfBridge Profiles and Menus are configured in the `confbridge.conf` configuration file - normally located at `/etc/asterisk/confbridge.conf`. The file contains three reserved sections:

- `[general]`
- `[default_bridge]`
- `[default_user]`

The **[general]** section is currently unused, but is reserved for future use. The **[default\_bridge]** section contains all options invoked when ConfBridge is instantiated from the dialplan without a bridge profile argument.

The **[default\_user]** section contains all options invoked when ConfBridge is instantiated from the dialplan without a user profile argument.

Each section contains a **type** definition. The type definition determines the function of the section. The three **types** are:

- `bridge`
- `user`
- `menu`

**bridge** is used to denote Bridge Profile section definitions.

**user** is used to denote User Profile section definitions.

**menu** is used to denote Conference Menu section definitions.

All other sections, defined by a section identifier encapsulated in square brackets, are user-definable.

#### Example

This is an example, using invalid options and functions, of a `confbridge.conf` configuration file, displaying the organizational layout. The various options and functions are described later in this page.

```
[general]
; comments are preceded by a comma
;
; the general section is blank
;
[default_bridge]
type=bridge
; Bridge Profile options go here
myoption=value
myoption2=othervalue
;
[default_user]
type=user
; User Profile options go here
myoption=value
myoption2=othervalue
;
[sample_menu]
type=menu
; Conferece Menu options go here
DTMF=function
otherDTMF=otherFunction
;
```

### Bridge Profile Configuration Options

A Bridge Profile provides the following configuration options:

Option	Values	Description	Notes
<code>type</code>	<code>bridge</code>	Set this to <code>bridge</code> to configure a bridge profile	
<code>max_members</code>	integer; e.g. 50	Limits the number of participants for a single conference to a specific number. By default, conferences have no participant limit. After the limit is reached, the conference will be locked until someone leaves. Admin-level users are exempt from this limit and will still be able to join otherwise-locked, because of limit, conferences.	

### On This Page

- [ConfBridge Configuration](#)
  - [Bridge Profile Configuration Options](#)
  - [User Profile Configuration Options](#)
  - [Conference Menu Configuration Options](#)

record_conference	yes/no	Records the conference call starting when the first user enters the room, and ending when the last user exits the room. The default recorded filename is 'confbridge-<name of conference bridge>-<start time>.wav and the default format is 8kHz signed linear. By default, this option is disabled. This file will be located in the configured monitoring directory as set in asterisk.conf	
record_file	path, e.g. /tmp/myfiles	When record_conference is set to yes, the specific name of the recorded file can be set using this option. Note that since multiple conferences may use the same Bridge profile, this can cause issues, depending on the configuration. It is recommended to only use this option dynamically with the CONFBRIDGE() dialplan function. This allows the recorded name to be specified and a unique name to be chosen. By default, the recorded file is stored in Asterisk's spool/monitor directory, with a unique filename starting with the 'confbridge' prefix.	
internal_sample_rate	auto, 8000, 12000, 16000, 24000, 32000, 44100, 48000, 96000, 192000	Sets the internal native sample rate at which to mix the conference. The "auto" option allows Asterisk to adjust the sample rate to the best quality / performance based on the participant makeup. Numbered values lock the rate to the specified numerical rate. If a defined number does not match an internal sampling rate supported by Asterisk, the nearest sampling rate will be used instead.	
mixing_interval	10, 20, 40, 80	Sets, in milliseconds, the internal mixing interval. By default, the mixing interval of a bridge is 20ms. This setting reflects how "tight" or "loose" the mixing will be for the conference. Lower intervals provide a "tighter" sound with less delay in the bridge and consume more system resources. Higher intervals provide a "looser" sound with more delay in the bridge and consume less resources	
video_mode	none, follow_talker, last_marked, first_marked	Configured video (as opposed to audio) distribution method for conference participants. Participants must use the same video codec. Confbridge does not provide MCU functionality. It does not transcode, scale, transrate, or otherwise manipulate the video. Options are "none," where no video source is set by default and a video source may be later set via AMI or DTMF actions; "follow_talker," where video distribution follows whomever is talking and providing video; "last_marked," where the last marked user with video capabilities to join the conference will be the single video source distributed to all other participants - when the current video source leaves, the marked user previous to the last-joined will be used as the video source; and "first-marked," where the first marked user with video capabilities to join the conference will be the single video source distributed to all other participants - when the current video source leaves, the marked user that joined next will be used as the video source. Use of video in conjunction with the jitterbuffer results in the audio being slightly out of sync with the video - because the jitterbuffer only operates on the audio stream, not the video stream. Jitterbuffer should be disabled when video is used.	
sound_join	filename	The sound played to the bridge when a user joins, typically some kind of beep sound	
sound_leave	filename	The sound played to the bridge when a user leaves, also typically some kind of beep sound	
sound_has_joined	filename	The sound played as a user intro, e.g. "xxxx has joined the conference."	
sound_has_left	filename	The sound played as a user parts the conference, e.g. "xxxx has left the conference."	
sound_kicked	filename	The sound played to a user who has been kicked from the conference.	
sound_muted	filename	The sound played to a user when the mute option is toggled on.	
sound_unmuted	filename	The sound played to a user when the mute option is toggled off.	
sound_only_person	filename	The sound played when a user is the only person in the conference.	
sound_only_one	filename	The sound played to a user when there is only one other person in the conference.	
sound_there_are	filename	The sound played when announcing how many users there are in a conference.	
sound_other_in_party	filename	Used in conjunction with the sound_there_are option, used like "sound_there_are" <number of participants> "sound_other_in_party"	
sound_place_into_conference	filename	The sound played when someone is placed into a conference, after waiting for a marked user.	
sound_wait_for_leader	filename	The sound played when a user is placed into a conference that cannot start until a marked user enters.	
sound_leader_has_left	filename	The sound played when the last marked user leaves the conference.	

sound_get_pin	filename	The sound played when prompting for a conference PIN	
sound_invalid_pin	filename	The sound played when an invalid PIN is entered too many (3) times	
sound_locked	filename	The sound played to a user trying to join a locked conference.	
sound_locked_now	filename	The sound played to an Admin-level user after toggling the conference to locked mode.	
sound_unlocked_now	filename	The sound played to an Admin-level user after toggling the conference to unlocked mode.	
sound_error_menu	filename	The sound played when an invalid menu option is entered.	
sound_participants_muted	filename	The sound played when all non-admin participants are muted.	<b>New in Asterisk 11</b>
sound_participants_unmuted	filename	The sound played when all non-admin participants are unmuted	<b>New in Asterisk 11</b>

### Example

In this example, a Bridge Profile called "fancybridge" will be created. It will be configured to allow up to 20 callers, and will be set to mix at 10ms (tight mixing) at an automatic sampling rate. Additionally, it will be recorded.

```
[fancybridge]
type=bridge
max_members=20
mixing_interval=10
internal_sample_rate=auto
record_conference=yes
```

### User Profile Configuration Options

A User Profile provides the following configuration options:

Option	Values	Description	Notes
type	user	Set this to user to configure a user profile	
admin	yes/no	Sets if the user is an Admin or not. By default, no.	
marked	yes/no	Sets if the user is Marked or not. By default, no.	
startmuted	yes/no	sets if the user should start out muted. By default, no.	
music_on_hold_when_empty	yes/no	Sets whether music on hold should be played when only one person is in the conference or when the user is waiting on a marked user to enter the conference. By default, off.	
music_on_hold_class	music on hold class	Sets the music on hold class to use for music on hold.	
quiet	yes/no	When set to "yes," enter/leave prompts and user introductions are not played. By default, no.	
announce_user_count	yes/no	Sets if the number of users in the conference should be announced to the caller. By default, no.	
announce_user_count_all	yes/no; or an integer	Sets if the number of users should be announced to all other users in the conference when someone joins. When set to a number, the announcement will only occur once the user count is above the specified number	
announce_only_user	yes/no	Sets if the only user announcement should be played when someone enters an empty conference. By default, yes.	
announcement	filename	If set, the sound file specified by <code>filename</code> will be played to the user, and only the user, upon joining the conference bridge.	<b>New in Asterisk 11</b>
wait_marked	yes/no	Sets if the user must wait for another marked user to enter before joining the conference. By default, no.	
end_marked	yes/no	If enabled, every user with this option in their profile will be removed from the conference when the last marked user exists the conference.	

dsp_drop_silence	yes/no	Drops what Asterisk detects as silence from entering into the bridge. Enabling this option will drastically improve performance and help remove the buildup of background noise from the conference. This option is highly recommended for large conferences, due to its performance improvements.	
dsp_talking_threshold	integer in milliseconds	The time, in milliseconds, by default 160, of sound above what the DSP has established as base-line silence for a user, before that user is considered to be talking. This value affects several options: <ol style="list-style-type: none"> <li>1. Audio is only mixed out of a user's incoming audio stream if talking is detected. If this value is set too loose, the user will hear themselves briefly each time they begin talking until the DSP has time to establish that they are in fact talking.</li> <li>2. When talker detection AMI events are enabled, this value determines when talking has begun, which causes AMI events to fire. If this value is set too tight, AMI events may be falsely triggered by variants in the background noise of the caller.</li> <li>3. The drop_silence option depends on this value to determine when the user's audio should be mixed into the bridge after periods of silence. If this value is too loose, the beginning of a user's speech will get cut off as they transition from silence to talking.</li> </ol>	
dsp_silence_threshold	integer in milliseconds	The time, in milliseconds, by default 2500, of sound falling within what the DSP has established as the baseline silence, before a user is considered to be silent. The best way to approach this option is to set it slightly above the maximum amount of milliseconds of silence a user may generate during natural speech. This value affects several operations: <ol style="list-style-type: none"> <li>1. When talker detection AMI events are enabled, this value determines when the user has stopped talking after a period of talking. If this value is set too low, AMI events indicating that the user has stopped talking may get falsely sent out when the user briefly pauses during mid sentence.</li> <li>2. The drop_silence option depends on this value to determine when the user's audio should begin to be dropped from the bridge, after the user stops talking. If this value is set too low, the user's audio stream may sound choppy to other participants.</li> </ol>	
talk_detection_events	yes/no	Sets whether or not notifications of when a user begins and ends talking should be sent out as events over AMI. By default, no.	
denoise	yes/no	Whether or not a noise reduction filter should be applied to the audio before mixing. By default, off. This requires codec_speex to be built and installed. Do not confuse this option with drop_silence. denoise is useful if there is a lot of background noise for a user, as it attempts to remove the noise while still preserving the speech. This option does not remove silence from being mixed into the conference and does come at the cost of a slight performance hit.	
jitterbuffer	yes/no	Whether or not to place a jitter buffer on the caller's audio stream before any audio mixing is performed. This option is highly recommended, but will add a slight delay to the audio and will incur a slight performance penalty. This option makes use of the JITTERBUFFER dialplan function's default adaptive jitter buffer. For a more fine-tuned jitter buffer, disable this option and use the JITTERBUFFER dialplan function on the calling channel, before it enters the ConfBridge application.	
pin	integer	Sets if the user must enter a PIN before joining the conference. The user will be prompted for the PIN.	
announce_join_leave	yes/no	When enabled, this option prompts the user for their name when entering the conference. After the name is recorded, it will be played as the user enters and exists the conference. By default, no.	
dtmf_passthrough	yes/no	Whether or not DTMF received from users should pass through the conference to other users. By default, no.	

### Example

In this example, we will create a user profile called "fancyuser" that includes music on hold, user count announcements, join/leave announcements, silence detection, noise reduction and requires a PIN of 456.

```
[fancyuser]
type=user
music_on_hold_when_empty=yes
music_on_hold_class=default
announce_user_count_all=yes
announce_join_leave=yes
dsp_drop_silence=yes
denoise=yes
pin=456
```

## Conference Menu Configuration Options

A Conference Menu provides the following configuration options:

Option	Values	Description	Notes
type	menu	Set this to menu to configure a conference menu	
playback	(<name of audio file1>&<name of audio file2>&...)	Plays back an audio file, or a string of audio files chained together using the & character, to the user and then immediately returns them to the conference.	
playback_and_continue	(<name of audio file 1>&<name of audio file 2>&...)	Plays back an audio file, or a series of audio files chained together using the & character, to the user while continuing the collect the DTMF sequence. This is useful when using a menu prompt that describes all of the menu options. Note that any DTMF during this action will terminate the prompt's playback.	
toggle_mute		Toggles mute on and off. When a user is muted, they will not be able to speak to other conference users, but they can still listen to other users. While muted, DTMF keys from the caller will continue to be collected.	
no_op		This action does nothing. Its only real purpose exists for being able to reserve a sequence in the configuration as a menu exit sequence.	
decrease_listening_volume		Decreases the caller's listening volume. Everything they hear will sound quieter.	
increase_listening_volume		Increases the caller's listening volume. Everything they hear will sound louder.	
reset_listening_volume		Resets the caller's listening volume to the default level.	
decrease_talking_volume		Decreases the caller's talking volume. Everything they say will sound quieter to other callers.	
increase_talking_volume		Increases the caller's talking volume. Everything they say will sound louder to other callers.	
reset_talking_volume		Resets the caller's talking volume to the default level.	
dialplan_exec	(context,exten,priority)	Allows one to escape from the conference and execute commands in the dialplan. Once the dialplan exits, the user will be put back into the conference.	
leave_conference		Allows a user to exit the conference and continue execution in the dialplan.	
admin_kick_last		Allows an Admin to remove the last participant from the conference. This action only works for users whose User Profiles set them as conference Admins.	
admin_toggle_conference_lock		Allows an Admin to toggle locking and unlocking the conference. When the conference is locked, only other Admin users can join. When the conference is unlocked, any user may join up to the limit defined by the max_members Bridge Profile option. This action only works for users whose User Profiles set them as conference Admins.	
admin_toggle_mute_participants		Allows an Admin to mute/unmute all non-admin participants in the conference.	<b>New in Asterisk 11</b>
set_as_single_video_src		Allows a user to set themselves as the single video distribution source for all other participants. This overrides the video_mode setting.	
release_as_single_video_src		Allows a user to release themselves as the single video source. Upon release of the video source, and/or if video_mode is set to "none," this action will result in the conference returning to whatever video mode the Bridge Profile is using. This action will have no effect if the user is not currently the video source. The user is also not guaranteed that the use of this action will prevent them from becoming the video source later.	
participant_count		Plays back the current number of participants into the conference.	<b>New in Asterisk 11</b>

### Example

In this example, we'll create a menu called "fancymenu." This menu will utilize many of the options listed above. We will construct a features menu that plays when the user enters the \* character. Since we will do this using the playback\_and\_continue option, we will define other menu items as being a

"subset" of the \* command, e.g. \*4, so that once the user presses \*, they can listen to the menu options and then press the specific "after-star" option, e.g. 4, to affect the option. Additionally, we will duplicate those same sub-features as non-\* features, so that the user does not need to have entered the \* menu structure in order to affect the options, they can just press the key, e.g. "4" at any time, regardless of whether or not they're in the \*-tree.

```
[fancymenu]
type=menu
*=playback_and_continue(conf-togglemute&press&digits/1&silence/1&conf-leave&press&digits/2&silence/1&add-a-caller&press&digit
s/3&silence/1&conf-decrease-talking&press&digits/4&silence/1&reset-talking&press&digits/5&silence/1&increase-talking&press&di
gits/6&silence/1&conf-decrease-listening&press&digits/7&silence/1&conf-reset-listening&press&digits/8&silence/1&conf-increase
-listening&press&digits/9&silence/1&conf-exit-menu&press&digits/0)
*1=toggle_mute
1=toggle_mute
*2=leave_conference
2=leave_conference
*3=dialplan_exec(addcallers,1,1)
3=dialplan_exec(addcallers,1,1)
*4=decrease_listening_volume
4=decrease_listening_volume
*5=reset_listening_volume
5=reset_listening_volume
*6=increase_listening_volume
6=increase_listening_volume
*7=decrease_talking_volume
7=decrease_talking_volume
*8=reset_talking_volume
8=reset_talking_volume
*9=increase_talking_volume
9=increase_talking_volume
*0=no_op
0=no_op
```

Of particular note in this example, we're calling the dialplan\_exec option. Here, we're specifying "addcaller,1,1." This means that when someone dials 3, Asterisk will escape them out of the bridge momentarily to go execute priority 1 of extension 1 in the addcaller context of the dialplan (extensions.conf). Our dialplan, including the addcaller context, in this case, might look like:

```
[addcaller]
exten => 1,1,Originate(SIP/otherpeer,exten,conferences,100,1)

[conferences]
exten => 100,1,ConfBridge(1234)
```

Thus, when someone dials "3" while in the bridge, they'll Originate a call from the dialplan that puts SIP/otherpeer into the conference. Once the dial has completed, the person that dialed "3" will find themselves back in the bridge, with the other participants.

## ConfBridge Functions

### Function CONFBRIDGE

The CONFBRIDGE dialplan function is used to set customized Bridge and/or User Profiles on a channel for the ConfBridge application. It uses the same options defined in confbridge.conf and allows the creation of dynamic, dialplan-driven conferences.

#### On This Page

- [Function CONFBRIDGE](#)
  - [Syntax](#)
  - [Examples](#)
- [Function CONFBRIDGE\\_INFO](#)
  - [Syntax](#)

### Syntax

```
CONFBRIDGE(type,option)
```

- type - Refers to which type of profile the option belongs to. Type can be either "bridge" or "user."
- option - Refers to the confbridge.conf option that is to be set dynamically on the channel. This can also refer to an existing Bridge or User Profile by using the keyword "template." In this case, an existing Bridge or User Profile can be appended or modified on-the-fly.

### Examples

#### Example 1

In this example, the custom set User Profile on this channel enables announce\_join\_leave (so users will be announced as they come and go), sets users to join muted (so that they're not able to speak), and pushes them into bridge "1."

```
exten => 1,1,Answer()  
exten => 1,n,Set(CONFBRIDGE(user,announce_join_leave)=yes)  
exten => 1,n,Set(CONFBRIDGE(user,startmuted)=yes)  
exten => 1,n,ConfBridge(1)
```

#### Example 2

In this example, we will include an existing User Profile, the default\_user User Profile as defined in confbridge.conf, and we will set additional parameters (admin and marked) that aren't already defined in the default\_user User Profile.

```
exten => 1,1,Answer()  
exten => 1,n,Set(CONFBRIDGE(user,template)=default_user)  
exten => 1,n,Set(CONFBRIDGE(user,admin)=yes)  
exten => 1,n,Set(CONFBRIDGE(user,marked)=yes)  
exten => 1,n,ConfBridge(1)
```

### Function CONFBRIDGE\_INFO

The CONFBRIDGE\_INFO dialplan function is used to retrieve information about a conference, such as locked/unlocked status and the number of parties including admins and marked users.

### Syntax

```
CONFBRIDGE_INFO(type,conf)
```

- type - Refers to which information type to be retrieved. Type can be either "parties," "admins," "marked," or "locked."
- conf - Refers to the name of the conference being referenced.

The CONFBRIDGE\_INFO function returns a non-negative integer for valid conference identifiers, 0 or 1 for locked, and "" for invalid conference identifiers.



## ConfBridge Sound Prompts

The following Conference Menu and Bridge Profile options sound files are available as part of the latest Asterisk core sounds package - currently only available in the English language package.

- confbridge-begin-glorious-a - "The conference will begin when our glorious leader arrives."
- confbridge-begin-glorious-b - "The conference will begin when our **glorious leader** arrives."
- confbridge-begin-glorious-c - "The conference will begin when our **glorious leader arrives.**"
- confbridge-conf-begin - "The conference will now begin."
- confbridge-conf-end - "The conference has ended."
- confbridge-dec-list-vol-in - "To decrease the audio volume from other participants..."
- confbridge-dec-list-vol-out - "...to decrease the audio volume from other participants."
- confbridge-dec-talk-vol-in - "To decrease your speaking volume to other participants..."
- confbridge-dec-talk-vol-out - "...to decrease your speaking volume to other participants."
- confbridge-has-joined - "...has joined the conference."
- confbridge-has-left - "...has left the conference."
- confbridge-inc-list-vol-in - "To increase the audio volume from other participants..."
- confbridge-inc-list-vol-out - "...to increase the audio volume from other participants."
- confbridge-inc-talk-vol-in - "To increase your speaking volume to other participants..."
- confbridge-inc-talk-vol-out - "...to increase your speaking volume to other participants."
- confbridge-invalid - "You have entered an invalid option."
- confbridge-leave-in - "To leave the conference..."
- confbridge-leave-out - "...to leave the conference."
- confbridge-lock-extended - "...to lock or unlock the conference. When a conference is locked, only conference administrators can join."
- confbridge-lock-in - "To lock or unlock the conference."
- confbridge-lock-no-join - "The conference is currently locked and cannot be joined."
- confbridge-lock-out 0 - "...to lock or unlock the conference."
- confbridge-locked - "The conference is now locked."
- confbridge-menu-exit-in - "To exit the menu..."
- confbridge-menu-exit-out - "...to exit the menu."
- confbridge-mute-extended - "...to mute or unmute yourself. When you are muted, you cannot send audio to other participants; however you will still hear audio from other unmuted participants."
- confbridge-mute-in - "To mute or unmute yourself..."
- confbridge-mute-out - "...to mute or unmute yourself."
- confbridge-muted - "You are now muted."
- confbridge-only-one - "There is currently one other participant in the conference."
- confbridge-only-participant - "You are currently the only participant in the conference."
- confbridge-participants - "...participants in the conference."
- confbridge-pin-bad - "You have entered too many invalid personal identification numbers."
- confbridge-pin - "Please enter your personal identification number followed by the pound or hash key."
- confbridge-remove-last-in - "To remove the participant who most recently joined the conference..."
- confbridge-remove-last-out - "...to remove the participant who most recently joined the conference."
- confbridge-removed - "You have been removed from the conference."
- confbridge-rest-list-vol-in - "To reset the audio volume of the conference to the default level..."
- confbridge-rest-list-vol-out - "...to reset the audio volume of the conference to the default level."
- confbridge-rest-talk-vol-in - "To reset your speaking volume to the default level..."
- confbridge-rest-talk-vol-out - "...to reset your speaking volume to the default level."
- confbridge-there-are - "There are currently..."
- confbridge-unlocked - "The conference is now unlocked."
- confbridge-unmuted - "You are no longer muted."

# Functions



Under Construction



Top-level page for pages describing how to use functions, pointing to the auto-generated function reference pages and possibly going into detail on a few of the commonly used or very detailed functions.

# Simple Message Desk Interface (SMDI) Integration

## Accessing SMDI information in the dialplan.

There are two dialplan functions that can be used to access the details of incoming SMDI messages.

```
*CLI> core show function SMDI_MSG_RETRIEVE

-- Info about function 'SMDI_MSG_RETRIEVE' --
```

### Syntax

```
SMDI_MSG_RETRIEVE(<smdi port>,<search key>[,timeout[,options]])
```

### Synopsis

Retrieve an SMDI message.

### Description

This function is used to retrieve an incoming SMDI message. It returns an ID which can be used with the SMDI\_MSG() function to access details of the message. Note that this is a destructive function in the sense that once an SMDI message is retrieved using this function, it is no longer in the global SMDI message queue, and can not be accessed by any other Asterisk channels. The timeout for this function is optional, and the default is 3 seconds. When providing a timeout, it should be in milliseconds. The default search is done on the forwarding station ID. However, if you set one of the search key options in the options field, you can change this behavior.

### Options

- t - Instead of searching on the forwarding station, search on the message desk terminal.
- n - Instead of searching on the forwarding station, search on the message desk number.

```
*CLI> core show function SMDI_MSG

-- Info about function 'SMDI_MSG' --
```

### Syntax

```
SMDI_MSG(<message_id>,<component>)
```

### Synopsis

Retrieve details about an SMDI message.

### Description

This function is used to access details of an SMDI message that was pulled from the incoming SMDI message queue using the SMDI\_MSG\_RETRIEVE() function. Valid message components are:

- station - The forwarding station
- callerid - The callerID of the calling party that was forwarded
- type - The call type. The value here is the exact character that came in on the SMDI link. Typically, example values are: D - Direct Calls, A - Forward All Calls, B - Forward Busy Calls, N - Forward No Answer Calls

Here is an example of how to use these functions:

```

; Retrieve the SMDI message that is associated with the number that
; was called in Asterisk.
exten => _0XXX,1,Set(SMDI_MSG_ID=${SMDI_MSG_RETRIEVE(/dev/tty0,${EXTEN})})

; Ensure that the message was retrieved.
exten => _0XXX,n,GotoIf(["x${SMDI_MSG_ID}" != "x"]?processcall:hangup)
exten => _0XXX,n(hangup),NoOp(No SMDI message retrieved for ${EXTEN})

; Grab the details out of the SMDI message.
exten => _0XXX,n(processcall),NoOp(Message found for ${EXTEN})
exten => _0XXX,n,Set(SMDI_EXTEN=${SMDI_MSG(${SMDI_MSG_ID},station)})
exten => _0XXX,n,Set(SMDI_CID=${SMDI_MSG(${SMDI_MSG_ID},callerid)})

; Map SMDI message types to the right voicemail option.  If it is "B", use the
; busy option.  Otherwise, use the unavailable option.
exten => _0XXX,n,GotoIf(["${SMDI_MSG(${SMDI_MSG_ID},type)}" == "B"]?usebusy:useunavail)

exten => _0XXX,n(usebusy),Set(SMDI_VM_TYPE=b)
exten => _0XXX,n,Goto(continue)

exten => _0XXX,n,(useunavail),Set(SMDI_VM_TYPE=u)

exten => _0XXX,n(continue),NoOp( Process the rest of the call ... )

```

## Ensuring complete MWI information over SMDI

Another change has been made to ensure that MWI state is properly propagated over the SMDI link. This replaces the use of `externnotify=smdi` for `voicemail.conf`. The issue is that we have to poll mailboxes occasionally for changes that were made using an IMAP client. So, this ability was added to `res_smdi`. To configure this, there is a new section in `smdi.conf`. It looks like this:

```
[mailboxes]
; This section configures parameters related to MWI handling for the SMDI link.
;
; This option configures the polling interval used to check to see if the
; mailboxes have any new messages. This option is specified in seconds.
; The default value is 10 seconds.
;
;pollinginterval=10
;
; Before specifying mailboxes, you must specify an SMDI interface. All mailbox
; definitions that follow will correspond to that SMDI interface. If you
; specify another interface, then all definitions following that will correspond
; to the new interface.
;
; Every other entry in this section of the configuration file is interpreted as
; a mapping between the mailbox ID on the SMDI link, and the local Asterisk
; mailbox name. In many cases, they are the same thing, but they still must be
; listed here so that this module knows which mailboxes it needs to pay
; attention to.
;
; Syntax:
; <SMDI mailbox ID>=<Asterisk Mailbox Name>[@Asterisk Voicemail Context]
;
; If no Asterisk voicemail context is specified, "default" will be assumed.
;
;
;smddiport=/dev/ttyS0
;2565551234=1234@vmcontext1
;2565555678=5678@vmcontext2
;smddiport=/dev/ttyS1
;2565559999=9999
```

# Reporting

Asterisk has two reporting systems. **Call Detail Records (CDR)** and **Channel Event Logging (CEL)**. Both of these systems log specific events that occur on calls and individual channels. The events and their details are provided in a machine readable format separate from Asterisk's standard logging and debug facilities. Both systems provide at least CSV output and utilize other modules to output through a variety of back-end interfaces.

[Call Detail Records](#) is the older system that provides one or more records for each call depending on what version of Asterisk you are using and what is happening in the call. It is useful for administrators who need a simple way to track what calls have taken place on the Asterisk system. It isn't recommended for generating billing data.

[Channel Event Logging](#) is the newer system that provides much more detail than CDR. CEL is designed to excel where CDR fails and this is noticed first in the amount of detail that CEL provides. For any given calling scenario CDR may produce one or two simple records compared to dozens of records for CEL. If you want very precise data on every call happening in Asterisk then you should use CEL. Hence, CEL is the recommended reporting system to use for generating billing data.

If you are looking for logging, debugging or Application Programming Interfaces then you should check out the following resources:

- [Asterisk Logging and Asterisk Logging Configuration](#)
- [Asterisk Interfaces](#)

In This Section
<ul style="list-style-type: none"><li>• <a href="#">Call Detail Records (CDR)</a></li><li>• <a href="#">Channel Event Logging (CEL)</a></li></ul>

# Call Detail Records (CDR)

Top-level page for all things CDR

## CDR Variables

If the channel has a CDR, that CDR has its own set of variables which can be accessed just like channel variables. The following builtin variables are available.

- `$(CDR(clid))` Caller ID
- `$(CDR(src))` Source
- `$(CDR(dst))` Destination
- `$(CDR(dcontext))` Destination context
- `$(CDR(channel))` Channel name
- `$(CDR(dstchannel))` Destination channel
- `$(CDR(lastapp))` Last app executed
- `$(CDR(lastdata))` Last app's arguments
- `$(CDR(start))` Time the call started.
- `$(CDR(answer))` Time the call was answered.
- `$(CDR(end))` Time the call ended.
- `$(CDR(duration))` Duration of the call.
- `$(CDR(billsec))` Duration of the call once it was answered.
- `$(CDR(disposition))` ANSWERED, NO ANSWER, BUSY
- `$(CDR(amaflags))` DOCUMENTATION, BILL, IGNORE etc
- `$(CDR(accountcode))` The channel's account code.
- `$(CDR(uniqueid))` The channel's unique id.
- `$(CDR(userfield))` The channels uses specified field.

In addition, you can set your own extra variables by using `Set(CDR(name)=value)`. These variables can be output into a text-format CDR by using the `cdr_custom` CDR driver; see the `cdr_custom.conf.sample` file in the `configs` directory for an example of how to do this.



## CDR Storage Backends

Top-level page for information about storage backends for Asterisk's CDR engine.

## MSSQL CDR Backend

Asterisk can currently store CDRs into a Microsoft SQL Server database in two different ways: `cdr_odbc` or `cdr_tds`

Call Data Records can be stored using unixODBC (which requires the FreeTDS package) `cdr_odbc` or directly by using just the FreeTDS package `cdr_tds`. The following provide some examples known to get asterisk working with mssql.



Only choose one db connector.

### ODBC using `cdr_odbc`

Compile, configure, and install the latest unixODBC package:

```
tar -zxvf unixODBC-2.2.9.tar.gz && cd unixODBC-2.2.9 && ./configure --sysconfdir=/etc --prefix=/usr --disable-gui && make && make install
```

Compile, configure, and install the latest FreeTDS package:

```
tar -zxvf freetds-0.62.4.tar.gz && cd freetds-0.62.4 && ./configure --prefix=/usr --with-tdsver=7.0 \ --with-unixodbc=/usr/lib && make && make install
```

Compile, or recompile, asterisk so that it will now add support for `cdr_odbc`.

```
make clean && ./configure --with-odbc && make update && make && make install
```

Setup odbc configuration files.

These are working examples from my system. You will need to modify for your setup. You are not required to store usernames or passwords here.

`/etc/odbcinst.ini`

```
[FreeTDS]
Description = FreeTDS ODBC driver for MSSQL
Driver = /usr/lib/libtdsodbc.so
Setup = /usr/lib/libtdsS.so
FileUsage = 1
```

`/etc/odbc.ini`

```
[MSSQL-asterisk]
description = Asterisk ODBC for MSSQL
driver = FreeTDS
server = 192.168.1.25
port = 1433
database = voipdb
tds_version = 7.0
language = us_english
```



Only install one database connector. Do not confuse asterisk by using both ODBC (`cdr_odbc`) and FreeTDS (`cdr_tds`). This command will erase the contents of `cdr_tds.conf`

```
[ -f /etc/asterisk/cdr_tds.conf ] > /etc/asterisk/cdr_tds.conf
```



unixODBC requires the freeTDS package, but asterisk does not call freeTDS directly.

Now set up `cdr_odbc` configuration files.

These are working samples from my system. You will need to modify for your setup. Define your usernames and passwords here, secure file as well.

`/etc/asterisk/cdr_odbc.conf`

```
[global]
dsn=MSSQL-asterisk
username=voipdbuser
password=voipdbpass
loguniqueid=yes
```

And finally, create the 'cdr' table in your mssql database.

```

CREATE TABLE cdr (
  [calldate] [datetime] NOT NULL ,
  [clid] [varchar] (80) NOT NULL ,
  [src] [varchar] (80) NOT NULL ,
  [dst] [varchar] (80) NOT NULL ,
  [dcontext] [varchar] (80) NOT NULL ,
  [channel] [varchar] (80) NOT NULL ,
  [dstchannel] [varchar] (80) NOT NULL ,
  [lastapp] [varchar] (80) NOT NULL ,
  [lastdata] [varchar] (80) NOT NULL ,
  [duration] [int] NOT NULL ,
  [billsec] [int] NOT NULL ,
  [disposition] [varchar] (45) NOT NULL ,
  [amaflags] [int] NOT NULL ,
  [accountcode] [varchar] (20) NOT NULL ,
  [uniqueid] [varchar] (150) NOT NULL ,
  [userfield] [varchar] (255) NOT NULL
)

```

Start asterisk in verbose mode.

You should see that asterisk logs a connection to the database and will now record every call to the database when it's complete.

### TDS, using cdr\_tds

Compile, configure, and install the latest FreeTDS package:

```
tar -zxvf freetds-0.62.4.tar.gz && cd freetds-0.62.4 && ./configure --prefix=/usr --with-tdsver=7.0 make && make install
```

Compile, or recompile, asterisk so that it will now add support for cdr\_tds.

```
make clean && ./configure --with-tds && make update && make && make install
```



Only install one database connector. Do not confuse asterisk by using both ODBC (cdr\_odbc) and FreeTDS (cdr\_tds). This command will erase the contents of cdr\_odbc.conf

```
[ -f /etc/asterisk/cdr_odbc.conf ] > /etc/asterisk/cdr_odbc.conf
```

Setup cdr\_tds configuration files.

These are working samples from my system. You will need to modify for your setup. Define your usernames and passwords here, secure file as well.

```
/etc/asterisk/cdr_tds.conf [global] hostname=192.168.1.25 port=1433 dbname=voipdb user=voipdbuser password=voipdbpass charset=BIG5
```

And finally, create the 'cdr' table in your mssql database.

```

CREATE TABLE cdr (
  [accountcode] [varchar] (20) NULL ,
  [src] [varchar] (80) NULL ,
  [dst] [varchar] (80) NULL ,
  [dcontext] [varchar] (80) NULL ,
  [clid] [varchar] (80) NULL ,
  [channel] [varchar] (80) NULL ,
  [dstchannel] [varchar] (80) NULL ,
  [lastapp] [varchar] (80) NULL ,
  [lastdata] [varchar] (80) NULL ,
  [start] [datetime] NULL ,
  [answer] [datetime] NULL ,
  [end] [datetime] NULL ,
  [duration] [int] NULL ,
  [billsec] [int] NULL ,
  [disposition] [varchar] (20) NULL ,
  [amaflags] [varchar] (16) NULL ,
  [uniqueid] [varchar] (150) NULL ,
  [userfield] [varchar] (256) NULL
)

```


Start asterisk in verbose mode.

You should see that asterisk logs a connection to the database and will now record every call to the database when it's complete.

## MySQL CDR Backend

### ODBC

Using MySQL for CDR records is supported by using ODBC and the `cdr_adaptive_odbc` module (depends on `res_odbc`).

 The below `cdr_mysql` module has been deprecated in 1.8.

### Native

Alternatively, there is a native MySQL CDR module.

To use it, configure the module in `cdr_mysql.conf`. Create a table called `cdr` under the database name you will be using the following schema.

```
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 '',  
  uniqueid varchar(32) NOT NULL default '',  
  userfield varchar(255) NOT NULL default ''  
);
```

### In 1.8 and later

The following columns can also be defined:

```
peeraccount varchar(20) NOT NULL default ''  
  linkedid varchar(32) NOT NULL default ''  
  sequence int(11) NOT NULL default '0'
```

## PostgreSQL CDR Backend

If you want to go directly to postgresql database, and have the cdr\_pgsql.so compiled you can use the following sample setup. On Debian, before compiling asterisk, just install libpqxx-dev. Other distros will likely have a similiar package. Once you have the compile done, copy the sample cdr\_pgsql.conf file or create your own.

Here is a sample:

/etc/asterisk/cdr\_pgsql.conf

```
; Sample Asterisk config file for CDR logging to PostgreSQL
[global]
hostname=localhost
port=5432
dbname=asterisk
password=password
user=postgres
table=cdr
```

Now create a table in postgresql for your cdrs

```
CREATE TABLE cdr (
    calldate timestamp NOT NULL ,
    clid varchar (80) NOT NULL ,
    src varchar (80) NOT NULL ,
    dst varchar (80) NOT NULL ,
    dcontext varchar (80) NOT NULL ,
    channel varchar (80) NOT NULL ,
    dstchannel varchar (80) NOT NULL ,
    lastapp varchar (80) NOT NULL ,
    lastdata varchar (80) NOT NULL ,
    duration int NOT NULL ,
    billsec int NOT NULL ,
    disposition varchar (45) NOT NULL ,
    amaflags int NOT NULL ,
    accountcode varchar (20) NOT NULL ,
    uniqueid varchar (150) NOT NULL ,
    userfield varchar (255) NOT NULL
);
```

In 1.8 and later

The following columns can also be defined:

```
peeraccount varchar(20) NOT NULL
linkedid varchar(150) NOT NULL
sequence int NOT NULL
```

## SQLite 2 CDR Backend

SQLite version 2 is supported in cdr\_sqlite.

## SQLite 3 CDR Backend

SQLite version 3 is supported in cdr\_sqlite3\_custom.

## RADIUS CDR Backend

### What is needed

- FreeRADIUS server
- Radiusclient-ng library
- Asterisk PBX

### Installation of the Radiusclient library

Download the sources

From <http://developer.berlios.de/projects/radiusclient-ng/>

Untar the source tarball:

```
root@localhost:/usr/local/src# tar xvfz radiusclient-ng-0.5.2.tar.gz
```

Compile and install the library:

```
root@localhost:/usr/local/src# cd radiusclient-ng-0.5.2
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# ./configure
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# make
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# make install
```

### Configuration of the Radiusclient library

By default all the configuration files of the radiusclient library will be in `/usr/local/etc/radiusclient-ng` directory.

File "radiusclient.conf" Open the file and find lines containing the following:

```
authserver localhost
```

This is the hostname or IP address of the RADIUS server used for authentication. You will have to change this unless the server is running on the same host as your Asterisk PBX.

```
acctserver localhost
```

This is the hostname or IP address of the RADIUS server used for accounting. You will have to change this unless the server is running on the same host as your Asterisk PBX.

File "servers"

RADIUS protocol uses simple access control mechanism based on shared secrets that allows RADIUS servers to limit access from RADIUS clients.

A RADIUS server is configured with a secret string and only RADIUS clients that have the same secret will be accepted.

You need to configure a shared secret for each server you have configured in radiusclient.conf file in the previous step. The shared secrets are stored in `/usr/local/etc/radiusclient-ng/servers` file.

Each line contains hostname of a RADIUS server and shared secret used in communication with that server. The two values are separated by white spaces. Configure shared secrets for every RADIUS server you are going to use.

```
File "dictionary"
```

Asterisk uses some attributes that are not included in the dictionary of radiusclient library, therefore it is necessary to add them. A file called dictionary.digium (kept in the contrib dir) was created to list all new attributes used by Asterisk. Add to the end of the main dictionary

file `/usr/local/etc/radiusclient-ng/dictionary` the line:

```
$INCLUDE /path/to/dictionary.digium
```

### Install FreeRADIUS Server (Version 1.1.1)

Download sources tarball from:

<http://freeradius.org/>

Untar, configure, build, and install the server:

```
root@localhost:/usr/local/src# tar xvfz freeradius-1.1.1.tar.gz
root@localhost:/usr/local/src# cd freeradius-1.1.1
root@localhost:/usr/local/src/freeradius-1.1.1# ./configure
root@localhost:/usr/local/src/freeradius-1.1.1# make
root@localhost:/usr/local/src/freeradius-1.1.1# make install
```

All the configuration files of FreeRADIUS server will be in `/usr/local/etc/raddb` directory.



## Configuration of the FreeRADIUS Server

There are several files that have to be modified to configure the RADIUS server. These are presented next.

File "clients.conf"

File /usr/local/etc/raddb/clients.conf contains description of RADIUS clients that are allowed to use the server. For each of the clients you need to specify its hostname or IP address and also a shared secret. The shared secret must be the same string you configured in radiusclient library.

Example:

```
client myhost { secret = mysecret shortname = foo }
```

This fragment allows access from RADIUS clients on "myhost" if they use "mysecret" as the shared secret. The file already contains an entry for localhost (127.0.0.1), so if you are running the RADIUS server on the same host as your Asterisk server, then modify the existing entry instead, replacing the default password.

File "dictionary"



As of version 1.1.2, the dictionary.digium file ships with FreeRADIUS.

The following procedure brings the dictionary.digium file to previous versions of FreeRADIUS.

File /usr/local/etc/raddb/dictionary contains the dictionary of FreeRADIUS server. You have to add the same dictionary file (dictionary.digium), which you added to the dictionary of radiusclient-ng library. You can include it into the main file, adding the following line at the end of file /usr/local/etc/raddb/dictionary:

```
$INCLUDE /path/to/dictionary.digium
```

That will include the same new attribute definitions that are used in radiusclient-ng library so the client and server will understand each other.

## Asterisk Accounting Configuration

Compilation and installation:

The module will be compiled as long as the radiusclient-ng library has been detected on your system.

By default FreeRADIUS server will log all accounting requests into /usr/local/var/log/radius/radacct directory in form of plain text files. The server will create one file for each hostname in the directory. The following example shows how the log files look like.

Asterisk now generates Call Detail Records. See /include/asterisk/cdr.h for all the fields which are recorded. By default, records in comma separated values will be created in /var/log/asterisk/cdr-csv.

The configuration file for cdr\_radius.so module is /etc/asterisk/cdr.conf

This is where you can set CDR related parameters as well as the path to the radiusclient-ng library configuration file.

Logged Values

- "Asterisk-Acc-Code", The account name of detail records
- "Asterisk-Src",
- "Asterisk-Dst",
- "Asterisk-Dst-Ctx", The destination context
- "Asterisk-Clid",
- "Asterisk-Chan", The channel
- "Asterisk-Dst-Chan", (if applicable)
- "Asterisk-Last-App", Last application run on the channel
- "Asterisk-Last-Data", Argument to the last channel
- "Asterisk-Start-Time",
- "Asterisk-Answer-Time",
- "Asterisk-End-Time",
- "Asterisk-Duration", Duration is the whole length that the entire call lasted. ie. call rx'd to hangup "end time" minus "start time"
- "Asterisk-Bill-Sec", The duration that a call was up after other end answered which will be <= to duration "end time" minus "answer time"
- "Asterisk-Disposition", ANSWERED, NO ANSWER, BUSY
- "Asterisk-AMA-Flags", DOCUMENTATION, BILL, IGNORE etc, specified on a per channel basis like accountcode.
- "Asterisk-Unique-ID", Unique call identifier
- "Asterisk-User-Field" User field set via SetCDRUserField

# Channel Event Logging (CEL)

## Overview

Asterisk's Channel Event Logging provides a mechanism for tracking many channel related events. CEL is granular and fine-grained, having been designed with billing information in mind. It supports many storage back-ends and is a great alternative to Call Detail Records for administrators that need extremely detailed event logs. The extensive detail will allow building of accurate billing or call-flow data.

Features:

- Control over which Asterisk applications are tracked.
- Control over which events should be raised.
- Configurable date format.
- Integration with the Asterisk Manager Interface.
- Integration with RADIUS
- Modules for various logging back-ends including customized CEL output, integration with ODBC, PGSQL, SQLite and TDS.

The child pages in this section discuss the configuration of Channel Event Logging.

## Events

While CEL, CDR and AMI are all basically event tracking mechanisms, the events tracked by CEL are focused on a use case for generating billing data. The specific events and fields are covered in the [Asterisk 12 CEL Specification](#).

## Historical Information and Changes

CEL underwent significant rework and improvement in Asterisk 12. Please see the specific changes as mentioned under the CEL section of the [New in 12](#) wiki page.

# CEL Design Goals

## Overview

This page is simply an effort at explaining the reason and purpose behind the design of CEL. You can skip this section if you don't care why things are the way they are and only want to get it up and running.

## Improving on CDR

CEL, or Channel Event Logging, was written with the hopes that it will help solve some of the problems that were difficult to address in CDR records. Some difficulties in CDR generation are the fact that the CDR record stores three events: the "Start" time, the "Answer" time, and the "End" time. Billing time is usually the difference between "Answer" and "End", and total call duration was the difference in time from "Start" to "End". The trouble with this direct and simple approach is the fact that calls can be transferred, put on hold, involved in conferencing, forwarded, etc. In general, those who create billing applications with Asterisk find they have to do all sorts of very creative things to overcome the shortcomings of CDR records, often supplementing the CDR records with AGI scripts and manager event filters.

## Channel-relevant events

The fundamental assumption is that the Channel is the fundamental communication object in asterisk, which basically provides a communication channel between two communication ports. It makes sense to have an event system aimed at recording important events on channels. Each event is attached to a channel, like ANSWER or HANGUP. Some events are meant to connect two or more channels, like the BRIDGE\_START event. Some events, like BLINDTRANSFER, are initiated by one channel, but affect two others. These events use the Peer field, like BRIDGE would, to point to the target channel.

## Billing requires detail

The design philosophy of CEL is to generate event data that can be grouped together to form a billing record. There are definite parallels between Asterisk Manager Interface events and CEL events, but there are some differences. Some events that are generated by CEL are not generated by the Manager interface (yet). CEL is optimized for databases, and Manager events are not. The focus of CEL is billing. The Manager interface is targeted to real-time monitoring and control of asterisk.

Looking at an example of all the interactions and events involved in a moderately complex call scenario can give the reader a feel for the complexities involved in billing. Please take note of the following sequence of events.

Remember that 150, 151, and 152 are all Zap extension numbers, and their respective devices are Zap/50, Zap/51, and Zap/52.

- 152 dials 151; 151 answers.
- 152 parks 151; 152 hangs up.
- 150 picks up the park (dials 701).
- 150 and 151 converse.
- 151 flashes hook; dials 152, talks to 152, then 151 flashes hook again for 3-way conference.
- 151 converses with the other two for a while, then hangs up.
- 150 and 152 keep conversing, then hang up. 150 hangs up first.(not that it matters).

Click the expansion link to see the resulting 42 CEL events with annotations.

▼ [Click here to expand...](#)

Note that the actual CEL events below are in CSV format and do not include the ;;; and text after that which gives a description of what the event represents.

```
"EV_CHAN_START","2007-05-09
12:46:16","fxs.52","152","","","s","extension","Zap/52-1","","","DOCUMENTATION","","1178736376.3","","" ;;; 152 takes the
phone off-hook
"EV_APP_START","2007-05-09
12:46:18","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51|30|TtWw","DOCUMENTATION","","1178736376.3"
;;; 152 finishes dialing 151
"EV_CHAN_START","2007-05-09
12:46:18","fxs.51","151","","","s","extension","Zap/51-1","","","DOCUMENTATION","","1178736378.4","","" ;;; 151 channel
created, starts ringing
(151 is ringing)
"EV_ANSWER","2007-05-09 12:46:19","","151","152","","","151","extension","Zap/51-1","AppDial","(Outgoing
Line)","DOCUMENTATION","","1178736378.4","","" ;;; 151 answers
"EV_ANSWER","2007-05-09
12:46:19","fxs.52","152","","","151","extension","Zap/52-1","Dial","Zap/51|30|TtWw","DOCUMENTATION","","1178736376.3","
","" ;;; so does 152 (???)
"EV_BRIDGE_START","2007-05-09
12:46:20","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51|30|TtWw","DOCUMENTATION","","1178736376.3","
","Zap/51-1" ;;; 152 and 151 are bridged
(151 and 152 are conversing)
"EV_BRIDGE_END","2007-05-09
12:46:25","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51|30|TtWw","DOCUMENTATION","","1178736376.3","
","" ;;; after 5 seconds, the bridge ends (152 dials #700?)
"EV_BRIDGE_START","2007-05-09
12:46:25","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51|30|TtWw","DOCUMENTATION","","1178736376.3","
","Zap/51-1" ;;; extraneous 0-second bridge?
"EV_BRIDGE_END","2007-05-09
12:46:25","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51|30|TtWw","DOCUMENTATION","","1178736376.3","
","" ;;;
```

"EV\_PARK\_START", "2007-05-09 12:46:27", "", "151", "152", "", "", "", "extension", "Zap/51-1", "Parked Call", "", "DOCUMENTATION", "", "1178736378.4", "", "" ;; 151 is parked

"EV\_HANGUP", "2007-05-09 12:46:29", "fxs.52", "152", "152", "", "", "h", "extension", "Zap/52-1", "", "", "DOCUMENTATION", "", "1178736376.3", "", "" ;; 152 hangs up 2 sec later

"EV\_CHAN\_END", "2007-05-09 12:46:29", "fxs.52", "152", "152", "", "", "h", "extension", "Zap/52-1", "", "", "DOCUMENTATION", "", "1178736376.3", "", "" ;; 152's channel goes away (151 is parked and listening to MOH! now, 150 picks up, and dials 701)

"EV\_CHAN\_START", "2007-05-09 12:47:08", "fxs.50", "150", "", "", "s", "extension", "Zap/50-1", "", "", "DOCUMENTATION", "", "1178736428.5", "", "" ;; 150 picks up the phone, dials 701

"EV\_PARK\_END", "2007-05-09 12:47:11", "", "151", "152", "", "", "", "extension", "Zap/51-1", "Parked Call", "", "DOCUMENTATION", "", "1178736378.4", "", "" ;; 151's park comes to end

"EV\_ANSWER", "2007-05-09 12:47:11", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "" ;; 150 gets answer (twice)

"EV\_ANSWER", "2007-05-09 12:47:12", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "" ;;

"EV\_BRIDGE\_START", "2007-05-09 12:47:12", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; bridge begins between 150 and recently parked 151 (150 and 151 are conversing, then 151 hits flash)

"EV\_CHAN\_START", "2007-05-09 12:47:51", "fxs.51", "151", "", "", "s", "extension", "Zap/51-2", "", "", "DOCUMENTATION", "", "1178736471.6", "", "" ;; 39 seconds later, 51-2 channel is created. (151 flashes hook)

"EV\_HOOKFLASH", "2007-05-09 12:47:51", "", "151", "152", "", "", "", "extension", "Zap/51-1", "Bridged Call", "Zap/50-1", "DOCUMENTATION", "", "1178736378.4", "", "Zap/51-2" ;; a marker to record that 151 flashed the hook

"EV\_BRIDGE\_END", "2007-05-09 12:47:51", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; bridge ends between 150 and 151

"EV\_BRIDGE\_START", "2007-05-09 12:47:51", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; 0-second bridge from 150 to ? 150 gets no sound at all

"EV\_BRIDGE\_END", "2007-05-09 12:47:51", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;;

"EV\_BRIDGE\_START", "2007-05-09 12:47:51", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; bridge start on 150 (151 has dialtone after hitting flash; dials 152)

"EV\_APP\_START", "2007-05-09 12:47:55", "fxs.51", "151", "151", "", "", "152", "extension", "Zap/51-2", "Dial", "Zap/52|30|TtWw", "DOCUMENTATION", "", "1178736471.6", "", "" ;; 151-2 dials 152 after 4 seconds

"EV\_CHAN\_START", "2007-05-09 12:47:55", "fxs.52", "152", "", "", "s", "extension", "Zap/52-1", "", "", "DOCUMENTATION", "", "1178736475.7", "", "" ;; 152 channel created to ring 152. (152 ringing)

"EV\_ANSWER", "2007-05-09 12:47:58", "", "152", "151", "", "", "152", "extension", "Zap/52-1", "AppDial", "(Outgoing Line)", "DOCUMENTATION", "", "1178736475.7", "", "" ;; 3 seconds later, 152 answers

"EV\_ANSWER", "2007-05-09 12:47:58", "fxs.51", "151", "151", "", "", "152", "extension", "Zap/51-2", "Dial", "Zap/52|30|TtWw", "DOCUMENTATION", "", "1178736471.6", "", "" ;; ... and 151-2 also answers

"EV\_BRIDGE\_START", "2007-05-09 12:47:59", "fxs.51", "151", "151", "", "", "152", "extension", "Zap/51-2", "Dial", "Zap/52|30|TtWw", "DOCUMENTATION", "", "1178736471.6", "", "Zap/51-1" ;; 1 second later, bridge formed betw. 151-2 and 151 (152 answers, 151 and 152 converging; 150 is listening to silence; 151 hits flash again... to start a 3way)

"EV\_3WAY\_START", "2007-05-09 12:48:58", "", "151", "152", "", "", "", "extension", "Zap/51-1", "Bridged Call", "Zap/50-1", "DOCUMENTATION", "", "1178736378.4", "", "Zap/51-2" ;; another hook-flash to begin a 3-way conference

"EV\_BRIDGE\_END", "2007-05-09 12:48:58", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; - almost 1 minute later, the bridge ends (151 flashes hook again)

"EV\_BRIDGE\_START", "2007-05-09 12:48:58", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; 0-second bridge at 150. (3 way conf formed)

"EV\_BRIDGE\_END", "2007-05-09 12:48:58", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;;

"EV\_BRIDGE\_START", "2007-05-09 12:48:58", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; bridge starts for 150 (3way now, then 151 hangs up.)

"EV\_BRIDGE\_END", "2007-05-09 12:49:26", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; 28 seconds later, bridge ends

"EV\_HANGUP", "2007-05-09 12:49:26", "", "151", "152", "", "", "", "extension", "Zap/51-1", "Bridged Call", "Zap/50-1", "DOCUMENTATION", "", "1178736378.4", "", "" ;; 151 hangs up, leaves 150 and 152 connected

"EV\_CHAN\_END", "2007-05-09 12:49:26", "", "151", "152", "", "", "", "extension", "Zap/51-1", "Bridged Call", "Zap/50-1", "DOCUMENTATION", "", "1178736378.4", "", "" ;; 151 channel ends

"EV\_CHAN\_END", "2007-05-09 12:49:26", "fxs.51", "151", "151", "", "", "h", "extension", "Zap/51-2ZOMBIE", "", "", "DOCUMENTATION", "", "1178736428.5", "", "" ;; 152-2 channel ends (zombie) (just 150 and 152 now)

"EV\_BRIDGE\_END", "2007-05-09 12:50:13", "fxs.50", "150", "150", "", "", "152", "extension", "Zap/50-1", "Dial", "Zap/52|30|TtWw", "DOCUMENTATION", "", "1178736471.6", "", "" ;; 47 sec later, the bridge from 150 to 152 ends

"EV\_HANGUP", "2007-05-09 12:50:13", "", "152", "151", "", "", "", "extension", "Zap/52-1", "Bridged Call", "Zap/50-1", "DOCUMENTATION", "", "1178736475.7", "", "" ;; 152 hangs up

"EV\_CHAN\_END", "2007-05-09 12:50:13", "", "152", "151", "", "", "", "extension", "Zap/52-1", "Bridged

```
Call,"Zap/50-1","DOCUMENTATION","", "1178736475.7","", "" ;; 152 channel ends
"EV_HANGUP","2007-05-09
12:50:13","fxs.50","150","150","", "", "h","extension","Zap/50-1","", "", "DOCUMENTATION","", "1178736471.6","", "" ;; 150 hangs
up
```

```
"EV_CHAN_END", "2007-05-09
12:50:13", "fxs.50", "150", "150", "", "", "h", "extension", "Zap/50-1", "", "", "DOCUMENTATION", "", "1178736471.6", "", "" ;;; 150 ends
```

In terms of Manager events, the above Events correspond to the following 80 Manager events shown in the expansion link.

▼ [Click here to expand...](#)

```
Event: Newchannel
Privilege: call,all
Channel: Zap/52-1
State: Rsrvd
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801102.5

Event: Newcallerid
Privilege: call,all
Channel: Zap/52-1
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801102.5
CID-CallingPres: 0 (Presentation Allowed, Not Screened)
Event: Newcallerid
Privilege: call,all
Channel: Zap/52-1
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801102.5
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newstate
Privilege: call,all
Channel: Zap/52-1
State: Ring
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801102.5
Event: Newexten
Privilege: call,all
Channel: Zap/52-1
Context: extension
Extension: 151
Priority: 1
Application: Set
AppData: CDR(myvar)=zingo
Uniqueid: 1178801102.5
Event: Newexten
Privilege: call,all
Channel: Zap/52-1
Context: extension
Extension: 151
Priority: 2
Application: Dial
AppData: Zap/51|30|TtWw
Uniqueid: 1178801102.5

Event: Newchannel
Privilege: call,all
Channel: Zap/51-1
State: Rsrvd
CallerIDNum: 151
CallerIDName: fxs.51
Uniqueid: 1178801108.6
Event: Newstate
Privilege: call,all
Channel: Zap/51-1
State: Ringing
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801108.6

Event: Dial
Privilege: call,all
SubEvent: Begin
Source: Zap/52-1
Destination: Zap/51-1
CallerIDNum: 152
CallerIDName: fxs.52
SrcUniqueID: 1178801102.5
DestUniqueID: 1178801108.6
Event: Newcallerid
Privilege: call,all
Channel: Zap/51-1
CallerIDNum: 151
CallerIDName: <Unknown>
```

Uniqueid: 1178801108.6  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newstate  
Privilege: call,all  
Channel: Zap/52-1  
State: Ringing  
CallerIDNum: 152  
CallerIDName: fxs.52  
Uniqueid: 1178801102.5  
Event: Newstate  
Privilege: call,all  
Channel: Zap/51-1  
State: Up  
CallerIDNum: 151  
CallerIDName: <unknown>  
Uniqueid: 1178801108.6  
Event: Newstate  
Privilege: call,all  
Channel: Zap/52-1  
State: Up  
CallerIDNum: 152  
CallerIDName: fxs.52  
Uniqueid: 1178801102.5

Event: Link  
Privilege: call,all  
Channel1: Zap/52-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801102.5  
Uniqueid2: 1178801108.6  
CallerID1: 152  
CallerID2: 151  
Event: Unlink  
Privilege: call,all  
Channel1: Zap/52-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801102.5  
Uniqueid2: 1178801108.6  
CallerID1: 152  
CallerID2: 151

Event: Link  
Privilege: call,all  
Channel1: Zap/52-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801102.5  
Uniqueid2: 1178801108.6  
CallerID1: 152  
CallerID2: 151  
Event: Unlink  
Privilege: call,all  
Channel1: Zap/52-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801102.5  
Uniqueid2: 1178801108.6  
CallerID1: 152  
CallerID2: 151

Event: ParkedCall  
Privilege: call,all  
Exten: 701  
Channel: Zap/51-1  
From: Zap/52-1  
Timeout: 45  
CallerIDNum: 151  
CallerIDName: <unknown>  
Event: Dial  
Privilege: call,all  
SubEvent: End  
Channel: Zap/52-1  
DialStatus: ANSWER

Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: h  
Priority: 1  
Application: Goto  
AppData: labell  
Uniqueid: 1178801102.5  
Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: h  
Priority: 4  
Application: Goto

AppData: label2  
Uniqueid: 1178801102.5

Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: h  
Priority: 2  
Application: NoOp  
AppData: In Hangup! myvar is zingo and accountcode is billsec is 26 and duration is 40 and end is 2007-05-10 06:45:42.  
Uniqueid: 1178801102.5

Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: h  
Priority: 3  
Application: Goto  
AppData: label3  
Uniqueid: 1178801102.5

Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: h  
Priority: 5  
Application: NoOp  
AppData: More Hangup message after hopping around"  
Uniqueid: 1178801102.5

Event: Hangup  
Privilege: call,all  
Channel: Zap/52-1  
Uniqueid: 1178801102.5  
Cause: 16  
Cause-txt: Normal Clearing

Event: Newchannel  
Privilege: call,all  
Channel: Zap/50-1  
State: Rsrvd  
CallerIDNum: 150  
CallerIDName: fxs.50  
Uniqueid: 1178801162.7

Event: Newcallerid  
Privilege: call,all  
Channel: Zap/50-1  
CallerIDNum: 150  
CallerIDName: fxs.50  
Uniqueid: 1178801162.7  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newcallerid  
Privilege: call,all  
Channel: Zap/50-1  
CallerIDNum: 150  
CallerIDName: fxs.50  
Uniqueid: 1178801162.7  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newstate  
Privilege: call,all  
Channel: Zap/50-1  
State: Ring  
CallerIDNum: 150  
CallerIDName: fxs.50  
Uniqueid: 1178801162.7

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: 701  
Priority: 1  
Application: ParkedCall  
AppData: 701  
Uniqueid: 1178801162.7

Event: UnParkedCall  
Privilege: call,all  
Exten: 701  
Channel: Zap/51-1  
From: Zap/50-1  
CallerIDNum: 151  
CallerIDName: <unknown>

Event: Newstate  
Privilege: call,all  
Channel: Zap/50-1  
State: Up  
CallerIDNum: 150



CallerIDName: fxs.50  
Uniqueid: 1178801162.7

Event: Link  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151  
Event: Newchannel  
Privilege: call,all  
Channel: Zap/51-2  
State: Rsrvd  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8

Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151  
Event: Link  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151  
Event: Link  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151  
Event: Newcallerid  
Privilege: call,all  
Channel: Zap/51-2  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newcallerid  
Privilege: call,all  
Channel: Zap/51-2  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)  
Event: Newstate  
Privilege: call,all  
Channel: Zap/51-2  
State: Ring  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8

Event: Newexten  
Privilege: call,all  
Channel: Zap/51-2  
Context: extension  
Extension: 152  
Priority: 1  
Application: Set  
AppData: CDR(myvar)=zingo  
Uniqueid: 1178801218.8  
Event: Newexten  
Privilege: call,all  
Channel: Zap/51-2  
Context: extension  
Extension: 152

Priority: 2  
Application: Dial  
AppData: Zap/52|30|TtWw  
Uniqueid: 1178801218.8

Event: Newchannel  
Privilege: call,all  
Channel: Zap/52-1  
State: Rsrvd  
CallerIDNum: 152  
CallerIDName: fxs.52  
Uniqueid: 1178801223.9  
Event: Newstate  
Privilege: call,all  
Channel: Zap/52-1  
State: Ringing  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801223.9  
Event: Dial  
Privilege: call,all  
SubEvent: Begin  
Source: Zap/51-2  
Destination: Zap/52-1  
CallerIDNum: 151  
CallerIDName: fxs.51  
SrcUniqueID: 1178801218.8  
DestUniqueID: 1178801223.9

Event: Newcallerid  
Privilege: call,all  
Channel: Zap/52-1  
CallerIDNum: 152  
CallerIDName: <Unknown>  
Uniqueid: 1178801223.9  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)  
Event: Newstate  
Privilege: call,all  
Channel: Zap/51-2  
State: Ringing  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8

Event: Newstate  
Privilege: call,all  
Channel: Zap/52-1  
State: Up  
CallerIDNum: 152  
CallerIDName: <unknown>  
Uniqueid: 1178801223.9  
Event: Newstate  
Privilege: call,all  
Channel: Zap/51-2  
State: Up  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8

Event: Link  
Privilege: call,all  
Channel1: Zap/51-2  
Channel2: Zap/52-1  
Uniqueid1: 1178801218.8  
Uniqueid2: 1178801223.9  
CallerID1: 151  
CallerID2: 152  
Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Link  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151  
Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7

Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Link  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151  
Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Hangup  
Privilege: call,all  
Channel: Zap/51-1  
Uniqueid: 1178801108.6  
Cause: 16  
Cause-txt: Normal

Clearing  
Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 1  
Application: Goto  
AppData: label1  
Uniqueid: 1178801162.7  
Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 4  
Application: Goto  
AppData: label2  
Uniqueid: 1178801162.7

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 2  
Application: NoOp  
AppData: In Hangup! myvar is and accountcode is billsec is 0 and duration is 0 and end is 2007-05-10 06:48:37.  
Uniqueid: 1178801162.7

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 3  
Application: Goto  
AppData: label3  
Uniqueid: 1178801162.7

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 5  
Application: NoOp  
AppData: More  
Hangup message after hopping around"  
Uniqueid: 1178801162.7

Event: Masquerade  
Privilege: call,all  
Clone: Zap/50-1  
CloneState: Up  
Original: Zap/51-2  
OriginalState: Up  
Event: Rename  
Privilege: call,all  
Oldname: Zap/50-1  
Newname: Zap/50-1<MASQ>  
Uniqueid: 1178801162.7

```
Event: Rename
Privilege: call,all
Oldname: Zap/51-2
Newname: Zap/50-1
Uniqueid: 1178801218.8
Event: Rename
Privilege: call,all
Oldname: Zap/50-1<MASQ>
Newname: Zap/51-2<ZOMBIE>
Uniqueid: 1178801162.7
Event: Hangup
Privilege: call,all
Channel: Zap/51-2<ZOMBIE>
Uniqueid: 1178801162.7
Cause: 0
Cause-txt: Unknown

Event: Unlink
Privilege: call,all
Channel1: Zap/50-1
Channel2: Zap/52-1
Uniqueid1: 1178801218.8
Uniqueid2: 1178801223.9
CallerID1: 150
CallerID2: 152
Event: Hangup
Privilege: call,all
Channel: Zap/52-1
Uniqueid: 1178801223.9
Cause: 16
Cause-txt: Normal Clearing

Event: Dial
Privilege: call,all
SubEvent: End
Channel: Zap/50-1
DialStatus: ANSWER
Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 1
Application: Goto
AppData: label1
Uniqueid: 1178801218.8

Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 4
Application: Goto
AppData: label2
Uniqueid: 1178801218.8
Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 2
Application: NoOp
AppData: In Hangup! myvar is and accountcode is billsec is 90 and duration is 94 and end is 2007-05-10 06:48:37.
Uniqueid: 1178801218.8

Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 3
Application: Goto
AppData: label3
Uniqueid: 1178801218.8
Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 5
Application: NoOp
AppData: More Hangup message after hopping around"
Uniqueid: 1178801218.8
Event: Hangup
Privilege: call,all
Channel: Zap/50-1
Uniqueid: 1178801218.8
```

Cause: 16  
Cause-txt: Normal Clearing

And, humorously enough, the above 80 manager events, or 42 CEL events, correspond to the following two CDR records (at the moment!):


```
""fxs.52" 152", "152", "h", "extension", "Zap/52-1", "Zap/51-1", "NoOp", "More Hangup message after hopping around", "2007-05-09  
17:35:56", "2007-05-09 17:36:20", "2007-05-09 17:36:36", "40", "16", "ANSWERED", "DOCUMENTATION", "", "1178753756.0", ""  
""fxs.50" 150", "150", "152", "extension", "Zap/50-1", "Zap/51-1", "NoOp", "More Hangup message after hopping around", "2007-05-09  
17:37:59", "2007-05-09 17:38:06", "2007-05-09 17:39:11", "72", "65", "ANSWERED", "DOCUMENTATION", "", "1178753871.3", ""
```

# CEL Applications and Functions

## Applications

The [CELGenUserEvent](#) application allows you to instruct Asterisk to generate a user defined event with custom type name.

The event triggered is the **USER\_DEFINED** event as listed in the [Asterisk 12 CEL Specification](#). The **eventtype** and **userdeftype** fields will be populated with data passed through the respective arguments provided to the CELGenUserEvent application.

 Please note that there is no restrictions on the name supplied. If it happens to match a standard CEL event name, it will look like that event was generated. This could be a blessing or a curse!

## Functions

Most CEL fields are populated by common channel data, so a unique function is not required to read or write that data on the channel. That channel data is already available via the [CHANNEL](#) function in currently supported versions of Asterisk.

Older versions of Asterisk had a unique CEL function. You can run "core show function CEL" to see if you have this function and display the help text.

# CEL Configuration Files

## CEL General Configuration

Primary CEL configuration settings are located in `cel.conf`. Note that CEL only publishes record types to back-ends that are enabled in the general CEL configuration.

## Back-end Configuration

Configuration for specific logging or storage back-ends is located in separate configuration files. The exception is the AMI and RADIUS back-ends. Sample configurations are provided with the Asterisk 12 source for all of these back-ends.

Name	Config File	Description
Manager (AMI)	<code>cel.conf</code>	The manager CEL output module publishes records over AMI as CEL events with the record type published under the "EventName" key. This module is configured in <code>cel.conf</code> in the [manager] section.
RADIUS	<code>cel.conf</code>	The RADIUS CEL output module allows the CEL engine to publish records to a RADIUS server. This module is configured in <code>cel.conf</code> in the [radius] section.
CEL Custom	<code>cel_custom.conf</code>	The Custom CEL output module provides logging capability to a CSV file in a format described in the configuration file.
ODBC	<code>cel_odbc.conf</code>	The ODBC CEL output module provides logging capability to any ODBC-compatible database.
PGSQL	<code>cel_pgsql.conf</code>	The PGSQL CEL output module provides logging capability to PostgreSQL databases when it is desirable to avoid the ODBC abstraction layer.
SQLite	<code>cel_sqlite3_custom.conf</code>	The SQLite CEL output module provides logging capability to a SQLite3 database in a format described in its configuration file.
TDS	<code>cel_tds.conf</code>	The TDS CEL output module provides logging capability to Sybase or Microsoft SQL Server databases when it is desirable to avoid the ODBC abstraction layer.

## CEL Configuration Examples

The child pages in this section provide examples of configuration with the specific logging or storage back-ends mentioned in [CEL Configuration Files](#).



## MSSQL CEL Backend

Asterisk can currently store Channel Events into an MSSQL database in two different ways: `cel_odbc` or `cel_tds`

Channel Event Records can be stored using unixODBC (which requires the FreeTDS package) `cel_odbc` or directly by using just the FreeTDS package `cel_tds`

The following provide some examples known to get asterisk working with mssql.



Only choose one db connector.

### ODBC using `cel_odbc`

Compile, configure, and install the latest unixODBC package:

```
tar -zxvf unixODBC-2.2.9.tar.gz && cd unixODBC-2.2.9 && ./configure --sysconfdir=/etc --prefix=/usr --disable-gui && make && make install
```

Compile, configure, and install the latest FreeTDS package:

```
tar -zxvf freetds-0.62.4.tar.gz && cd freetds-0.62.4 && ./configure --prefix=/usr --with-tdsver=7.0 \ --with-unixodbc=/usr/lib && make && make install
```

Compile, or recompile, asterisk so that it will now add support for `cel_odbc`.

```
make clean && ./configure --with-odbc && make update && make && make install
```

Setup odbc configuration files.

These are working examples from my system. You will need to modify for your setup. You are not required to store usernames or passwords here.

`/etc/odbcinst.ini`

```
[FreeTDS]
Description = FreeTDS ODBC driver for MSSQL
Driver = /usr/lib/libtdsodbc.so
Setup = /usr/lib/libtdsS.so
FileUsage = 1
```

`/etc/odbc.ini`

```
[MSSQL-asterisk]
description = Asterisk ODBC for MSSQL
driver = FreeTDS
server = 192.168.1.25
port = 1433
database = voipdb
tds_version = 7.0
language = us_english
```



Only install one database connector. Do not confuse asterisk by using both ODBC (`cel_odbc`) and FreeTDS (`cel_tds`). This command will erase the contents of `cel_tds.conf`

```
[ -f /etc/asterisk/cel_tds.conf ] > /etc/asterisk/cel_tds.conf
```



unixODBC requires the freeTDS package, but asterisk does not call freeTDS directly.

Now set up `cel_odbc` configuration files.

These are working samples from my system. You will need to modify for your setup. Define your usernames and passwords here, secure file as well.

`/etc/asterisk/cel_odbc.conf`

```
[global]
dsn=MSSQL-asterisk
username=voipdbuser
password=voipdbpass
loguniqueid=yes
```

And finally, create the 'cel' table in your mssql database.

```
CREATE TABLE cel (
  [eventtype] [varchar] (30) NOT NULL ,
  [eventtime] [datetime] NOT NULL ,
  [cidname] [varchar] (80) NOT NULL ,
  [cidnum] [varchar] (80) NOT NULL ,
  [cidani] [varchar] (80) NOT NULL ,
  [cidrdnis] [varchar] (80) NOT NULL ,
  [ciddnid] [varchar] (80) NOT NULL ,
  [exten] [varchar] (80) NOT NULL ,
  [context] [varchar] (80) NOT NULL ,
  [channname] [varchar] (80) NOT NULL ,
  [appname] [varchar] (80) NOT NULL ,
  [appdata] [varchar] (80) NOT NULL ,
  [amaflags] [int] NOT NULL ,
  [accountcode] [varchar] (20) NOT NULL ,
  [uniqueid] [varchar] (32) NOT NULL ,
  [peer] [varchar] (80) NOT NULL ,
  [userfield] [varchar] (255) NOT NULL
) ;
```

Start asterisk in verbose mode, you should see that asterisk logs a connection to the database and will now record every desired channel event at the moment it occurs.

### FreeTDS, using cel\_tds

Compile, configure, and install the latest FreeTDS package:

```
tar -zxvf freetds-0.62.4.tar.gz && cd freetds-0.62.4 && ./configure --prefix=/usr --with-tdsver=7.0 make && make install
```

Compile, or recompile, asterisk so that it will now add support for cel\_tds.

```
make clean && ./configure --with-tds && make update && make && make install
```



Only install one database connector. Do not confuse asterisk by using both ODBC (cel\_odbc) and FreeTDS (cel\_tds). This command will erase the contents of cel\_odbc.conf

```
[ -f /etc/asterisk/cel_odbc.conf ] > /etc/asterisk/cel_odbc.conf
```

Setup cel\_tds configuration files.

These are working samples from my system. You will need to modify for your setup. Define your usernames and passwords here, secure file as well.

/etc/asterisk/cel\_tds.conf

```
[global]
hostname=192.168.1.25
port=1433
dbname=voipdb
user=voipdbuser
password=voipdbpass
charset=BIG5
```

And finally, create the 'cel' table in your mssql database.

```
CREATE TABLE cel (
  [eventtype] [varchar] (30) NULL ,
  [eventtime] [datetime] NULL ,
  [cidname] [varchar] (80) NULL ,
  [cidnum] [varchar] (80) NULL ,
  [cidani] [varchar] (80) NULL ,
  [cidrdnis] [varchar] (80) NULL ,
  [ciddnid] [varchar] (80) NULL ,
  [exten] [varchar] (80) NULL ,
  [context] [varchar] (80) NULL ,
  [channname] [varchar] (80) NULL ,
  [appname] [varchar] (80) NULL ,
  [appdata] [varchar] (80) NULL ,
  [amaflags] [varchar] (16) NULL ,
  [accountcode] [varchar] (20) NULL ,
  [uniqueid] [varchar] (32) NULL ,
  [userfield] [varchar] (255) NULL ,
  [peer] [varchar] (80) NULL
) ;
```

Start asterisk in verbose mode, you should see that asterisk logs a connection to the database and will now record every call to the database when it's complete.

## PostgreSQL CEL Backend

If you want to go directly to postgresql database, and have the cel\_pgsql.so compiled you can use the following sample setup. On Debian, before compiling asterisk, just install libpqxx-dev. Other distros will likely have a similiar package.

Once you have the compile done, copy the sample cel\_pgsql.conf file or create your own.

Here is a sample:

/etc/asterisk/cel\_pgsql.conf

```
; Sample Asterisk config file for CEL logging to PostgreSQL
[global]
hostname=localhost
port=5432
dbname=asterisk
password=password
user=postgres
table=cel
```

Now create a table in postgresql for your cels

```
CREATE TABLE cel (
  id serial ,
  eventtype varchar (30) NOT NULL ,
  eventtime timestamp NOT NULL ,
  userdeftype varchar(255) NOT NULL ,
  cid_name varchar (80) NOT NULL ,
  cid_num varchar (80) NOT NULL ,
  cid_ani varchar (80) NOT NULL ,
  cid_rdnis varchar (80) NOT NULL ,
  cid_dnid varchar (80) NOT NULL ,
  exten varchar (80) NOT NULL ,
  context varchar (80) NOT NULL ,
  channame varchar (80) NOT NULL ,
  appname varchar (80) NOT NULL ,
  appdata varchar (80) NOT NULL ,
  amaflags int NOT NULL ,
  accountcode varchar (20) NOT NULL ,
  peeraccount varchar (20) NOT NULL ,
  uniqueid varchar (150) NOT NULL ,
  linkedid varchar (150) NOT NULL ,
  userfield varchar (255) NOT NULL ,
  peer varchar (80) NOT NULL
);
```

## RADIUS CEL Backend

### What is needed

- FreeRADIUS server
- Radiusclient-ng library
- Asterisk PBX

### Installation of the Radiusclient library

Download the sources

From <http://developer.berlios.de/projects/radiusclient-ng/>

Untar the source tarball:

```
root@localhost:/usr/local/src# tar xvfz radiusclient-ng-0.5.2.tar.gz
```

Compile and install the library:

```
root@localhost:/usr/local/src# cd radiusclient-ng-0.5.2
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# ./configure
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# make
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# make install
```

### Configuration of the Radiusclient library

By default all the configuration files of the radiusclient library will be in `/usr/local/etc/radiusclient-ng` directory.

File "radiusclient.conf" Open the file and find lines containing the following:

```
authserver localhost
```

This is the hostname or IP address of the RADIUS server used for authentication. You will have to change this unless the server is running on the same host as your Asterisk PBX.

```
acctserver localhost
```

This is the hostname or IP address of the RADIUS server used for accounting. You will have to change this unless the server is running on the same host as your Asterisk PBX.

File "servers"

RADIUS protocol uses simple access control mechanism based on shared secrets that allows RADIUS servers to limit access from RADIUS clients.

A RADIUS server is configured with a secret string and only RADIUS clients that have the same secret will be accepted.

You need to configure a shared secret for each server you have configured in radiusclient.conf file in the previous step. The shared secrets are stored in `/usr/local/etc/radiusclient-ng/servers` file.

Each line contains hostname of a RADIUS server and shared secret used in communication with that server. The two values are separated by white spaces. Configure shared secrets for every RADIUS server you are going to use.

```
File "dictionary"
```

Asterisk uses some attributes that are not included in the dictionary of radiusclient library, therefore it is necessary to add them. A file called dictionary.digium (kept in the contrib dir) was created to list all new attributes used by Asterisk. Add to the end of the main dictionary

file `/usr/local/etc/radiusclient-ng/dictionary` the line:

```
$INCLUDE /path/to/dictionary.digium
```

### Install FreeRADIUS Server (Version 1.1.1)

Download sources tarball from:

<http://freeradius.org/>

Untar, configure, build, and install the server:

```
root@localhost:/usr/local/src# tar xvfz freeradius-1.1.1.tar.gz
root@localhost:/usr/local/src# cd freeradius-1.1.1
root@localhost:/usr/local/src/freeradius-1.1.1# ./configure
root@localhost:/usr/local/src/freeradius-1.1.1# make
root@localhost:/usr/local/src/freeradius-1.1.1# make install
```

All the configuration files of FreeRADIUS server will be in `/usr/local/etc/raddb` directory.

## Configuration of the FreeRADIUS Server

There are several files that have to be modified to configure the RADIUS server. These are presented next.

File "clients.conf"

File /usr/local/etc/raddb/clients.conf contains description of RADIUS clients that are allowed to use the server. For each of the clients you need to specify its hostname or IP address and also a shared secret. The shared secret must be the same string you configured in radiusclient library.

Example:

```
client myhost { secret = mysecret shortname = foo }
```

This fragment allows access from RADIUS clients on "myhost" if they use "mysecret" as the shared secret. The file already contains an entry for localhost (127.0.0.1), so if you are running the RADIUS server on the same host as your Asterisk server, then modify the existing entry instead, replacing the default password.

File "dictionary"



As of version 1.1.2, the dictionary.digium file ships with FreeRADIUS.

The following procedure brings the dictionary.digium file to previous versions of FreeRADIUS.

File /usr/local/etc/raddb/dictionary contains the dictionary of FreeRADIUS server. You have to add the same dictionary file (dictionary.digium), which you added to the dictionary of radiusclient-ng library. You can include it into the main file, adding the following line at the end of file /usr/local/etc/raddb/dictionary:

```
$INCLUDE /path/to/dictionary.digium
```

That will include the same new attribute definitions that are used in radiusclient-ng library so the client and server will understand each other.

## Asterisk Accounting Configuration

Compilation and installation:

The module will be compiled as long as the radiusclient-ng library has been detected on your system.

By default FreeRADIUS server will log all accounting requests into /usr/local/var/log/radius/radacct directory in form of plain text files. The server will create one file for each hostname in the directory. The following example shows how the log files look like.

Asterisk now generates Call Detail Records. See /include/asterisk/cel.h for all the fields which are recorded. By default, records in comma separated values will be created in /var/log/asterisk/cel-csv.

The configuration file for cel\_radius.so module is :

/etc/asterisk/cel.conf This is where you can set CEL related parameters as well as the path to the radiusclient-ng library configuration file.

This is where you can set CDR related parameters as well as the path to the radiusclient-ng library configuration file.

Logged Values

- "Asterisk-Acc-Code", The account name of detail records
- "Asterisk-CidName",
- "Asterisk-CidNum",
- "Asterisk-Cidani",
- "Asterisk-Cidrnis",
- "Asterisk-Ciddnid",
- "Asterisk-Exten",
- "Asterisk-Context", The destination context
- "Asterisk-Channname", The channel name
- "Asterisk-Appname", Last application run on the channel
- "Asterisk-App-Data", Argument to the last channel
- "Asterisk-Event-Time",
- "Asterisk-Event-Type",
- "Asterisk-AMA-Flags", DOCUMENTATION, BILL, IGNORE etc, specified on a per channel basis like accountcode.
- "Asterisk-Unique-ID", Unique call identifier
- "Asterisk-User-Field" User field set via SetCELUserField
- "Asterisk-Peer" Name of the Peer for 2-channel events (like bridge)

# Interfaces

## Overview

There are many ways to interface Asterisk with scripts, other applications or storage systems. From the very trivial, such as using Asterisk Call Files, to sophisticated APIs such as the Asterisk Rest Interface. The pages in this section cover many of Asterisk's built-in interfaces, all of which provide some aspect of control, monitoring or storage.

### In this Section

- [Asterisk Calendaring](#)
- [Asterisk Call Files](#)
- [Asterisk Gateway Interface \(AGI\)](#)
- [Asterisk Manager Interface \(AMI\)](#)
- [Asterisk REST Interface \(ARI\)](#)
- [Back-end Database and Realtime Connectivity](#)
- [Distributed Device State](#)
- [Simple Network Management Protocol \(SNMP\) Support](#)
- [Speech Recognition API](#)
- [Utilizing the StatsD Dialplan Application](#)

Interface	Description
Asterisk Call Files	Asterisk can initiate calls based on information provided via flat text files in a spool directory. Asterisk can operate on these as soon as the file is inside the directory, or in the future depending on the timestamp of the file.
Asterisk Gateway Interface	AGI provides an interface between the Asterisk dialplan and an external program (via pipes, stdin and stdout) that wants to manipulate a channel in the dialplan.
Asterisk Management Interface	AMI is intended for management type functions. The manager is a client/server model over TCP. With the AMI you'll be able to control the PBX, originate calls, check mailbox status, monitor channels and queues as well as execute Asterisk commands.
Asterisk REST Interface	ARI is an asynchronous API that allows developers to build communications applications by exposing the raw primitive objects in Asterisk - channels, bridges, endpoints, media, etc. - through an intuitive REST interface. The state of the objects being controlled by the user are conveyed via JSON events over a WebSocket.
Calendaring	Asterisk's calendaring module allows read and write communication with various standardized calendar technologies. Asterisk dialplan can make use of calendar event information.
Database Connectivity	Asterisk has core support for ODBC connectivity, and many Asterisk modules provide support for a variety of back-end database connectivity, such as for MySQL or PostgreSQL.
Distributed Device State	Asterisk provides a few ways of distributing the state of devices between multiple Asterisk instances, whether on the same system or multiple systems.
SNMP	Basic SNMP support is included with Asterisk. This allows monitoring of a variety of Asterisk activity.
Speech Recognition API	The dialplan speech recognition API is based around a single speech utilities application file, which exports many applications to be used for speech recognition. These include an application to prepare for speech recognition, activate a grammar, and play back a sound file while waiting for the person to speak. Using a combination of these applications you can easily make a dialplan use speech recognition without worrying about what speech recognition engine is being used.
StatsD	This StatsD application is a dialplan application that is used to send statistics automatically whenever a call is made to an extension that employs the application. The user must provide the arguments to the application in the dialplan, but after that, the application will send statistics to StatsD without requiring the user to perform anymore actions whenever a call comes through that extension.

## Asterisk Calendaring

The Asterisk Calendaring API is provided by the `res_calendar` module. It aims to be a generic interface for integrating Asterisk with various calendaring technologies. The goal is to be able to support reading and writing of calendar events as well as allowing notification of pending events through the Asterisk dialplan.

There are four calendaring modules that ship with Asterisk that provide support for the following calendaring servers.

Calendar Server Support	Module Name
iCalendar	<code>res_calendar_icalendar.so</code>
CalDAV	<code>res_calender_caldav.so</code>
Microsoft Exchange Server	<code>res_calendar_exchange.so</code>
Microsoft Exchange Web Services	<code>res_calendar_ews.so</code>

All four modules support event notification. Both CalDAV and Exchange support reading and writing calendars, while iCalendar is a read-only format.

You can see list all registered calendar types at the CLI with "**calendar show types**".

```
*CLI> calendar show types
Type      Description
caldav    CalDAV calendars
exchange  MS Exchange calendars
ews       MS Exchange Web Service calend
ical      iCalendar .ics calendars
```

# Configuring Asterisk Calendaring

## The calendar.conf file

All asterisk calendaring modules are configured through calendar.conf. Each calendar module can define its own set of required parameters in addition to the parameters available to all calendar types. An effort has been made to keep all options the same in all calendaring modules, but some options will diverge over time as features are added to each module.

An example calendar.conf might look like:

```
[calendar_joe]
type = ical
url = https://example.com/home/jdoe/Calendar
user = jdoe
secret = mysecret
refresh = 15
timeframe = 600
autorereminder = 10
channel = SIP/joe
context = calendar_event_notify
extension = s
waittime = 30
```

- [The calendar.conf file](#)
  - [Module-independent settings](#)
  - [Module-dependent settings](#)
- [Specific Examples](#)
  - [Google Calendar](#)

## Module-independent settings

The settings related to calendar event notification are handled by the core calendaring API. These settings are:

- autorereminder - This allows the overriding of any alarms that may or may not be set for a calendar event. It is specified in minutes.
- refresh - How often to refresh the calendar data; specified in minutes.
- timeframe - How far into the future each calendar refresh should look. This is the amount of data that will be visible to queries from the dialplan. This setting should always be greater than or equal to the refresh setting or events may be missed. It is specified in minutes.
- channel - The channel that should be used for making the notification attempt.
- waittime - How long to wait, in seconds, for the channel to answer a notification attempt. There are two ways to specify how to handle a notification. One option is providing a context and extension, while the other is providing an application and the arguments to that application. One (and only one) of these options should be provided.
- context - The context of the extension to connect to the notification channel
- extension - The extension to connect to the notification. Note that the priority will always be 1.
- app - The dialplan application to execute upon the answer of a notification
- appdata - The data to pass to the notification dialplan application

## Module-dependent settings

Connection-related options are specific to each module. Currently, all modules take a url, user, and secret for configuration and no other module-specific settings have been implemented. At this time, no support for HTTP redirects has been implemented, so it is important to specify the correct URL-paying attention to any trailing slashes that may be necessary.

## Specific Examples

### Google Calendar

Requirements:

- The res\_calendar\_icalendar.so module must be loaded and running.
- You must have a Google calendar!
- Google requires that you share the calendar publicly. However you can choose to only share busy/free information to limit exposure to



details.

- Configure Asterisk to connect to the specific public calendar.

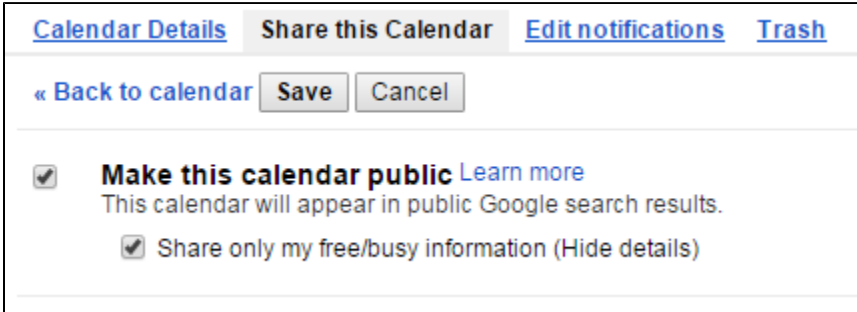
In this example we'll configure Asterisk to connect to your Google Calendar via the ical type calendar.

### Sharing your Google calendar

For detailed instructions see [Google's answer page](#).

We'll assume you are familiar with Google's interface and provide some brief instructions here.

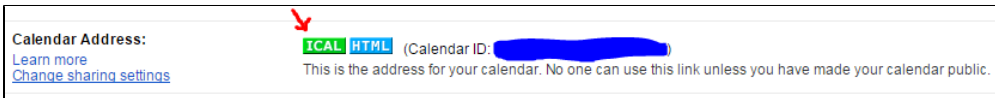
Go to your Google calendar settings, navigate to a specific calendar and open the **Share this Calendar** tab.



Make the calendar public and choose save.

### Get the Google calendar address

Go to the **Calendar Details** tab and find the button for getting the public ical address.



### Configure Asterisk with Google Calendar details

This specific configuration isn't too different than the generic example. Your calendar address goes in the **url** field.

```
[gcall]
type = ical
url = https://calendar.google.com/calendar/ical/example%40gmail.com/public/basic.ics
user = example@gmail.com
secret = a_very_secure_password
refresh = 15
timeframe = 60
```

Once you have a configuration you can startup Asterisk or else reload the modules. After this you can check to see if the calendar is being read. Use the commands "**calendar show calendars**" and "**calendar show calendar <calendar name>**"

```
CentOSLab*CLI> calendar show calendars
Calendar      Type      Status
-----
gcall         ical     busy
```

```
CentOSLab*CLI> calendar show calendar gcall
Name          : gcall
Notify channel :
Notify context :
Notify extension :
Notify applicatio :
Notify appdata :
Refresh time  : 15
Timeframe     : 60
Autorereminder : 0
Events
-----
Summary      : Busy
Description  :
Organizer    :
Location     :
Categories   :
Priority     : 0
UID          : 6b6ikcvq165i4701q5sdm7r1v4@google.com
Start       : 2015-11-25 05:40:00 PM -0600
End         : 2015-11-25 06:10:00 PM -0600
Alarm       :
```

The output should reflect your calendar settings and if it is reading from the calendar server you should see events that are present (on your calendar) within the configured Timeframe. If you don't see any events then go to your Google calendar and create an event within the timeframe. Save that event, then wait at least the Refresh time before checking the commands again to see if the event shows up.

# Calendar Dialplan Functions

## Read functions

The simplest dialplan query is the CALENDAR\_BUSY query. It takes a single option, the name of the calendar defined, and returns "1" for busy (including tentatively busy) and "0" for not busy.

For more information about a calendar event, a combination of CALENDAR\_QUERY and CALENDAR\_QUERY\_RESULT is used. CALENDAR\_QUERY takes the calendar name and optionally a start and end time in "unix time" (seconds from unix epoch). It returns an id that can be passed to CALENDAR\_QUERY\_RESULT along with a field name to return the data in that field. If multiple events are returned in the query, the number of the event in the list can be specified as well. The available fields to return are:

- **summary** - A short summary of the event
- **description** - The full description of the event
- **organizer** - Who organized the event
- **location** - Where the event is located
- **calendar** - The name of the calendar from calendar.conf
- **uid** - The unique identifier associated with the event
- **start** - The start of the event in seconds since Unix epoch
- **end** - The end of the event in seconds since Unix epoch
- **busystate** - The busy state 0=Free, 1=Tentative, 2=Busy
- **attendees** - A comma separated list of attendees as stored in the event and may include prefixes such as "mailto:".

When an event notification is sent to the dial plan, the CALENDAR\_EVENT function may be used to return the information about the event that is causing the notification. The fields that can be returned are the same as those from CALENDAR\_QUERY\_RESULT.

## Write functions

To write an event to a calendar, the CALENDAR\_WRITE function is used. This function takes a calendar name and also uses the same fields as CALENDAR\_QUERY\_RESULT. As a write function, it takes a set of comma-separated values that are in the same order as the specified fields. For example:

```
CALENDAR_WRITE(mycalendar,summary,organizer,start,end,busystate)= "My event", "mailto:jdoe@example.com", 228383580, 228383640, 1)
```

## Calendar dialplan examples

### Office hours

A common business PBX scenario would be executing dialplan logic based on when the business is open and the phones staffed. If the business is closed for holidays, it is sometimes desirable to play a message to the caller stating why the business is closed.

The standard way to do this in asterisk has been doing a series of GotoIfTime statements or time-based include statements. Either way can be tedious and requires someone with access to edit asterisk config files.

With calendaring, the administrator only needs to set up a calendar that contains the various holidays or even recurring events specifying the office hours. A custom greeting filename could even be contained in the description field for playback. For example:

```
[incoming]
exten => 5555551212,1,Answer
same => n,GotoIf(${CALENDAR_BUSY(officehours)}?closed:attendant,s,1)
same => n(closed),Set(id=${CALENDAR_QUERY(office,${EPOCH},${EPOCH})})
same => n,Set(soundfile=${CALENDAR_QUERY_RESULT(${id},description)})
same => n,Playback(${ISNULL(soundfile)} ? generic-closed :: ${soundfile})
same => n,Hangup
```

### Meeting reminders

One useful application of Asterisk Calendaring is the ability to execute dialplan logic based on an event notification. Most calendaring technologies allow a user to set an alarm for an event. If these alarms are set on a calendar that Asterisk is monitoring and the calendar is set up for event notification via calendar.conf, then Asterisk will execute notify the specified channel at the time of the alarm. If an override notification time is set with the autorereminder setting, then the notification would happen at that time instead.

The following example demonstrates the set up for a simple event notification that plays back a generic message followed by the time of the upcoming meeting. calendar.conf.

```
[calendar_joe]
type = ical
url = https://example.com/home/jdoe/Calendar
user = jdoe
secret = mysecret
refresh = 15
timeframe = 600
autorereminder = 10
channel = SIP/joe
context = calendar_event_notify
extension = s
waittime = 30
```

extensions.conf :

```
[calendar_event_notify]
exten => s,1,Answer
same => n,Playback(you-have-a-meeting-at)
same => n,SayUnixTime(${CALENDAR_EVENT(start)})
same => n,Hangup
```

### Writing an event

Both CalDAV and Exchange calendar servers support creating new events. The following example demonstrates writing a log of a call to a calendar.

```
[incoming]
exten => 6000,1,Set(start=${EPOCH})
exten => 6000,n,Dial(SIP/joe)
exten => h,1,Set(end=${EPOCH})
exten => h,n,Set(CALENDAR_WRITE(calendar_joe,summary,start,end)=Call from
${CALLERID(all)},{start},{end})
```

# Asterisk Call Files

## Overview

Asterisk has the ability to initiate a call from outside of the normal methods such as the dialplan, manager interface, or spooling interface.

Using the call file method, you must give Asterisk the following information:

- How to perform the call, similar to the Dial() application
- What to do when the call is answered

With call files you submit this information simply by creating a file with the required syntax and placing it in the `outgoing` spooling directory, located by default in `/var/spool/asterisk/outgoing/` (this is configurable in `asterisk.conf`).

The `pbx_spool.so` module watches the spooling directly, either using an event notification system supplied by the operating system such as `inotify` or `kqueue`, or by polling the directory each second when one of those notification systems is unavailable. When a new file appears, Asterisk initiates a new call based on the file's contents.



### Creating Files in the Spool Directory

Do **not** write or create the call file directly in the `outgoing` directory, but always create the file in another directory of the same filesystem and then move the file to the `outgoing` directory, or Asterisk may read a partial file.



### NFS Considerations

By default, Asterisk will prefer to use `inotify` or `kqueue` where available. When the spooling directory is on a remote server and is mounted via NFS, the `inotify` method will fail to work. You can force Asterisk to use the older polling method by passing the `--without-inotify` flag to configure during compilation (e.g. `./configure --without-inotify`).

## Call File Syntax

The call file consists of `<Key>: <value>` pairs; one per line.

Comments are indicated by a '#' character that begins a line, or follows a space or tab character. To be consistent with the configuration files in Asterisk, comments can also be indicated by a semicolon. However, the multiline comments (`;;;`) used in Asterisk configuration files are not supported. Semicolons can be escaped by a backslash.

The following keys-value pairs are used to specify how setup a call:

- `Channel: <channel>` - The channel to use for the new call, in the form **technology/resource** as in the Dial application. This value is required.
- `Callerid: <callerid>` - The caller id to use.
- `WaitTime: <number>` - How many seconds to wait for an answer before the call fails (ring cycle). Defaults to 45 seconds.
- `MaxRetries: <number>` - Number of retries before failing, not including the initial attempt. Default = 0 e.g. don't retry if fails.
- `RetryTime: <number>` - How many seconds to wait before retry. The default is 300 (5 minutes).
- `Account: <account>` - The account code for the call. This value will be assigned to CDR(accountcode)

When the call answers there are two choices:

1. Execute a single application, or
2. Execute the dialplan at the specified context/extension/priority.

### To execute an application:

- `Application: <appname>` - The application to execute
- `Data: <args>` - The application arguments

### To start executing applications in the dialplan:

- `Context: <context>` - The context in the dialplan
- `Extension: <exten>` - The extension in the specified context
- `Priority: <priority>` - The priority of the specified extension; (numeric or label)
- `Setvar: <var=value>` - You may also assign values to variables that will be available to the channel, as if you had performed a `Set(var=value)` in the dialplan. More than one `Setvar`: may be specified.

The processing of the call file ends when the call is answered and terminated; when the call was not answered in the initial attempt and subsequent retries; or if the call file can't be successfully read and parsed.

To specify what to do with the call file at the end of processing:

- `Archive: <yes|no>` - If "no" the call file is deleted. If set to "yes" the call file is moved to the "outgoing\_done" subdirectory of the Asterisk spool directory. The default is to delete the call file.

If the call file is archived, Asterisk will append to the call file:

- `Status: <exitstatus>` - Can be "Expired", "Completed" or "Failed"

Other lines generated by Asterisk:

Asterisk keep track of how many retries the call has already attempted, appending to the call file the following key-pairs in the form:

```
StartRetry: <pid> <retrycount> (<time>)  
EndRetry: <pid> <retrycount> (<time>)
```

With the main process ID (pid) of the Asterisk process, the retry number, and the attempts start and end times in `time_t` format.

## Directory locations

- `<astspooldir>/outgoing` - The outgoing dir, where call files are put for processing
- `<astspooldir>/outgoing_done` - The archive dir
- `<astspooldir>` - Is specified in `asterisk.conf`, usually `/var/spool/asterisk`

## How to schedule a call

Call files that have the time of the last modification in the future are ignored by Asterisk. This makes it possible to modify the time of a call file to the wanted time, move to the outgoing directory, and Asterisk will attempt to create the call at that time.

# Asterisk Gateway Interface (AGI)

 This page is under construction!

On This Page
<ul style="list-style-type: none"><li>• Overview</li><li>• AGI Libraries and Frameworks</li></ul>

## Overview

AGI is analogous to CGI in Apache. AGI provides an interface between the Asterisk dialplan and an external program that wants to manipulate a channel in the dialplan. In general, the interface is synchronous - actions taken on a channel from an AGI block and do not return until the action is completed.

## AGI Libraries and Frameworks

Name	Language	Website	Protocols
Adhearsion	Ruby	<a href="http://www.adhearsion.com/">http://www.adhearsion.com/</a>	AMI/FastAGI
Asterisk-Java	Java	<a href="https://blogs.reucon.com/asterisk-java/">https://blogs.reucon.com/asterisk-java/</a>	AMI/FastAGI
PAGI	PHP	<a href="https://github.com/marcelog/PAGI">https://github.com/marcelog/PAGI</a>	AGI
Panoramisk	Python+AsyncIO	<a href="https://github.com/gawel/panoramisk">https://github.com/gawel/panoramisk</a>	AMI/FastAGI
Pyst2	Python	<a href="https://github.com/rdegges/pyst2">https://github.com/rdegges/pyst2</a>	AMI/AGI
StarPy	Python+Twisted	<a href="https://github.com/asterisk/starpy">https://github.com/asterisk/starpy</a>	AMI/FastAGI
Nanoagi	C++	<a href="http://sourceforge.net/projects/nanoagi/">http://sourceforge.net/projects/nanoagi/</a>	AGI
AsterNET	.NET (C#/VB.net)	<a href="https://github.com/skrusty/AsterNET">https://github.com/skrusty/AsterNET</a>	AMI/FastAGI

## **Asterisk Manager Interface (AMI)**

What is the Asterisk Manager Interface, or AMI? Read on...



## The Asterisk Manager TCP IP API

The manager is a client/server model over TCP. With the manager interface, you'll be able to control the PBX, originate calls, check mailbox status, monitor channels and queues as well as execute Asterisk commands.

AMI is the standard management interface into your Asterisk server. You configure AMI in `manager.conf`. By default, AMI is available on TCP port 5038 if you enable it in `manager.conf`.

AMI receive commands, called "actions". These generate a "response" from Asterisk. Asterisk will also send "Events" containing various information messages about changes within Asterisk. Some actions generate an initial response and data in the form list of events. This format is created to make sure that extensive reports do not block the manager interface fully.

Management users are configured in the configuration file `manager.conf` and are given permissions for read and write, where write represents their ability to perform this class of "action", and read represents their ability to receive this class of "event".

If you develop AMI applications, treat the headers in Actions, Events and Responses as local to that particular message. There is no cross-message standardization of headers.

If you develop applications, please try to reuse existing manager headers and their interpretation. If you are unsure, discuss on the asterisk-dev mailing list.

Manager subscribes to extension status reports from all channels, to be able to generate events when an extension or device changes state. The level of details in these events may depend on the channel and device configuration. Please check each channel configuration file for more information. (in `sip.conf`, check the section on subscriptions and call limits)

## AMI Command Syntax

Management communication consists of tags of the form "header: value", terminated with an empty newline (`\r\n`) in the style of SMTP, HTTP, and other headers.

The first tag **MUST** be one of the following:

- Action: An action requested by the CLIENT to the Asterisk SERVER. Only one "Action" may be outstanding at any time.
- Response: A response to an action from the Asterisk SERVER to the CLIENT.
- Event: An event reported by the Asterisk SERVER to the CLIENT

## AMI Manager Commands

To see all of the available manager commands, use the "manager show commands" CLI command.

You can get more information about a manager command with the "manager show command <commandname>" CLI command in Asterisk.

## AMI Examples

- Login - Log a user into the manager interface.

```
Action: Login
Username: testuser
Secret: testsecret
```

- Originate - Originate a call from a channel to an extension.

```
Action: Originate
Channel: sip/12345
Exten: 1234
Context: default
```

- Originate - Originate a call from a channel to an extension without waiting for call to complete.

```
Action: Originate
Channel: sip/12345
Exten: 1234
Context: default
Async: yes
```

- Redirect with ExtraChannel:  
Attempted goal: Have a 'robot' program Redirect both ends of an already-connected call to a meetme room using the ExtraChannel feature through the management interface.

```
Action: Redirect
Channel: DAHDI/1-1
ExtraChannel: SIP/3064-7e00 (varies)
Exten: 680
Priority: 1
```

\*Where 680 is an extension that sends you to a MeetMe room.

There are a number of GUI tools that use the manager interface, please search the mailing list archives and the documentation page on the <http://www.asterisk.org> web site for more information.

## Ensuring all modules are loaded with AMI

It is possible to connect to the manager interface before all Asterisk modules are loaded. To ensure that an application does not send AMI actions that might require a module that has not yet loaded, the application can listen for the FullyBooted manager event. It will be sent upon connection if all modules have been loaded, or as soon as loading is complete. The event:

```
Event: FullyBooted  
Privilege: system,all  
Status: Fully Booted
```

## Some Standard AMI Headers

- Account: – Account Code (Status)
- AccountCode: – Account Code (cdr\_manager)
- ACL: <Y | N> – Does ACL exist for object ?
- Action: <action> – Request or notification of a particular action
- Address-IP: – IPAddress
- Address-Port: – IP port number
- Agent: <string> – Agent name
- AMAFlags: – AMA flag (cdr\_manager, sippeers)
- AnswerTime: – Time of answer (cdr\_manager)
- Append: <bool> – CDR userfield Append flag
- Application: – Application to use
- Async: – Whether or not to use fast setup
- AuthType: – Authentication type (for login or challenge) "md5"
- BillableSeconds: – Billable seconds for call (cdr\_manager)
- CallerID: – Caller id (name and number in Originate & cdr\_manager)
- CallerID: – CallerID number Number or "<unknown>" or "unknown" (should change to "<unknown>" in app\_queue)
- CallerID1: – Channel 1 CallerID (Link event)
- CallerID2: – Channel 2 CallerID (Link event)
- CallerIDName: – CallerID name Name or "<unknown>" or "unknown" (should change to "<unknown>" in app\_queue)
- Callgroup: – Call group for peer/user
- CallsTaken: <num> – Queue status variable
- Cause: <value> – Event change cause - "Expired"
- Cause: <value> – Hangupcause (channel.c)
- CID-CallingPres: – Caller ID calling presentation
- Channel: <channel> – Channel specifier
- Channel: <dialstring> – Dialstring in Originate
- Channel: <tech/[peer/username]> – Channel in Registry events (SIP, IAX2)
- Channel: <tech> – Technology (SIP/IAX2 etc) in Registry events
- ChannelType: – Tech: SIP, IAX2, DAHDI, MGCP etc
- Channel1: – Link channel 1
- Channel2: – Link channel 2
- ChanObjectType: – "peer", "user"
- Codecs: – Codec list
- CodecOrder: – Codec order, separated with comma ", "
- Command: – Cli command to run
- Context: – Context
- Count: <num> – Number of callers in queue
- Data: – Application data
- Default-addr-IP: – IP address to use before registration
- Default-Username: – Username part of URI to use before registration
- Destination: – Destination for call (Dialstring ) (dial, cdr\_manager)
- DestinationContext: – Destination context (cdr\_manager)
- DestinationChannel: – Destination channel (cdr\_manager)
- DestUniqueID: – UniqueID of destination (dial event)
- Direction: <type> – Audio to mute (read | write | both)
- Disposition: – Call disposition (CDR manager)
- Domain: <domain> – DNS domain
- Duration: <secs> – Duration of call (cdr\_manager)
- Dynamic: <Y | N> – Device registration supported?
- Endtime: – End time stamp of call (cdr\_manager)
- EventList: <flag> – Flag being "Start", "End", "Cancelled" or "ListObject"
- Events: <eventmask> – Eventmask filter ("on", "off", "system", "call", "log")
- Exten: – Extension (Redirect command)
- Extension: – Extension (Status)
- Family: <string> – ASTdb key family
- File: <filename> – Filename (monitor)
- Format: <format> – Format of sound file (monitor)
- From: <time> – Parking time (ParkedCall event)
- Hint: – Extension hint
- Incominglimit: – SIP Peer incoming limit
- Key: Key: – ASTdb Database key
- LastApplication: – Last application executed (cdr\_manager)
- LastCall: <num> – Last call in queue
- LastData: – Data for last application (cdr\_manager)
- Link: – (Status)
- ListItems: <number> – Number of items in Eventlist (Optionally sent in "end" packet)
- Location: – Interface (whatever that is -maybe tech/name in app\_queue )
- Loginchan: – Login channel for agent
- Logintime: <number> – Login time for agent

- Mailbox: – VM Mailbox (id@vmcontext) (mailboxstatus, mailboxcount)
- MD5SecretExist: <Y | N> – Whether secret exists in MD5 format
- Membership: <string> – "Dynamic" or "static" member in queue
- Message: <text> – Text message in ACKs, errors (explanation)
- Mix: <bool> – Boolean parameter (monitor)
- MOHSuggest: – Suggested music on hold class for peer (mohsuggest)
- NewMessages: <count> – Count of new Mailbox messages (mailboxcount)
- Newname:
- ObjectName: – Name of object in list
- OldName: – Something in Rename (channel.c)
- OldMessages: <count> – Count of old mailbox messages (mailboxcount)
- Outgoinglimit: – SIP Peer outgoing limit
- Paused: <num> – Queue member paused status
- Peer: <tech/name> – "channel" specifier
- PeerStatus: <tech/name> – Peer status code "Unregistered", "Registered", "Lagged", "Reachable"
- Penalty: <num> – Queue penalty
- Priority: – Extension priority
- Privilege: <privilege> – AMI authorization class (system, call, log, verbose, command, agent, user)
- Pickupgroup: – Pickup group for peer
- Position: <num> – Position in Queue
- Queue: – Queue name
- Reason: – "Autologoff"
- Reason: – "Chanunavail"
- Response: <response> – response code, like "200 OK" "Success", "Error", "Follows"
- Restart: – "True", "False"
- RegExpire: – SIP registry expire
- RegExpiry: – SIP registry expiry
- Reason: – Originate reason code
- Seconds: – Seconds (Status)
- Secret: <password> – Authentication secret (for login)
- SecretExist: <Y | N> – Whether secret exists
- Shutdown: – "Uncleanly", "Cleanly"
- SIP-AuthInsecure:
- SIP-FromDomain: – Peer FromDomain
- SIP-FromUser: – Peer FromUser
- SIP-NatSupport:
- SIPLastMsg:
- Source: – Source of call (dial event, cdr\_manager)
- SrcUniqueID: – UniqueID of source (dial event)
- StartTime: – Start time of call (cdr\_manager)
- State: – Channel state
- State: <1 | 0> – Mute flag
- Status: – Registration status (Registry events SIP)
- Status: – Extension status (Extensionstate)
- Status: – Peer status (if monitored) \*\* Will change name \*\* "unknown", "lagged", "ok"
- Status: <num> – Queue Status
- Status: – DND status (DNDState)
- Time: <sec> – Roundtrip time (latency)
- Timeout: – Parking timeout time
- Timeout: – Timeout for call setup (Originate)
- Timeout: <seconds> – Timeout for call
- Uniqueid: – Channel Unique ID
- Uniqueid1: – Channel 1 Unique ID (Link event)
- Uniqueid2: – Channel 2 Unique ID (Link event)
- User: – Username (SIP registry)
- UserField: – CDR userfield (cdr\_manager)
- Val: – Value to set/read in ASTdb
- Variable: – Variable AND value to set (multiple separated with | in Originate)
- Variable: <name> – For channel variables
- Value: <value> – Value to set
- VoiceMailbox: – VM Mailbox in SIPpeers
- Waiting: – Count of mailbox messages (mailboxstatus)



Please try to re-use existing headers to simplify manager message parsing in clients.\*

Read [Coding Guidelines](#) if you develop new manager commands or events.

## **Asynchronous Javascript Asterisk Manager (AJAM)**

AJAM is a new technology which allows web browsers or other HTTP enabled applications and web pages to directly access the Asterisk Manager Interface (AMI) via HTTP. Setting up your server to process AJAM involves a few steps:



## Setting up the Asterisk HTTP server

Next you'll need to enable Asterisk's [Builtin mini-HTTP server](#).

1. Uncomment the line "enabled=yes" in /etc/asterisk/http.conf to enable Asterisk's builtin micro HTTP server.
2. If you want Asterisk to actually deliver simple HTML pages, CSS, javascript, etc. you should uncomment "enablestatic=yes"
3. Adjust your "bindaddr" and "bindport" settings as appropriate for your desired accessibility
4. Adjust your "prefix" if appropriate, which must be the beginning of any URI on the server to match. The default is blank, that is no prefix and the rest of these instructions assume that value.

## Allow Manager Access via HTTP


### Configuring `manager.conf`

1. Make sure you have both "enabled = yes" and "webenabled = yes" setup in `/etc/asterisk/manager.conf`
2. You may also use "httptimeout" to set a default timeout for HTTP connections.
3. Make sure you have a manager username/secret

### Usage of AMI over HTTP

Once those configurations are complete you can reload or restart Asterisk and you should be able to point your web browser to specific URI's which will allow you to access various web functions. A complete list can be found by typing "http show status" at the Asterisk CLI.

#### Examples:

 Be sure the syntax for the URLs below is followed precisely

- `http://localhost:8088/manager?action=login&username=foo&secret=bar`

This logs you into the manager interface's "HTML" view. Once you're logged in, Asterisk stores a cookie on your browser (valid for the length of `httptimeout`) which is used to connect to the same session.

- `http://localhost:8088/rawman?action=status`

Assuming you've already logged into manager, this URI will give you a "raw" manager output for the "status" command.


- `http://localhost:8088/mxml?action=status`

This will give you the same status view but represented as AJAX data, theoretically compatible with RICO (<http://www.openrico.org>).

- `http://localhost:8088/static/ajamdemo.html`

If you have enabled static content support and have done a make install, Asterisk will serve up a demo page which presents a live, but very basic, "astman" like interface. You can login with your username/secret for manager and have a basic view of channels as well as transfer and hangup calls. It's only tested in Firefox, but could probably be made to run in other browsers as well.

A sample library (`astman.js`) is included to help ease the creation of manager HTML interfaces.

 For the demo, there is no need for **any external web server**.

## Integration with other web servers

Asterisk's micro HTTP server is **not designed to replace a general purpose web server** and it is intentionally created to provide only the minimal interfaces required. Even without the addition of an external web server, one can use Asterisk's interfaces to implement screen pops and similar tools pulling data from other web servers using iframes, div's etc. If you want to integrate CGI's, databases, PHP, etc. you will likely need to use a more traditional web server like Apache and link in your Asterisk micro HTTP server with something like this:

```
ProxyPass /asterisk http://localhost:8088/asterisk
```

## Asterisk Manager Interface (AMI) Changes

Container page for AMI related content.

## AMI 1.1 Changes

### Changes to manager version 1.1:

#### SYNTAX CLEANUPS

- Response: headers are now either
  - "Success" - Action OK, this message contains response
  - "Error" - Action failed, reason in Message: header
  - "Follows" - Action OK, response follows in following Events.
- Manager version changed to 1.1

#### CHANGED EVENTS AND ACTIONS

- The Hold/Unhold events
  - Both are now "Hold" events
  - For hold, there's a "Status: On" header, for unhold, status is off
  - Modules chan\_sip/chan\_iax2
- The Ping Action
  - Now use Response: success
  - New header "Ping: pong" :-)
- The Events action
  - Now use Response: Success
  - The new status is reported as "Events: On" or "Events: Off"
- The JabberSend action
  - The Response: header is now the first header in the response
  - now sends "Response: Error" instead of "Failure"
- Newstate and Newchannel events
  - these have changed headers
    - "State" -> ChannelStateDesc Text based channel state
    - > ChannelState Numeric channel state
  - The events does not send "<unknown>" for unknown caller IDs just an empty field
- Newchannel event
  - Now includes "AccountCode"
- Newstate event
  - Now has "CalleridNum" for numeric caller id, like Newchannel
  - The event does not send "<unknown>" for unknown caller IDs just an empty field
- Newexten and VarSet events
  - Now are part of the new Dialplan privilege class, instead of the Call class
- Dial event
  - Event Dial has new headers, to comply with other events
  - Source -> Channel Channel name (caller)
  - SrcUniqueID -> UniqueID Uniqueid
  - (new) -> Dialstring Dialstring in app data
- Link and Unlink events
  - The "Link" and "Unlink" bridge events in channel.c are now renamed to "Bridge"
  - The link state is in the bridgestate: header as "Link" or "Unlink"
  - For channel.c bridges, "Bridgetype: core" is added. This opens up for bridge events in rtp.c
  - The RTP channel also reports Bridge: events with bridgetypes
    - rtp-native RTP native bridge
    - rtp-direct RTP peer-2-peer bridge (NAT support only)
    - rtp-remote Remote (re-invite) bridge. (Not reported yet)
- The "Rename" manager event has a renamed header, to use the same terminology for the current channel as other events
  - Oldname -> Channel
- The "NewCallerID" manager event has a renamed header
  - CallerID -> CallerIDnum
  - The event does not send "<unknown>" for unknown caller IDs just an empty field
- Reload event
  - The "Reload" event sent at manager reload now has a new header and is now implemented in more modules than manager to alert a reload. For channels, there's a CHANNELRELOAD event to use.
    - (new) -> Module: manager | CDR | DNSmgr | RTP | ENUM
    - (new) -> Status: enabled | disabled
  - To support reload events from other modules too
    - cdr module added
- Status action replies (Event: Status)  
Header changes
  - link -> BridgedChannel
  - Account -> AccountCode
  - (new) -> BridgedUniqueid

- StatusComplete Event  
New header  
- (new) -> Items Number of channels reported
  
- The ExtensionStatus manager command now has a "StatusDesc" field with text description of the state
- The Registry and Peerstatus events in chan\_sip and chan\_iax now use "ChannelType" instead of "ChannelDriver"
- The Response to Action: IAXpeers now have a Response: Success header
- The MeetmeJoin now has caller ID name and Caller ID number fields (like MeetMeLeave)
  
- Action DAHDIShowChannels  
Header changes  
- Channel: -> DAHDICchannel  
For active channels, the Channel: and Uniqueid: headers are added  
You can now add a "DAHDICchannel: " argument to DAHDIShowchannels actions  
to only get information about one channel.
  
- Event DAHDIShowChannelsComplete  
New header  
- (new) -> Items: Reports number of channels reported
  
- Action VoicemailUsersList  
Added new headers for SayEnvelope, SayCID, AttachMessage, CanReview  
and CallOperator voicemail configuration settings.
  
- Action Originate  
Now requires the new Originate privilege.  
If you call out to a subshell in Originate with the Application parameter,  
you now also need the System privilege.
  
- Event QueueEntry now also returns the Uniqueid field like other events from app\_queue.
  
- Action IAXpeerlist  
Now includes if the IAX link is a trunk or not
  
- Action IAXpeers  
Now includes if the IAX link is a trunk or not
  
- Action Ping  
Response now includes a timestamp
  
- Action SIPshowpeer  
Response now includes the configured parkinglot

```
- Action: SKINNYshowline
Response now includes the configured parkinglot
```

## NEW ACTIONS

```
- Action: DataGet
Modules: data.c
Purpose:
  To be able to retrieve the asterisk data tree.
Variables:
  ActionID: <id>          Action ID for this transaction. Will be returned.
  Path: <data path>      The path to the callback node to retrieve.
  Filter: <filter>       Which nodes to retrieve.
  Search: <search>       Search condition.

- Action: IAXregistry
Modules: chan_iax2
Purpose:
  To list all IAX2 peers in the IAX registry with their registration status.
Variables:
  ActionID: <id> Action ID for this transaction. Will be returned.

- Action: ModuleLoad
Modules: loader.c
Purpose:
  To be able to unload, reload and unload modules from AMI.
Variables:
  ActionID: <id>          Action ID for this transaction. Will be returned.
  Module: <name>          Asterisk module name (including .so extension)
                        or subsystem identifier:
  cdr, enum, dnsmgr, extconfig, manager, rtp, http
  LoadType: load | unload | reload
                        The operation to be done on module
  If no module is specified for a reload loadtype, all modules are reloaded

- Action: ModuleCheck
Modules: loader.c
Purpose:
  To check version of a module - if it's loaded
Variables:
  ActionID: <id>          Action ID for this transaction. Will be returned.
  Module: <name>          Asterisk module name (not including extension)
Returns:
  If module is loaded, returns version number of the module

Note: This will have to change. I don't like sending Response: failure
on both command not found (trying this command in earlier versions of
Asterisk) and module not found.
Also, check if other manager actions behave that way.

- Action: QueueSummary
Modules: app_queue
Purpose:
  To request that the manager send a QueueSummary event (see the NEW EVENTS
  section for more details).
Variables:
  ActionID: <id> Action ID for this transaction. Will be returned.
  Queue: <name> Queue for which the summary is desired

- Action: QueuePenalty
Modules: app_queue
Purpose:
  To change the penalty of a queue member from AMI
Variables:
  Interface: <tech/name> The interface of the member whose penalty you wish to change
  Penalty: <number> The new penalty for the member. Must be nonnegative.
  Queue: <name> If specified, only set the penalty for the member for this queue;
  Otherwise, set the penalty for the member in all queues to which
  he belongs.

- Action: QueueRule
Modules: app_queue
Purpose:
  To list queue rules defined in queuerules.conf
Variables:
  ActionID: <id>          Action ID for this transaction. Will be returned.
  Rule: <name> The name of the rule whose contents you wish to list. If this variable
  is not present, all rules in queuerules.conf will be listed.

- Action: Atxfer
Modules: none
Purpose:
  Initiate an attended transfer
Variables:
  Channel: The transferer channel's name
  Exten: The extension to transfer to
  Priority: The priority to transfer to
```

```

Context: The context to transfer to

- Action: SipShowRegistry
Modules: chan_sip
Purpose:
  To request that the manager send a list of RegistryEntry events.
Variables:
  ActionId: <id> Action ID for this transaction. Will be returned.

- Action: QueueReload
Modules: app_queue
Purpose:
  To reload queue rules, a queue's members, a queue's parameters, or all of the aforementioned
Variable:
  ActionID: <id>
  Queue: <name> The name of the queue to take action on.
           If no queue name is specified, then all queues are affected
  Rules: <yes or no> Whether to reload queuerules.conf
  Members: <yes or no> Whether to reload the queue's members
  Parameters: <yes or no> Whether to reload the other queue options

- Action: QueueReset
Modules: app_queue
Purpose:
  Reset the statistics for a queue
Variables:
  ActionID: <id>
  Queue: <name> The name of the queue on which to reset statistics

- Action: SKINNYdevices
Modules: chan_skinny
Purpose:
  To list all SKINNY devices configured.
Variables:
  ActionId: <id> Action ID for this transaction. Will be returned.

- Action: SKINNYlines
Modules: chan_skinny
Purpose:
  To list all SKINNY lines configured.
Variables:
  ActionId: <id> Action ID for this transaction. Will be returned.

- Action SKINNYshowdevice
Modules: chan_skinny
Purpose:
  To list the information about a specific SKINNY device.
Variables:
  Device: <device> Device to show information about.

- Action SKINNYshowline
Modules: chan_skinny
Purpose:
  To list the information about a specific SKINNY line.
Variables:
  Line: <line> Line to show information about.

- Action: CoreSettings
Modules: manager.c
Purpose: To report core settings, like AMI and Asterisk version,
maxcalls and maxload settings.
  * Integrated in SVN trunk as of May 4th, 2007
Example:
Response: Success
ActionID: 1681692777
AMIVersion: 1.1
AsteriskVersion: SVN-oej-moremanager-r61756M
SystemName: EDVINA-node-a
CoreMaxCalls: 120
CoreMaxLoadAvg: 0.000000
CoreRunUser: edvina
CoreRunGroup: edvina

- Action: CoreStatus
Modules: manager.c
Purpose: To report current PBX core status flags, like
number of concurrent calls, startup and reload time.
  * Integrated in SVN trunk as of May 4th, 2007
Example:
Response: Success
ActionID: 1649760492
CoreStartupTime: 22:35:17
CoreReloadTime: 22:35:17
CoreCurrentCalls: 20

- Action: MixMonitorMute
Modules: app_mixmonitor.c
Purpose:
  Mute / unMute a Mixmonitor recording.

```



Variables:

ActionId: <id> Action ID for this transaction. Will be returned.

Channel: the channel MixMonitor is running on

Direction: Which part of the recording to mute: read, write or both (from

```
channel, to channel or both channels).  
State: Turn mute on or off : 1 to turn on, 0 to turn off.
```

## NEW EVENTS

```
- Event: FullyBooted  
Modules: loader.c  
Purpose:  
It is handy to have a single event notification for when all Asterisk  
modules have been loaded--especially for situations like running  
automated tests. This event will fire 1) immediately upon all modules  
loading or 2) upon connection to the AMI interface if the modules have  
already finished loading before the connection was made. This ensures  
that a user will never miss getting a FullyBooted event. In vary rare  
circumstances, it might be possible to get two copies of the message  
if the AMI connection is made right as the modules finish loading.  
Example:  
Event: FullyBooted  
Privilege: system,all  
Status: Fully Booted  
  
- Event: Transfer  
Modules: res_features, chan_sip  
Purpose:  
Inform about call transfer, linking transferer with transfer target  
You should be able to trace the call flow with this missing piece  
of information. If it works out well, the "Transfer" event should  
be followed by a "Bridge" event  
The transfermethod: header informs if this is a pbx core transfer  
or something done on channel driver level. For SIP, check the example:  
Example:  
  
Event: Transfer  
Privilege: call,all  
TransferMethod: SIP  
TransferType: Blind  
Channel: SIP/device1-01849800  
SIP-Callid: 091386f505842c87016c4d93195ec67d@127.0.0.1  
TargetChannel: SIP/device2-01841200  
TransferExten: 100  
TransferContext: default  
  
- Event: ChannelUpdate  
Modules: chan_sip.c, chan_iax2.c  
Purpose:  
Updates channel information with ID of PVT in channel driver, to  
be able to link events on channel driver level.  
* Integrated in SVN trunk as of May 4th, 2007  
  
Example:  
  
Event: ChannelUpdate  
Privilege: system,all  
Uniqueid: 1177271625.27  
Channel: SIP/olle-01843c00  
Channeltype: SIP  
SIPcallid: NTQzYWFiOWM4NmE0MWRkZjExMzU2YzQ3OWQwNzg3ZmI.  
SIPfullcontact: sip:olle@127.0.0.1:49054  
  
- Event: NewAccountCode  
Modules: cdr.c  
Purpose: To report a change in account code for a live channel  
Example:  
Event: NewAccountCode  
Privilege: call,all  
Channel: SIP/olle-01844600  
Uniqueid: 1177530895.2  
AccountCode: Stinas account 1234848484  
OldAccountCode: OllesAccount 12345  
  
- Event: ModuleLoadReport  
Modules: loader.c  
Purpose: To report that module loading is complete. Some aggressive  
clients connect very quickly to AMI and needs to know when  
all manager events embedded in modules are loaded  
Also, if this does not happen, something is seriously wrong.  
This could happen to chan_sip and other modules using DNS.  
Example:  
Event: ModuleLoad  
ModuleLoadStatus: Done  
ModuleSelection: All  
ModuleCount: 24  
  
- Event: QueueSummary  
Modules: app_queue  
Purpose: To report a summary of queue information. This event is generated by  
issuing a QueueSummary AMI action.
```

Example:

```
Event: QueueSummary
Queue: Sales
LoggedIn: 12
Available: 5
Callers: 10
HoldTime: 47
```

If an actionID was specified for the QueueSummary action, it will be appended as the last line of the QueueSummary event.

- Event: AgentRingNoAnswer

Modules: app\_queue

Purpose: Reports when a queue member was rung but there was no answer.

Example:

```
Event: AgentRingNoAnswer
Queue: Support
Uniqueid: 1177530895.2
Channel: SIP/1000-53aee458
Member: SIP/1000
MemberName: Thaddeus McClintock
Ringtime: 10
```

- Event: RegistryEntry

Modules: chan\_sip

Purpose: Reports the state of the SIP registrations. This event is generated by issuing a QueueSummary AMI action.

The RegistrationTime header is expressed as epoch.

Example:

```
Event: RegistryEntry
Host: sip.myvoipprovider.com
Port: 5060
Username: guestuser
Refresh: 105
State: Registered
RegistrationTime: 1219161830
```

If an actionID was specified for the SipShowRegistry action, it will be appended as the last line of the RegistrationsComplete event.

- Event: ChanSpyStart

Modules: app\_chanspy

Purpose: Reports when an active channel starts to be monitored by someone.

Example:

```
Event: ChanSpyStart
SpyerChannel: SIP/4321-13bba124
SpyeChannel: SIP/1234-56ecc098
```

- Event: ChanSpyStop

Modules: app\_chanspy

Purpose: Reports when an active channel stops to be monitored by someone.

Example:

Event: ChanSpyStop  
SpyeeChannel: SIP/1234-56ecc098

TODO

...

## AMI Libraries and Frameworks



This page is under construction!

Name	Language	Website	Protocols
Asterisk-Java	Java	<a href="https://blogs.reucon.com/asterisk-java/">https://blogs.reucon.com/asterisk-java/</a>	AMI/FastAGI
StarPy	Python+Twisted	<a href="https://github.com/asterisk/starpy">https://github.com/asterisk/starpy</a>	AMI/FastAGI
Panoramisk	Python+AsyncIO	<a href="https://github.com/gawel/panoramisk">https://github.com/gawel/panoramisk</a>	AMI/FastAGI
PAMI	PHP	<a href="https://github.com/marcelog/PAMI">https://github.com/marcelog/PAMI</a>	AMI
Pyst2	Python	<a href="https://github.com/rdegges/pyst2">https://github.com/rdegges/pyst2</a>	AMI/AGI
Adhearsion	Ruby	<a href="http://www.adhearsion.com/">http://www.adhearsion.com/</a>	AMI/FastAGI
node-asterisk	Node.js	<a href="https://github.com/danjenkins/node-asterisk-ami">https://github.com/danjenkins/node-asterisk-ami</a>	AMI

# Asterisk REST Interface (ARI)

## The Evolution of Asterisk APIs

When Asterisk was first created back in 1999, its design was focussed on being a stand-alone Private Branch eXchange (PBX) that you could configure via static `.conf` files. Control of the calls that passed through it was done through a special `.conf` file, `extensions.conf`, known as the "dialplan". The dialplan script told Asterisk which applications to execute on the call, and made logical decisions based on what the users did through their phones. This model worked well for a long time - it was certainly more flexible than what existed at the time, and the plethora of dialplan applications provided a large suite of functionality.

These dialplan applications, however, were - and still are - written in C. Because the applications act directly on the raw primitives in Asterisk, they are incredibly powerful. They have access to the channels, media, bridges, endpoints, and all other objects that Asterisk uses to make phones communicate. However, while powerful, there are times when a business use case is not met by the existing suite of applications. In the past, if the functionality you needed wasn't met by the dialplan application, you really only had one solution: write a patch in C - possibly adding a parameter to the application to tweak the behaviour - and submit it to the project. If you could not write the feature in C, you were, unfortunately, stuck.

## Enter AMI and AGI

Not long into the project, two application programming interfaces (APIs) were added to Asterisk: the Asterisk Gateway Interface (AGI) and the Asterisk Manager Interface (AMI). These interfaces have distinct purposes:

1. AGI is analogous to CGI in Apache. AGI provides an interface between the Asterisk dialplan and an external program that wants to manipulate a channel in the dialplan. In general, the interface is synchronous - actions taken on a channel from an AGI block and do not return until the action is completed.
2. AMI provides a mechanism to control where channels execute in the dialplan. Unlike AGI, AMI is an asynchronous, event driven interface. For the most part, AMI does not provide mechanisms to control channel execution - rather, it provides information about the state of the channels and controls about where the channels are executing.

Both of these interfaces are powerful and opened up a wide range of integration possibilities. Using AGI, remote dialplan execution could be enabled, which allowed developers to integrate Asterisk with PHP, Python, Java, and other applications. Using AMI, the state of Asterisk could be displayed, calls initiated, and the location of channels controlled. Using both APIs together, complex applications using Asterisk as the engine *could* be developed.

On This Page
<ul style="list-style-type: none"><li>• <a href="#">The Evolution of Asterisk APIs</a><ul style="list-style-type: none"><li>• <a href="#">Enter AMI and AGI</a></li><li>• <a href="#">ARI: An Interface for Communications Applications</a></li></ul></li><li>• <a href="#">ARI Fundamentals</a><ul style="list-style-type: none"><li>• <a href="#">What is REST?</a></li><li>• <a href="#">What is a WebSocket?</a></li><li>• <a href="#">What is Stasis?</a></li></ul></li><li>• <a href="#">Diving Deeper</a><ul style="list-style-type: none"><li>• <a href="#">Where to get the examples</a></li><li>• <a href="#">ARI Libraries</a></li></ul></li><li>• <a href="#">Recommended Practices</a><ul style="list-style-type: none"><li>• <a href="#">Don't access ARI directly from a web page</a></li><li>• <a href="#">Use an abstraction layer</a></li></ul></li></ul>

ARI In More Detail
<ul style="list-style-type: none"><li>• <a href="#">Getting Started with ARI</a></li><li>• <a href="#">Asterisk Configuration for ARI</a></li><li>• <a href="#">Introduction to ARI and Channels</a></li><li>• <a href="#">Introduction to ARI and Bridges</a></li><li>• <a href="#">Introduction to ARI and Media Manipulation</a></li><li>• <a href="#">The Asterisk Resource</a></li></ul>

However, there are some drawbacks to using AMI and AGI to create custom communication applications:

1. AGI is synchronous and blocks the thread servicing the AGI when an Asterisk action is taken on the channel. When creating a communications application, you will often want to respond to changes in the channel (DTMF, channel state, etc.); this is difficult to do with AGI by itself. Coordinating with AMI events can be challenging.
2. The dialplan can be limiting. Even with AMI and AGI, your fundamental operations are limited to what can be executed on a channel. While powerful, there are other primitives in Asterisk that are not available through those APIs: bridges, endpoints, device state, message waiting indications, and the actual media on the channels themselves. Controlling those through AMI and AGI can be difficult, and can often involve complex dialplan manipulation to achieve.
3. Finally, both AMI and AGI were created early in the Asterisk project, and are products of their time. While both are powerful interfaces, technologies that are used today were not in heavy use at the time. Concepts such as SOAP, XML/JSON-RPC, and REST were not in

heavy use. As such, newer APIs can be more intuitive and easier to adopt, leading to faster development for users of Asterisk.

And so, before Asterisk 12, if you wanted your own custom communication application, you could:

- Write an Asterisk module in C, **or**
- Write a custom application using both AGI and AMI, performing some herculean effort to marry the two APIs together (as well as some dialplan trickery)

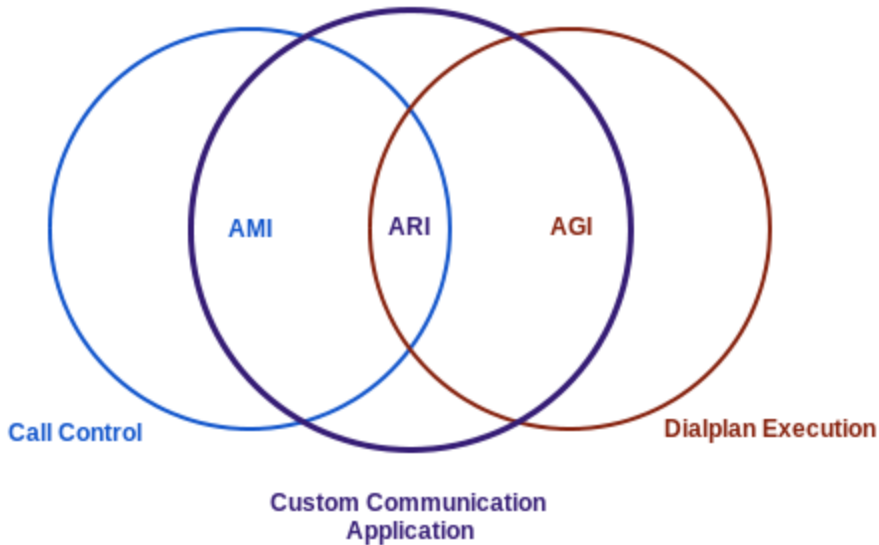
## ARI: An Interface for Communications Applications

The Asterisk RESTful Interface (ARI) was created to address these concerns. While AMI is good at call control and AGI is good at allowing a remote process to execute dialplan applications, neither of these APIs was designed to let a developer build their own custom communications application. ARI is an asynchronous API that allows developers to build communications applications by exposing the raw primitive objects in Asterisk - channels, bridges, endpoints, media, etc. - through an intuitive REST interface. The state of the objects being controlled by the user are conveyed via JSON events over a WebSocket.

These resources were traditionally the purview of Asterisk's C modules. By handing control of these resources over to all developers - regardless of their language choice - Asterisk becomes an engine of communication, with the business logic of how things should communicate deferred to the application using Asterisk.

***ARI is not about telling a channel to execute the VoiceMail dialplan application or redirecting a channel in the dialplan to VoiceMail.***

***It is about letting you build your own VoiceMail application.***



## ARI Fundamentals

ARI consists of three different pieces that are - for all intents and purposes - interrelated and used together. They are:

1. A **RESTful** interface that a client uses to control resources in Asterisk.
2. A **WebSocket** that conveys events in **JSON** about the resources in Asterisk to the client.
3. The **Stasis** dialplan application that hands over control of a channel from Asterisk to the client.

All three pieces work together, allowing a developer to manipulate and control the fundamental resources in Asterisk and build their own communications application.



### Developer Documentation

You can find some historical documents [on the wiki](#) about the development and architecture of ARI.

## What is REST?

**Representational State Transfer (REST)** is a software architectural style. It has several characteristics:

- Communication is performed using a client-server model.
- The communication is stateless. Servers do not store client state between requests.
- Connections are layered, allowing for intermediaries to assist in routing and load balancing.
- A uniform interface. Resources are identified in the requests, messages are self-descriptive, etc.

ARI *does not* strictly conform to a REST API. Asterisk, as a stand-alone application, has state that may change outside of a client request through ARI. For example, a SIP phone may be hung up, and Asterisk will hang up the channel - even though a client through ARI did not tell Asterisk to hang up the SIP phone. Asterisk lives in an asynchronous, state-ful world: hence, ARI is *RESTful*. It attempts to follow the tenants of REST as best as it can, without getting bogged down in philosophical constraints.

## What is a WebSocket?

**WebSockets** are a relatively new protocol standard (RFC 6455) that enable two-way communication between a client and server. The protocol's primary purpose is to provide a mechanism for browser-based applications that need two-way communication with servers, without relying on HTTP long polling or other, non-standard, mechanisms.

In the case of ARI, a WebSocket connection is used to pass asynchronous events from Asterisk to the client. These events are related to the RESTful interface, but are technically independent of it. They allow Asterisk to inform the client of changes in resource state that may occur because of and in conjunction with the changes made by the client through ARI.

## What is Stasis?

**Stasis** is a dialplan application in Asterisk. It is the mechanism that Asterisk uses to hand control of a channel over from the dialplan - which is the traditional way in which channels are controlled - to ARI and the client. Generally, ARI applications manipulate channels in the Stasis dialplan application, as well as other resources in Asterisk. Channels not in a Stasis dialplan application generally cannot be manipulated by ARI - the purpose of ARI, after all, is to build your own dialplan application, not manipulate an existing one.

## Diving Deeper

This space has a number of pages that explore different resources available to you in ARI and examples of what you can build with them. Generally, the examples assume the following:

- That you have some phone registered to Asterisk, typically using `chan_pjsip` or `chan_sip`
- That you have some basic knowledge of configuring Asterisk
- A basic knowledge of Python, JavaScript, or some other higher level programming language (or a willingness to learn!)

Most of the examples will not directly construct the HTTP REST calls, as a number of very useful libraries have been written to encapsulate those mechanics. These libraries are listed below.

## Where to get the examples

All of the examples on the pages below this one are available on [github](#). Check them out!

## ARI Libraries

Library	Language	Website
<code>ari-py</code>	Python	<a href="https://github.com/asterisk/ari-py">https://github.com/asterisk/ari-py</a>
<code>AsterNET.ARI</code>	C# / .NET	<a href="https://github.com/skrusty/AsterNET.ARI">https://github.com/skrusty/AsterNET.ARI</a>
<code>ari4java</code>	Java	<a href="https://github.com/l3nz/ari4java">https://github.com/l3nz/ari4java</a>
<code>node-ari-client</code>	JavaScript (node)	<a href="https://github.com/asterisk/node-ari-client">https://github.com/asterisk/node-ari-client</a>
<code>phpari</code>	PHP	<a href="http://www.phpari.org/">http://www.phpari.org/</a>
<code>asterisk-ari-client</code>	Ruby	<a href="https://github.com/svoboda-jan/asterisk-ari">https://github.com/svoboda-jan/asterisk-ari</a>
<code>Phparia</code>	PHP	<a href="http://phparia.org/">http://phparia.org/</a>

## Recommended Practices

### Don't access ARI directly from a web page

It's very convenient to use ARI directly from a web page for development, such as using Swagger-UI, or even abusing the [WebSocket echo demo](#) to get at the ARI WebSocket.

But, *please*, do not do this in your production applications. This would be akin to accessing your database directly from a web page. You need to hide Asterisk behind your own application server, where you can handle security, logging, multi-tenancy and other concerns that really don't belong in a communications engine.

### Use an abstraction layer



One of the beautiful things about ARI is that it's so easy to just bang out a request. But what's good for development isn't necessarily what's good for production.

Please don't spread lots of direct HTTP calls throughout your application. There are cross-cutting concerns with accessing the API that you'll want to deal with in a central location. Today, the only concern is authentication. But as the API evolves, other concerns (such as versioning) will also be important.

Note that the abstraction layer doesn't (and shouldn't) be complicated. Your client side API can even be something as simple wrapper around GET, POST and DELETE that addresses the cross-cutting concerns. The Asterisk TestSuite has a very simple abstraction library that can be used like this:

```
ari = ARI('localhost', ('username', 'password'))

# Hang up all channels
channels = ari.get('channels')
for channel in channels:
    ari.delete('channels', channel['id'])
```

In other words: **use one of the aforementioned libraries or write your own!**

# Getting Started with ARI

## Overview

Asterisk 12 introduces the [Asterisk REST Interface](#), a set of RESTful APIs for building Asterisk based applications. This article will walk you through getting ARI up and running.

There are three main components to building an ARI application.

The first, obviously, is **the RESTful API** itself. The API is documented using [Swagger](#), a lightweight specification for documenting RESTful APIs. The Swagger API docs are used to generate validations and boilerplate in Asterisk itself, along with [static wiki documentation](#), and interactive documentation using [Swagger-UI](#).

Then, Asterisk needs to send asynchronous events to the application (new channel, channel left a bridge, channel hung up, etc). This is done using a **Web Socket on /ari/events**. Events are sent as JSON messages, and are documented on the [REST Data Models page](#). (See the list of subtypes for the [Message data model](#).)

Finally, connecting the dialplan to your application is the [Stasis\(\) dialplan application](#). From within the dialplan, you can send a channel to [Stasis\(\)](#), specifying the name of the external application, along with optional arguments to pass along to the application.

On This Page
<ul style="list-style-type: none"><li>• <a href="#">Overview</a></li><li>• <a href="#">Example: ARI Hello World!</a><ul style="list-style-type: none"><li>• <a href="#">Getting wscat</a></li><li>• <a href="#">Getting curl</a></li><li>• <a href="#">Configuring Asterisk</a></li><li>• <a href="#">Hello World!</a></li></ul></li></ul>

## Example: ARI Hello World!

In this example, we will:

- Configure Asterisk to enable ARI
- Send a channel into Stasis
- And playback "Hello World" to the channel

This example will **not** cover:

1. Installing Asterisk. We'll assume you have Asterisk 12 or later installed and running.
2. Configuring a SIP device in Asterisk. For the purposes of this example, we are going to assume you have a SIP softphone or hardphone registered to Asterisk, using either `chan_sip` or `chan_pjsip`.

## Getting wscat

ARI needs a WebSocket connection to receive events. For the sake of this example, we're going to use [wscat](#), an incredibly handy command line utility similar to netcat but based on a node.js websocket library. If you don't have wscat:

1. If you don't have it already, **install** npm

```
$ apt-get install npm
```

2. **Install** the `ws` node package:

```
$ npm install -g wscat
```



Some distributions repos (e.g. Ubuntu) may have older versions of nodejs and npm that will throw a wrench in your install of the `ws` package. You'll have to install a newer version from another repo or via source.

[Installing Nodejs via packages](#)

[Installing npm in a variety of ways](#)

## Getting curl

In order to control a channel in the Stasis dialplan application through ARI, we also need an HTTP client. For the sake of this example, we'll use `curl`:

```
$ apt-get install curl
```

## Configuring Asterisk

1. Enable the Asterisk HTTP service in `http.conf`:

```
http.conf

[general]
enabled = yes
bindaddr = 0.0.0.0
```

2. Configure an ARI user in `ari.conf`:

```
ari.conf

[general]
enabled = yes
pretty = yes

[asterisk]
type = user
read_only = no
password = asterisk
```



### This is just a demo

Please use a more secure account user and password for production applications. Outside of examples and demos, asterisk/asterisk is a terrible, horrible, no-good choice...

3. Create a dialplan extension for your Stasis application. Here, we're choosing extension 1000 in context `default` - if your SIP phone is configured for a different context, adjust accordingly.

```
extensions.conf

[default]

exten => 1000,1,NoOp()
same =>      n,Answer()
same =>      n,Stasis(hello-world)
same =>      n,Hangup()
```

## Hello World!

1. Connect to Asterisk using `wscat`:

```
$ wscat -c
"ws://localhost:8088/ari/events?api_key=asterisk:asterisk&app=hello-world"
connected (press CTRL+C to quit)
>
```

In Asterisk, we should see a new WebSocket connection and a message telling us that our Stasis application has been created:

```
== WebSocket connection from '127.0.0.1:37872' for protocol '' accepted using
version '13'
Creating Stasis app 'hello-world'
```

2. From your SIP device, dial extension 1000:

```
-- Executing [1000@default:1] NoOp("PJSIP/1000-00000001", "") in new stack
-- Executing [1000@default:2] Answer("PJSIP/1000-00000001", "") in new stack
-- PJSIP/1000-00000001 answered
-- Executing [1000@default:3] Stasis("PJSIP/1000-00000001", "hello-world") in
new stack
```

In `wscat`, we should see the `StasisStart` event, indicating that a channel has entered into our Stasis application:

```
< {
  "application": "hello-world",
  "type": "StasisStart",
  "timestamp": "2014-05-20T13:15:27.131-0500",
  "args": [],
  "channel": {
    "id": "1400609726.3",
    "state": "Up",
    "name": "PJSIP/1000-00000001",
    "caller": {
      "name": "",
      "number": ""
    },
    "connected": {
      "name": "",
      "number": ""
    },
    "accountcode": "",
    "dialplan": {
      "context": "default",
      "exten": "1000",
      "priority": 3
    },
    "creationtime": "2014-05-20T13:15:26.628-0500"
  }
}
>
```

- Using `curl`, tell Asterisk to playback `hello-world`. Note that the identifier of the channel in the `channels` resource **must** match the channel id passed back in the `StasisStart` event:

```
$ curl -v -u asterisk:asterisk -X POST
"http://localhost:8088/ari/channels/1400609726.3/play?media=sound:hello-world"
```

The response to our HTTP request will tell us whether or not the request succeeded or failed (in our case, a success will queue the playback onto the channel), as well as return in JSON the Playback resource that was created for the operation:

```

* About to connect() to localhost port 8088 (#0)
* Trying 127.0.0.1... connected
* Server auth using Basic with user 'asterisk'
> POST /ari/channels/1400609726.3/play?media=sound:hello-world HTTP/1.1
> Authorization: Basic YXN0ZXJpc2s6c2VjcmV0
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1
zlib/1.2.3.4 libidn/1.23 librtmp/2.3
> Host: localhost:8088
> Accept: */*
>
< HTTP/1.1 201 Created
< Server: Asterisk/SVN-branch-12-r414137M
< Date: Tue, 20 May 2014 18:25:15 GMT
< Connection: close
< Cache-Control: no-cache, no-store
< Content-Length: 146
< Location: /playback/9567ea46-440f-41be-a044-6ecc8100730a
< Content-type: application/json
<
* Closing connection #0
{"id": "9567ea46-440f-41be-a044-6ecc8100730a",
 "media_uri": "sound:hello-world",
 "target_uri": "channel:1400609726.3",
 "language": "en",
 "state": "queued"}
$

```

In Asterisk, the sound file will be played back to the channel:

```
-- <PJSIP/1000-00000001> Playing 'hello-world.gsm' (language 'en')
```

And in our `wscat` WebSocket connection, we'll be informed of the start of the playback, as well as it finishing:

```

< {"application": "hello-world",
   "type": "PlaybackStarted",
   "playback": {
     "id": "9567ea46-440f-41be-a044-6ecc8100730a",
     "media_uri": "sound:hello-world",
     "target_uri": "channel:1400609726.3",
     "language": "en",
     "state": "playing"}
  }

< {"application": "hello-world",
   "type": "PlaybackFinished",
   "playback": {
     "id": "9567ea46-440f-41be-a044-6ecc8100730a",
     "media_uri": "sound:hello-world",
     "target_uri": "channel:1400609726.3",
     "language": "en",
     "state": "done"}
  }

```

4. Hang up the phone! This will cause the channel in Asterisk to be hung up, and the channel will leave the Stasis application, notifying the client via a `StasisEnd` event:

```
< {"application":"hello-world",
  "type":"StasisEnd",
  "timestamp":"2014-05-20T13:30:01.852-0500",
  "channel":{
    "id":"1400609726.3",
    "state":"Up",
    "name":"PJSIP/1000-00000001",
    "caller":{
      "name":"",
      "number":""},
    "connected":{
      "name":"",
      "number":""},
    "accountcode":"",
    "dialplan":{
      "context":"default",
      "exten":"1000",
      "priority":3},
    "creationtime":"2014-05-20T13:15:26.628-0500"}
}
```

## Using Swagger to Drive ARI

### Using Swagger-UI

The REST API that makes up ARI is documented using [Swagger](#), a lightweight specification for documenting RESTful API's. The Swagger API docs are used to generate validations and boilerplate in Asterisk itself, along with [static wiki documentation](#), and interactive documentation using [Swagger-UI](#).

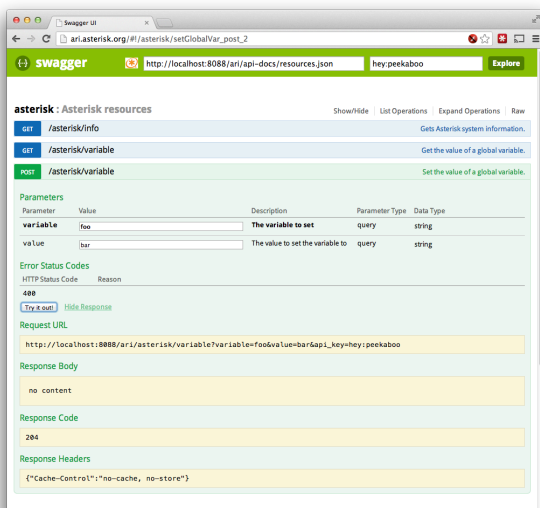
[Swagger-UI](#) is a pure HTML+JavaScript application which can download Swagger api-docs, and generate an interactive web page which allows you to view resources, their operations, and submit API requests directly from the documentation. A fork of Swagger-UI is hosted on [ari.asterisk.org](#), which enables DELETE operations (which are disabled by default in Swagger-UI), and sets the default URL to what it would be running Asterisk on your local system.

In order to access ARI, you have to populate the `api_key` field with a `username:password` configured in `ari.conf`. You should also set `allowed_origins` in `ari.conf` to allow the site hosting Swagger-UI to access ARI.

```
ari.conf

[general]
enabled=yes
;pretty=yes      ; we don't need pretty-printing of the JSON responses in this
                  ; example, but you might if you use curl a lot.
;
; In this example, we are going to use the version of Swagger-UI that is hosted
; at ari.asterisk.org. In order to get past CORS restrictions in the browser,
; That origin needs to be added to the allowed_origins list.
;
allowed_origins=http://ari.asterisk.org

[hey]
type=user
password=peekaboo
;read_only=no   ; Set to yes for read-only applications
```



# Asterisk Configuration for ARI

## Overview

ARI has a number of parts to it - the HTTP server in Asterisk servicing requests, the dialplan application handing control of channels over to a connected client, and the websocket sharing state in Asterisk with the external application. This page provides the configuration files in Asterisk that can be altered to suit deployment considerations.



This page does not include all of the configuration options available to a system administrator. It does cover some of the basics that you might be interested in when setting up your Asterisk system for ARI.

## Asterisk Configuration Options for ARI

### HTTP Server

The HTTP server in Asterisk is configured via `http.conf`. Note that this does not describe all of the options available via `http.conf` - rather, it lists the most useful ones for ARI.

#### On This Page

- Overview
- Asterisk Configuration Options for ARI
  - HTTP Server
  - ARI Configuration
- Configuring the Dialplan for ARI

Section	Parameter	Type	Default	Description
general				
	<code>enabled</code>	Boolean	False	Enable the HTTP server. <b>The HTTP server in Asterisk is disabled by default.</b> Unless it is enabled, ARI will not function!
	<code>bindaddr</code>	IP Address		The IP address to bind the HTTP server to. This can either be an explicit local address, or <code>0.0.0.0</code> to bind to all available interfaces.
	<code>bindport</code>	Port	8088	The port to bind the HTTP server to. Client making HTTP requests should specify 8088 as the port to send the request to.
	<code>prefix</code>	String		A prefix to require for all requests. If specified, requests must begin with the specified prefix.
	<code>tlsenable</code>	Boolean	False	Enable HTTPS
	<code>tlsbindaddr</code>	IP Address/Port		The IP address and port to bind the HTTPS server to. This should be an IP address and port, e.g., <code>0.0.0.0:8089</code>
	<code>tlscertfile</code>	Path		The full path to the certificate file to use. Asterisk only supports the <code>.pem</code> format.
	<code>tlsprivatekey</code>	Path		The full path to the private key file. Asterisk only supports the <code>.pem</code> format. If this is not specified, the certificate specified in <code>tlscertfile</code> will be searched for the private key.

### Example `http.conf`

#### `http.conf`

```
[general]
enabled = yes
bindaddr = 0.0.0.0
bindport = 8088
```



#### Use TLS!

It is **highly** recommended that you encrypt your HTTP signalling with TLS, and use secure WebSockets (WSS) for your events. This requires configuring the TLS information in `http.conf`, and establishing secure websocket/secure HTTP connections from your ARI application.



## ARI Configuration

ARI users and properties are configured via `ari.conf`. Note that all options may not be listed here; this listing includes the most useful ones for configuring users in ARI. For a full description, see the [ARI configuration](#) documentation.

Section	Parameter	Type	Default	Description
general				
	enabled	Boolean	Yes	Enable/disable ARI.
	pretty	Boolean	No	Format JSON responses and events in a human readable form. This makes the output easier to read, at the cost of some additional bytes.
	allowed_origins	String		A comma separated list of allowed origins for <a href="#">Cross-Origin Resource Sharing</a> .
[user_name]				
	type	String		Must be <code>user</code> . Specifies that this configuration section defines a user for ARI.
	read_only	Boolean	No	Whether or not the user can issue requests that alter the Asterisk system. If set to <code>Yes</code> , then only <code>GET</code> and <code>OPTIONS</code> HTTP requests will be serviced.
	password_format	String	plain	Can be either <code>plain</code> or <code>crypt</code> . When the password is plain, Asterisk will expect the user's password to be in plain text in the <code>password</code> field. When set to <code>crypt</code> , Asterisk will use <code>crypt(3)</code> to decrypt the password. A crypted password can be generated using <code>mkpasswd -m sha-512</code> .
	password	String		The password for the user.

### Example `ari.conf`

```
ari.conf

[general]
enabled = yes
pretty = yes
allowed_origins = localhost:8088,http://ari.asterisk.org

[asterisk]
type = user
read_only = no
password = asterisk

; password_format may be set to plain (the default) or crypt. When set to crypt,
; crypt(3) is used to validate the password. A crypted password can be generated
; using mkpasswd -m sha-512.
;

[asterisk-supersecret]
type = user
read_only = no
password =
$6$nvqAB8BvsldJ4V$8zCUygFXuXXp8EU3t2M8i.N8iCsY4WRchxe2AYgGOzHAQrmjIPif3DYrvdj5U2CilLLMcht
mFyvFa3XHSxB1B/
password_format = crypt
```

## Configuring the Dialplan for ARI

By default, ARI cannot just manipulate any arbitrary channel in Asterisk. That channel may be in a long running dialplan application; it may be controlled by an AGI; it may be hung up! Many operations that ARI exposes would be fundamentally unsafe if Asterisk did not hand control of the channel over to ARI in a safe fashion.

To hand a channel over to ARI, Asterisk uses a dialplan application called [Stasis](#). Stasis acts as any other dialplan application in Asterisk, except that it does not do anything to the channel other than safely pass control over to an ARI application. The Stasis dialplan application takes in two parameters:

1. The name of the ARI application to hand the channel over to. Multiple ARI applications can exist with a single instance of Asterisk, and each ARI application will only be able to manipulate the channels that it controls.
2. Optionally, arguments to pass to the ARI application when the channel is handed over.

### **Example: Two ARI Applications**

This snippet of dialplan, taken from `extensions.conf`, illustrates two ARI applications. The first hands a channel over to an ARI application "Intro-IVR" without any additional parameters; the second hands a channel over to an ARI application "Super-Conference" with a parameter that specifies a conference room to enter.

```
extensions.conf

[default]

exten => ivr,1,NoOp()
same =>     n,Stasis(Intro-IVR)
same =>     n,Hangup()

exten => conference,1,NoOp()
same =>     n,Stasis(Super-Conference,100)
same =>     n,Hangup()
```

When a channel enters into a Stasis application, Asterisk will check to see if a WebSocket connection has been established for that application. If so, the channel is handed over to ARI for control, a subscription for the channel is made for the WebSocket, and a `StasisStart` event is sent to the WebSocket notifying it that a channel has entered into its application.



#### **A WebSocket connection is necessary!**

If you have not connected a WebSocket to Asterisk for a particular application, when a channel enters into Stasis for that application, Asterisk will immediately eject the channel from the application and return back to the dialplan. This is to prevent channels from entering into an application before something is ready to handle them.

Note that if a connection is broken, Asterisk will know that a connection previously existed and will allow channels to enter (although you may get warned that events are about to get missed...)

# Introduction to ARI and Channels

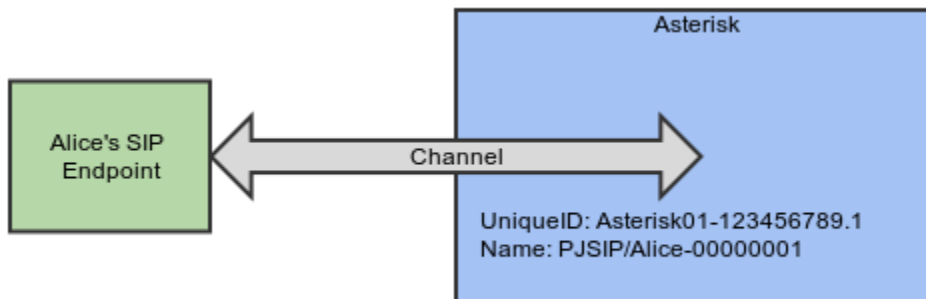
## Channels: An Overview

In Asterisk, a channel is a patch of communication between some endpoint and Asterisk itself. The path of communication encompasses all information passed to and from the endpoint. That includes both the signalling (such as "change the state of the device to ringing" or "hangup this call") as well as media (the actual audio or video being sent/received to/from the endpoint).

When a channel is created in Asterisk to represent this path of communication, Asterisk assigns it both a **UniqueID** - which acts as a handle to the channel for its entire lifetime - as well as a unique **Name**. The UniqueID can be any globally unique identifier provided by the ARI client. If the ARI client does not provide a UniqueID to the channel, then Asterisk will assign one to the channel itself. By default, it uses an epoch timestamp with a monotonically increasing integer, optionally along with the Asterisk system name.

## Channels to Endpoints

The channel name consists of two parts: the type of channel being created, along with a descriptive identifier determined by the channel type. What channel types are available depends on how the Asterisk system is configured; for the purposes of most examples, we will use "PJSIP" channels to communicate with SIP devices.



In the above diagram, Alice's SIP device has called into Asterisk, and Asterisk has assigned the resulting channel a UniqueID of **Asterisk01-123456789.1**, while the PJSIP channel driver has assigned a name of **PJSIP/Alice-00000001**. In order to manipulate this channel, ARI operations would use the UniqueID Asterisk01-123456789.1 as the handle to the channel.

On This Page
<ul style="list-style-type: none"><li>• Channels: An Overview<ul style="list-style-type: none"><li>• Channels to Endpoints</li><li>• Internal Channels - Local Channels</li></ul></li><li>• Channels in a Stasis Application</li><li>• Example: Interacting with Channels<ul style="list-style-type: none"><li>• Dialplan</li><li>• Python<ul style="list-style-type: none"><li>• channel-dump.py</li><li>• channel-dump.py in action</li></ul></li><li>• JavaScript (Node.js)<ul style="list-style-type: none"><li>• channel-dump.js</li><li>• channel-dump.js in action</li></ul></li></ul></li></ul>

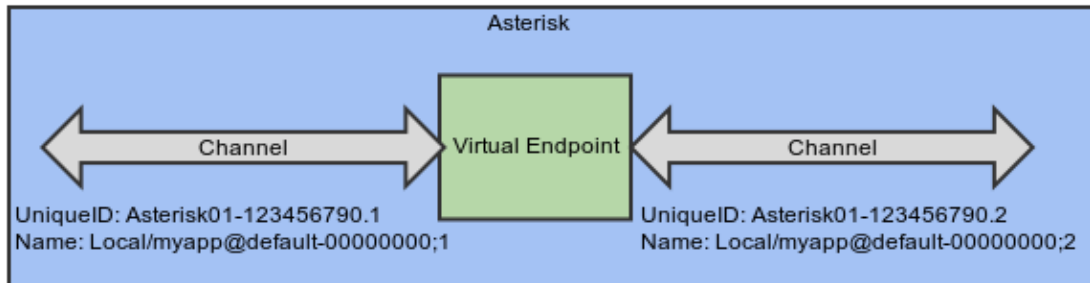
Channels In Depth
<ul style="list-style-type: none"><li>• ARI and Channels: Manipulating Channel State</li><li>• ARI and Channels: Simple Media Manipulation</li><li>• ARI and Channels: Handling DTMF</li></ul>

## Internal Channels - Local Channels

While most channels are between some external endpoint and Asterisk, Asterisk can also create channels that are completely internal within itself. These channels - called **Local** channels - help to move media between various resources within Asterisk.

Local channels are special in that Local channels always come in pairs of channels. Creating a single Local "channel" will *always* result in two channels being created in Asterisk. Sitting between the Local channel pairs is a special virtual endpoint that forwards media back and forth between the two Local channel pairs. One end of each Local channel is permanently tied to this virtual endpoint and cannot be moved about - however, the other end of each

Local channel can be manipulated in any fashion. All media that enters into one of the Local channel halves is passed out through the other Local channel half, and vice versa.



In the above diagram, ARI has created a Local channel, `Local/myapp@default`. As a result, Asterisk has created a pair of Local channels with the UniqueIDs of **Asterisk01-123456790.1** and **Asterisk01-123456790.2**. The names of the Local channel halves are **Local/myapp@default-00000000;1** and **Local/myapp@default-00000000;2** - where the ;1 and ;2 denote the two halves of the Local channel.

## Channels in a Stasis Application

When a channel is created in Asterisk, it begins to execute dialplan. All channels enter into the dialplan at some location defined by a **context/extension/priority** tuple. Each tuple location in the dialplan defines some Asterisk application that the channel should go execute. When the application completes, the priority in the tuple is increased by one, and the next location in the dialplan is executed. This continues until the dialplan runs out of things to execute, the dialplan application tells the channel to hangup, or until the device itself hangs up.

Channels are handed over to ARI through the `Stasis` dialplan application. This special application takes control of the channel from the dialplan, and indicates to an ARI client with a connected websocket that a channel is now ready to be controlled. When this occurs, a `StasisStart` event is emitted; when the channel leaves the `Stasis` dialplan application - either because it was told to leave or because the device hung up - a `StasisEnd` event is emitted. When the `StasisEnd` event is emitted, ARI no longer controls the channel and the channel is handed back over to the dialplan.

Resources in Asterisk do not, by default, send events about themselves to a connected ARI application. In order to get events about resources, one of three things must occur:

1. The resource must be a channel that entered into a `Stasis` dialplan application. A subscription is implicitly created in this case. The subscription is implicitly destroyed when the channel leaves the `Stasis` dialplan application.
2. While a channel is in a `Stasis` dialplan application, the channel may interact with other resources - such as a `bridge`. While channels interact with the resource, a subscription is made to that resource. When no more channels in a `Stasis` dialplan application are interacting with the resource, the implicit subscription is destroyed.
3. At any time, an ARI application may make a subscription to a resource in Asterisk through `application` operations. While that resource exists, the ARI application owns the subscription.

## Example: Interacting with Channels

For this example, we're going to write an ARI application that will do the following:

1. When it connects, it will print out the names of all existing channels. If there are no existing channels, it will tell us that as well.
2. When a channel enters into its `Stasis` application, it will print out all of the specific information about that channel.
3. When a channel leaves its `Stasis` application, it will print out that the channel has left.

### Dialplan

The dialplan for this will be very straight forward: a simple extension that drops a channel into `Stasis`.

```

extensions.conf

[default]

exten => 1000,1,NoOp()
same => n,Answer()
same => n,Stasis(channel-dump)
same => n,Hangup()

```

### Python

For our Python examples, we will rely primarily on the `ari-py` library. Because the `ari` library will emit useful information using Python logging, we should go

ahead and set that up as well - for now, a `basicConfig` with `ERROR` messages displayed should be sufficient. Finally, we'll need to get a client made by initiating a connection to Asterisk. This occurs using the `ari.connect` method, where we have to specify three things:

1. The HTTP base URI of the Asterisk server to connect to. Here, we assume that this is running on the same machine as the script, and that we're using the default port for Asterisk's HTTP server - 8088.
2. The username of the ARI user account to connect as. In this case, we're specifying it as `asterisk`.
3. The password for the ARI user account. In this case, that's `asterisk`.



Modify the connection credentials as appropriate for your server, although many examples will use these credentials.

**Please don't use these credentials in production systems!**

```
#!/usr/bin/env python

import ari
import logging

logging.basicConfig(level=logging.ERROR)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')
```

Once we've made our connection, our first task is to print out all existing channels or - if there are no channels - print out that there are no channels. The `channels` resource has an operation for this - `GET /channels`. Since the `ari-py` library will dynamically construct operations on objects that map to resource calls using the nickname of an operation, we can use the `list` method on the `channels` resource to get all current channels in Asterisk:

```
current_channels = client.channels.list()
if (len(current_channels) == 0):
    print "No channels currently :-("
else:
    print "Current channels:"
    for channel in current_channels:
        print channel.json.get('name')
```

The `GET /channels` operation returns back a list of `Channel` resources. Those resources, however, are returned as JSON from the operation, and while the `ari-py` library converts the `uniqueid` of those into an attribute on the object, it leaves the rest of them in the JSON dictionary. Since what we want is the name, we can just extract it ourselves out of the JSON and print it out.

Our next step involves a bit more - we want to print out all the information about a channel when it enters into our Stasis dialplan application "channel-dump" and print the channel name when it leaves. To do that, we need to subscribe for the `StasisStart` and `StasisEnd` events:

```
client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('StasisEnd', stasis_end_cb)
```

We need two handler functions - `stasis_start_cb` for the `StasisStart` event and `stasis_end_cb` for the `StasisEnd` event:

```
def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    channel = channel_obj.get('channel')
    print "Channel %s has entered the application" % channel.json.get('name')

    for key, value in channel.json.items():
        print "%s: %s" % (key, value)

def stasis_end_cb(channel, ev):
    """Handler for StasisEnd event"""

    print "%s has left the application" % channel.json.get('name')
```

Finally, we need to tell the `client` to run our application. Once we call `client.run`, the websocket connection will be made and our application will wait on events infinitely. We can use `Ctrl+C` to kill it and break the connection.

```
client.run(apps='channel-dump')
```

### **channel-dump.py**

The full source code for `channel-dump.py` is shown below:

```
channel-dump.py
#!/usr/bin/env python

import ari
import logging

logging.basicConfig(level=logging.ERROR)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

current_channels = client.channels.list()
if (len(current_channels) == 0):
    print "No channels currently :-( "
else:
    print "Current channels:"
    for channel in current_channels:
        print channel.json.get('name')

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    channel = channel_obj.get('channel')
    print "Channel %s has entered the application" % channel.json.get('name')

    for key, value in channel.json.items():
        print "%s: %s" % (key, value)

def stasis_end_cb(channel, ev):
    """Handler for StasisEnd event"""

    print "%s has left the application" % channel.json.get('name')

client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('StasisEnd', stasis_end_cb)

client.run(apps='channel-dump')
```

### **channel-dump.py in action**

Here's sample output from `channel-dump.py`. When it first connects there are no channels in Asterisk - 😞 - but afterwards a PJSIP channel from Alice enters into extension 1000. This prints out all the information about her channels. After hearing silence for awhile, she hangs up - and our script notifies us that her channel has left the application.

```

asterisk:~$ python channel-dump.py
No channels currently :-()
Channel PJSIP/alice-00000001 has entered the application
accountcode:
name: PJSIP/alice-00000001
caller: {u'Alice': u'', u'6575309': u''}
creationtime: 2014-06-09T17:36:31.698-0500
state: Up
connected: {u'name': u'', u'number': u''}
dialplan: {u'priority': 3, u'exten': u'1000', u'context': u'default'}
id: asterisk-01-1402353503.1
PJSIP/alice-00000001 has left the application

```

## JavaScript (Node.js)

For our JavaScript examples, we will rely primarily on the Node.js [ari-client](#) library. We'll need to get a client made by initiating a connection to Asterisk. This occurs using the `ari.connect` method, where we have to specify four things:

1. The HTTP base URI of the Asterisk server to connect to. Here, we assume that this is running on the same machine as the script, and that we're using the default port for Asterisk's HTTP server - 8088.
2. The username of the ARI user account to connect as. In this case, we're specifying it as `asterisk`.
3. The password for the ARI user account. In this case, that's `asterisk`.
4. A callback that will be called with an error if one occurred, followed by an instance of an ARI client.



Modify the connection credentials as appropriate for your server, although many examples will use these credentials.

**Please don't use these credentials in production systems!**

```

/*jshint node:true*/
'use strict';

var ari = require('ari-client');
var util = require('util');

ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

// handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }
}

```

Once we've made our connection, our first task is to print out all existing channels or - if there are no channels - print out that there are no channels. The `channels` resource has an operation for this - `GET /channels`. Since the `ari-client` library will dynamically construct a client with operations on objects that map to resource calls using the nickname of an operation, we can use the `list` method on the `channels` resource to get all current channels in Asterisk:

```

client.channels.list(function(err, channels) {
  if (!channels.length) {
    console.log('No channels currently :-()');
  } else {
    console.log('Current channels:');
    channels.forEach(function(channel) {
      console.log(channel.name);
    });
  }
});

```

The `GET /channels` operation expects a callback that will be called with an error if one occurred and a list of `Channel` resources. `ari-client` will

return a JavaScript object for each Channel resource. Properties such as name can be accessed on the object directly.

Our next step involves a bit more - we want to print out all the information about a channel when it enters into our Stasis dialplan application "channel-dump" and print the channel name when it leaves. To do that, we need to subscribe for the StasisStart and StasisEnd events:

```
client.on('StasisStart', stasisStart);
client.on('StasisEnd', stasisEnd);
```

We need two callback functions - stasisStart for the StasisStart event and stasisEnd for the StasisEnd event:

```
// handler for StasisStart event
function stasisStart(event, channel) {
  console.log(util.format(
    'Channel %s has entered the application', channel.name));
  // use keys on event since channel will also contain channel operations
  Object.keys(event.channel).forEach(function(key) {
    console.log(util.format('%s: %s', key, JSON.stringify(channel[key])));
  });
}
// handler for StasisEnd event
function stasisEnd(event, channel) {
  console.log(util.format(
    'Channel %s has left the application', channel.name));
}
```

Finally, we need to tell the client to start our application. Once we call client.start, a websocket connection will be established and the client will emit Node.js events as events come in through the websocket. We can use Ctrl+C to kill it and break the connection.

```
client.start('channel-dump');
```

### **channel-dump.js**

The full source code for channel-dump.js is shown below:



```

/*jshint node:true*/
'use strict';

var ari = require('ari-client');
var util = require('util');

ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

// handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }

  client.channels.list(function(err, channels) {
    if (!channels.length) {
      console.log('No channels currently :-(');
    } else {
      console.log('Current channels:');
      channels.forEach(function(channel) {
        console.log(channel.name);
      });
    }
  });
};

// handler for StasisStart event
function stasisStart(event, channel) {
  console.log(util.format(
    'Channel %s has entered the application', channel.name));

  // use keys on event since channel will also contain channel operations
  Object.keys(event.channel).forEach(function(key) {
    console.log(util.format('%s: %s', key, JSON.stringify(channel[key])));
  });
}

// handler for StasisEnd event
function stasisEnd(event, channel) {
  console.log(util.format(
    'Channel %s has left the application', channel.name));
}

client.on('StasisStart', stasisStart);
client.on('StasisEnd', stasisEnd);

client.start('channel-dump');
}

```

### ***channel-dump.js in action***

Here's sample output from `channel-dump.js`. When it first connects there are no channels in Asterisk - 😞 - but afterwards a PJSIP channel from Alice enters into extension 1000. This prints out all the information about her channels. After hearing silence for a while, she hangs up - and our script notifies us that her channel has left the application.

```
asterisk:~$ node channel-dump.js
No channels currently :-(
Channel PJSIP/alice-00000001 has entered the application
accountcode:
name: PJSIP/alice-00000001
caller: {u'Alice': u'', u'6575309': u''}
creationtime: 2014-06-09T17:36:31.698-0500
state: Up
connected: {u'name': u'', u'number': u''}
dialplan: {u'priority': 3, u'exten': u'1000', u'context': u'default'}
id: asterisk-01-1402353503.1
PJSIP/alice-00000001 has left the application
```

## ARI and Channels: Manipulating Channel State

### Channel State

A channel's state reflects the current state of the path of communication between Asterisk and a device. What state a channel is in also affects what operations are allowed on it and/or how certain operations will affect a device.

While there are many states a channel can be in, the following are the most common:

- **Down** - a path of communication exists or used to exist between Asterisk and the device, but no media can flow between the two.
- **Ringing** - the device is ringing. Media may or may not be able to flow from Asterisk to the device.
- **Up** - the device has been answered. When in the up state, media can flow bidirectionally between Asterisk and the device.



#### More Channel States

Certain channel technologies, such as DAHDI analog channels, may have additional channel states (such as "Pre-ring" or "Dialing Offhook"). When handling channel state, consult the [Channel data model](#) for all possible values.

#### On This Page

- Channel State
  - [Indicating Ringing](#)
  - [Answering a Channel](#)
  - [Hanging up a channel](#)
- Example: Manipulating Channel State
  - [Dialplan](#)
  - [Python](#)
    - [channel-state.py](#)
    - [channel-state.py in action](#)
  - [JavaScript \(Node.js\)](#)
    - [channel-state.js](#)
    - [channel-state.js in action](#)

### Indicating Ringing

Asterisk can inform a device that it should start playing a ringing tone back to the caller using the `POST /channels/{channel_id}/ring` operation. Likewise, ringing can be stopped using the `DELETE /channels/{channel_id}/ring` operation. Note that indicating ringing typically does not actually transmit media from Asterisk to the device in question - Asterisk merely signals the device to ring. It is up to the device itself to actually play something back for the user.

### Answering a Channel

When a channel isn't answered, Asterisk has typically not yet informed the device how it will communicate with it. Answering a channel will cause Asterisk to complete the path of communication, such that media flows bi-directionally between the device and Asterisk.

You can answer a channel using the `POST /channels/{channel_id}/answer` operation.

### Hanging up a channel

You can hang up a channel using the `DELETE /channels/{channel_id}` operation. When this occurs, the path of communication between Asterisk and the device is terminated, and the channel will leave the Stasis application. Your application will be notified of this via a `StasisEnd` event.

The same is true if the device initiates the hang up. In the same fashion, the path of communication between Asterisk and the device is terminated, the channel is hung up, and your application is informed that the channel is leaving your application via a `StasisEnd` event.

Generally, once a channel leaves your application, you won't receive any more events about the channel. There are times, however, when you may be subscribed to all events coming from a channel - regardless if that channel is in your application or not. In that case, a `ChannelDestroyed` event will inform you when the channel is well and truly dead.

### Example: Manipulating Channel State

For this example, we're going to write an ARI application that will do the following:

1. Wait for a channel to enter its Stasis application.
2. When a channel enters its Stasis application, it will indicate ringing to the channel. If the channel wasn't already ringing, it will now!
3. After a few seconds, it will answer the channel.
4. Once the channel is answered, we'll start silence on the channel so that the user feels a comfortable wishing noise. Then, after a few more seconds, we'll hangup the channel.
5. If at any point in time the phone hangs up first, we'll gracefully handle that.

## Dialplan

For this example, we need to just drop the channel into Stasis, specifying our application:

```
extensions.conf
exten => 1000,1,NoOp()
same =>      n,Stasis(channel-state)
same =>      n,Hangup()
```

## Python

This example will use the [ari-py](#) library. The basic structure is very similar to the [channel-dump Python example](#) - see that example for more information on the basics of setting up an ARI connection using this library.

To start, once our ARI client has been set up, we will want to register handlers for three different events - `StasisStart`, `ChannelStateChange`, and `StasisEnd`.

1. The bulk of the work will be done in `StasisStart`, which is called when the channel enters our application. For the most part, this will involve setting up Python timers to initiate actions on the channel.
2. The `ChannelStateChange` handler will merely print out the channel state changes for us, which is informative as it will tell us when the channel is answered.
3. Finally, the `StasisEnd` event will clean up for us by cancelling any pending timers that we initiated. This will get called when the channel leaves our application - which will happen when the user hangs up the channel, or when we hang up the channel.

We can store the timers that we've set up for a channel using a dictionary of channel IDs to timer instances:

```
channel_timers = {}
```

And we can register for our three events:

```
client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('ChannelStateChange', channel_state_change_cb)
client.on_channel_event('StasisEnd', stasis_end_cb)
```

The `StasisStart` event is the most interesting part.

1. First, we tell the channel to ring, and after two seconds, to answer the channel:

```
channel.ring()
# Answer the channel after 2 seconds
timer = threading.Timer(2, answer_channel, [channel])
channel_timers[channel.id] = timer
timer.start()
```

If we didn't have that there, then the caller would probably just have dead space to listen to! Not very enjoyable. We store the timer in the `channel_timers` dictionary so that our `StasisEnd` event can cancel it for us if the user hangs up the phone.

2. Once we're in the `answer_channel` handler, we answer the channel and start silence on the channel. That (hopefully) gives them a slightly more ambient silence noise. Note that we'll go ahead and declare `answer_channel` as a nested function inside our `StasisStart` handler, `stasis_start_cb`:

```
def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    def answer_channel(channel):
        """Callback that will actually answer the channel"""
        print "Answering channel %s" % channel.json.get('name')
        channel.answer()
        channel.startSilence()
```

3. After we've answered the channel, we kick off another Python timer to hang up the channel in 4 seconds. When that timer fires, it will call `hangup_channel`. This does the final action on the channel by hanging it up. Again, we'll declare `hangup_channel` as a nested function inside our `StasisStart` handler.

sisStart handler:

```
def hangup_channel(channel):
    """Callback that will actually hangup the channel"""

    print "Hanging up channel %s" % channel.json.get('name')
    channel.hangup()
```

When we create a timer - such as when we started ringing on the channel - we stored it in our `channel_timers` dictionary. In our `StasisEnd` event handler, we'll want to cancel any pending timers. Otherwise, our timers may fire and try to perform an action on channel that has already left our Stasis application, which is a good way to get an HTTP error response code.

```
def stasis_end_cb(channel, ev):
    """Handler for StasisEnd event"""

    print "Channel %s just left our application" % channel.json.get('name')

    # Cancel any pending timers
    timer = channel_timers.get(channel.id)
    if timer:
        timer.cancel()
        del channel_timers[channel.id]
```

Finally, we want to print out the state of the channel in the `ChannelStateChanged` handler. This will tell us exactly when our channel has been answered:

```
def channel_state_change_cb(channel, ev):
    """Handler for changes in a channel's state"""
    print "Channel %s is now: %s" % (channel.json.get('name'),
                                    channel.json.get('state'))
```

### channel-state.py

The full source code for `channel-state.py` is shown below:

```
channel-state.py
#!/usr/bin/env python

import ari
import logging
import threading

logging.basicConfig(level=logging.ERROR)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

channel_timers = {}

def stasis_end_cb(channel, ev):
    """Handler for StasisEnd event"""

    print "Channel %s just left our application" % channel.json.get('name')

    # Cancel any pending timers
    timer = channel_timers.get(channel.id)
    if timer:
        timer.cancel()
        del channel_timers[channel.id]
```

```

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    def answer_channel(channel):
        """Callback that will actually answer the channel"""
        print "Answering channel %s" % channel.json.get('name')
        channel.answer()
        channel.startSilence()

        # Hang up the channel in 4 seconds
        timer = threading.Timer(4, hangup_channel, [channel])
        channel_timers[channel.id] = timer
        timer.start()

    def hangup_channel(channel):
        """Callback that will actually hangup the channel"""

        print "Hanging up channel %s" % channel.json.get('name')
        channel.hangup()

    channel = channel_obj.get('channel')
    print "Channel %s has entered the application" % channel.json.get('name')

    channel.ring()
    # Answer the channel after 2 seconds
    timer = threading.Timer(2, answer_channel, [channel])
    channel_timers[channel.id] = timer
    timer.start()

def channel_state_change_cb(channel, ev):
    """Handler for changes in a channel's state"""
    print "Channel %s is now: %s" % (channel.json.get('name'),
                                    channel.json.get('state'))

client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('ChannelStateChange', channel_state_change_cb)

```

```
client.on_channel_event('StasisEnd', stasis_end_cb)

client.run(apps='channel-state')
```

### channel-state.py in action

Here, we see the output from the `channel-state.py` script when a PJSIP channel for endpoint 'alice' enters into the application:

```
Channel PJSIP/alice-00000001 has entered the application
Answering channel PJSIP/alice-00000001
Channel PJSIP/alice-00000001 is now: Up
Hanging up channel PJSIP/alice-00000001
Channel PJSIP/alice-00000001 just left our application
```

### JavaScript (Node.js)

This example will use the `ari-client` library. The basic structure is very similar to the `channel-dump JavaScript example` - see that example for more information on the basics of setting up an ARI connection using this library.

To start, once our ARI client has been set up, we will want to register callbacks for three different events - `StasisStart`, `ChannelStateChange`, and `StasisEnd`.

1. The bulk of the work will be done in `StasisStart`, which is called when the channel enters our application. For the most part, this will involve setting up JavaScript timeouts to initiate actions on the channel.
2. The `ChannelStateChange` handler will merely print out the channel state changes for us, which is informative as it will tell us when the channel is answered.
3. Finally, the `StasisEnd` event will clean up for us by cancelling any pending timeouts that we initiated. This will get called when the channel leaves our application - which will happen when the user hangs up the channel, or when we hang up the channel.

We can store the timeouts that we've set up for a channel using an object of channel IDs to timer instances:

```
timers = {}
```

And we can register for our three events:

```
client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('ChannelStateChange', channel_state_change_cb)
client.on_channel_event('StasisEnd', stasis_end_cb)
```

The `StasisStart` event is the most interesting part.

1. First, we tell the channel to ring, and after two seconds, to answer the channel:

```
channel.ring(function(err) {
  if (err) {
    throw err;
  }
});
// answer the channel after 2 seconds
var timer = setTimeout(answer, 2000);
timers[channel.id] = timer;
```

If we didn't have that there, then the caller would probably just have dead space to listen to! Not very enjoyable. We store the timer in the `timers` object so that our `StasisEnd` event can cancel it for us if the user hangs up the phone.

2. Once we're in the `answer` callback, we answer the channel and start silence on the channel. That (hopefully) gives them a slightly more ambient silence noise:

```

// callback that will answer the channel
function answer() {
  console.log(util.format('Answering channel %s', channel.name));
  channel.answer(function(err) {
    if (err) {
      throw err;
    }
  });
  channel.startSilence(function(err) {
    if (err) {
      throw err;
    }
  });
  // hang up the channel in 4 seconds
  var timer = setTimeout(hangup, 4000);
  timers[channel.id] = timer;
}

```

3. After we've answered the channel, we kick off another timer to hang up the channel in 4 seconds. When that timer fires, it will call the `hangup` callback. This does the final action on the channel by hanging it up:

```

// callback that will hangup the channel
function hangup() {
  console.log(util.format('Hanging up channel %s', channel.name));
  channel.hangup(function(err) {
    if (err) {
      throw err;
    }
  });
}

```

When we create a timer - such as when we started ringing on the channel - we stored it in our `timers` object. In our `StasisEnd` event handler, we'll want to cancel any pending timers. Otherwise, our timers may fire and try to perform an action on channel that has already left our Stasis application, which is a good way to get an HTTP error response code.

```

// handler for StasisEnd event
function stasisEnd(event, channel) {
  console.log(util.format(
    'Channel %s just left our application', channel.name));
  var timer = timers[channel.id];
  if (timer) {
    clearTimeout(timer);
    delete timers[channel.id];
  }
}

```

Finally, we want to print out the state of the channel in the `ChannelStateChanged` callback. This will tell us exactly when our channel has been answered:

```

// handler for ChannelStateChange event
function channelStateChange(event, channel) {
  console.log(util.format(
    'Channel %s is now: %s', channel.name, channel.state));
}

```

**channel-state.js**



The full source code for `channel-state.js` is shown below:

### channel-state.js

```
/*jshint node: true*/
'use strict';

var ari = require('ari-client');
var util = require('util');

var timers = {};
ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

// handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }

  // handler for StasisStart event
  function stasisStart(event, channel) {
    console.log(util.format(
      'Channel %s has entered the application', channel.name));

    channel.ring(function(err) {
      if (err) {
        throw err;
      }
    });
    // answer the channel after 2 seconds
    var timer = setTimeout(answer, 2000);
    timers[channel.id] = timer;

    // callback that will answer the channel
    function answer() {
      console.log(util.format('Answering channel %s', channel.name));
      channel.answer(function(err) {
        if (err) {
          throw err;
        }
      });
      channel.startSilence(function(err) {
        if (err) {
          throw err;
        }
      });
      // hang up the channel in 4 seconds
      var timer = setTimeout(hangup, 4000);
      timers[channel.id] = timer;
    }

    // callback that will hangup the channel
    function hangup() {
      console.log(util.format('Hanging up channel %s', channel.name));
      channel.hangup(function(err) {
        if (err) {
          throw err;
        }
      });
    }
  }
}
```

```
    }  
  }  
  
  // handler for StasisEnd event  
  function stasisEnd(event, channel) {  
    console.log(util.format(  
      'Channel %s just left our application', channel.name));  
    var timer = timers[channel.id];  
    if (timer) {  
      clearTimeout(timer);  
      delete timers[channel.id];  
    }  
  }  
}  
  
// handler for ChannelStateChange event  
function channelStateChange(event, channel) {  
  console.log(util.format(  
    'Channel %s is now: %s', channel.name, channel.state));  
}  
  
client.on('StasisStart', stasisStart);  
client.on('StasisEnd', stasisEnd);  
client.on('ChannelStateChange', channelStateChange);
```

```
client.start('channel-state');  
}
```

### **channel-state.js in action**

Here, we see the output from the `channel-state.js` script when a PJSIP channel for endpoint 'alice' enters into the application:

```
Channel PJSIP/alice-00000001 has entered the application  
Answering channel PJSIP/alice-00000001  
Channel PJSIP/alice-00000001 is now: Up  
Hanging up channel PJSIP/alice-00000001  
Channel PJSIP/alice-00000001 just left our application
```

## ARI and Channels: Simple Media Manipulation

### Simple media playback

Almost all media is played to a channel using the `POST /channels/{channel_id}/play` operation. This will do the following:

1. Create a new `Playback` object for the channel. If a media operation is currently in progress on the channel, the new `Playback` object will be queued up for the channel.
2. The `media` URI passed to the `play` operation will be inspected, and Asterisk will attempt to find the media requested. Currently, the following media schemes are supported:

URI Scheme	Description
<code>sound</code>	A sound file located on the Asterisk system. You can use the <code>/sounds</code> resource to query for available sounds on the system.
<code>recording</code>	A <code>StoredRecording</code> stored on the Asterisk system. You can use the <code>/recordings/stored</code> resource to query for available <code>StoredRecordings</code> on the system.
<code>number</code>	Play back the specified number. This uses the same mechanism as Asterisk's <code>Say</code> family of applications.
<code>digits</code>	Play back the specified digits. This uses the same mechanism as Asterisk's <code>Say</code> family of applications.
<code>characters</code>	Play back the specified characters. This uses the same mechanism as Asterisk's <code>Say</code> family of applications.
<code>tone</code>	Play a particular tone sequence until stopped. This can be used to play locale specific ringing, stutter, busy, congestion, or other tones to a device.

3. Once the media operation is started or enqueued, the `Playback` object will be returned to the caller in the `HTTP` response to the request. The caller can use that playback object to manipulate the media operation.

### On This Page

- [Simple media playback](#)
  - [Early Media](#)
- [Example: Playing back tones](#)
  - [Dialplan](#)
  - [Python](#)
    - [channel-tones.py](#)
    - [channel-tones.py in action](#)
  - [JavaScript \(Node.js\)](#)
    - [channel-tones.js](#)
    - [channel-tones.js in action](#)
- [Example: Playing back a sound file](#)
  - [Dialplan](#)
  - [Python](#)
    - [channel-playback-monkeys.py](#)
    - [channel-playback-monkeys.py in action](#)
  - [JavaScript \(Node.js\)](#)
    - [channel-playback-monkeys.js](#)
    - [channel-playback-monkeys.js in action](#)



#### Specify your own ID!

It is **highly** recommended that the `POST /channels/{channel_id}/play/{playback_id}` operation be used instead of the `POST /channels/{channel_id}/play` variant. Asterisk lives in an asynchronous world - which is the same world you and I live in. Sometimes, if things happen *very* quickly, you may get notifications over the WebSocket about things you have started before the HTTP response completes!

When you specify your own ID, you have the ability to tie information coming from events back to whatever operation you initiated - if you so choose to. If you use the non-ID providing variant, Asterisk will happily generate a UUID for your `Playback` object - but then it is up to you to

deal with whatever information comes back from the WebSocket.

## Early Media

Generally, before a channel has been answered and transitioned to the Up state, media cannot pass between Asterisk and a device. For example, if Asterisk is placing an outbound call to a device, the device may be ringing but no one has picked up a handset yet! In such circumstances, media cannot be successfully played to the ringing device - after all, who could listen to it?

However, with inbound calls, Asterisk is the entity that decides when the path of communication between itself and the device is answered - not the user on the other end. This can be useful when answering the channel may trigger billing times or other mechanisms that we don't want to fire yet. This is called "early media". For the channel technologies that support this, ARI and Asterisk will automatically handle sending the correct indications to the ringing phone before sending it media. The same `play` operation can be used both for "regular" playback of media, as well as for "early media" scenarios.

## Example: Playing back tones

This example ARI application will do the following:

1. When a channel enters into the Stasis application, it will start a playback of a French ringing tone.
2. After 8 seconds, the channel will be answered.
3. After 1 second, the channel will be rudely hung up on - we didn't want to talk to them anyway!

## Dialplan

For this example, we need to just drop the channel into Stasis, specifying our application:

### extensions.conf

```
exten => 1000,1,NoOp()  
same =>      n,Stasis(channel-tones)  
same =>      n,Hangup()
```

## Python

This example will use a very similar structure as the `channel-state.py` example. Instead of performing a `ring` operation in our `StasisStart` handler, we'll instead initiate a playback using the `playWithId` operation on the channel. Note that our URI uses the `tone` scheme, which supports an optional `tonezone` parameter. We specify our `tonezone` as `fr`, so that we get an elegant French ringing tone. Much like the `channel-state.py` example, we then use a Python timer to schedule a callback that will answer the channel. Since we care about both the `channel` and the `playback` initiated on it, we pass both parameters as `*args` parameters to the callback function.

```
playback_id = str(uuid.uuid4())  
playback = channel.playWithId(playbackId=playback_id,  
                              media='tone:ring;tonezone=fr')  
timer = threading.Timer(8, answer_channel, [channel, playback])
```

Since this is a media operation and not *technically* a ringing indication, when we `answer` the channel, the tone playback will not stop! To stop playing back our French ringing tone, we issue a `stop` operation on the `playback` object. This actually maps to a `DELETE /playbacks/{playback_id}` operation.

```
def answer_channel(channel, playback):  
    """Callback that will actually answer the channel"""  
  
    print "Answering channel %s" % channel.json.get('name')  
    playback.stop()  
    channel.answer()
```

Once answered, we'll schedule another Python timer that will do the actual hanging up of the channel.

## channel-tones.py

The full source code for `channel-tones.py` is shown below:

### channel-tones.py

```
#!/usr/bin/env python
```

```

import ari
import logging
import threading
import uuid

logging.basicConfig(level=logging.ERROR)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

channel_timers = {}

def stasis_end_cb(channel, ev):
    """Handler for StasisEnd event"""

    print "Channel %s just left our application" % channel.json.get('name')
    timer = channel_timers.get(channel.id)
    if timer:
        timer.cancel()
        del channel_timers[channel.id]

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    def answer_channel(channel, playback):
        """Callback that will actually answer the channel"""

        print "Answering channel %s" % channel.json.get('name')
        playback.stop()
        channel.answer()

        timer = threading.Timer(1, hangup_channel, [channel])
        channel_timers[channel.id] = timer
        timer.start()

    def hangup_channel(channel):
        """Callback that will actually hangup the channel"""

        print "Hanging up channel %s" % channel.json.get('name')
        channel.hangup()

    channel = channel_obj.get('channel')
    print "Channel %s has entered the application" % channel.json.get('name')

    playback_id = str(uuid.uuid4())
    playback = channel.playWithId(playbackId=playback_id,
                                  media='tone:ring;tonezone=fr')
    timer = threading.Timer(8, answer_channel, [channel, playback])
    channel_timers[channel.id] = timer
    timer.start()

client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('StasisEnd', stasis_end_cb)

```

```
client.run(apps='channel-tones')
```

### channel-tones.py in action

The following shows the output of the `channel-tones.js` script when a PJSIP channel for `alice` enters the application:

```
Channel PJSIP/alice-00000000 has entered the application
Answering channel PJSIP/alice-00000000
Hanging up channel PJSIP/alice-00000000
Channel PJSIP/alice-00000000 just left our application
```

### JavaScript (Node.js)

This example will use a very similar structure as the `channel-state.js` example. Instead of performing a `ring` operation in our `StasisStart` handler, we'll instead initiate a `playback` using the `play` operation on the channel. Note that our URI uses the `tone` scheme, which supports an optional `tonezone` parameter. We specify our `tonezone` as `fr`, so that we get an elegant French ringing tone. Much like the `channel-state.js` example, we then use a JavaScript timeout to schedule a callback that will answer the channel.

```
var playback = client.Playback();
channel.play({media: 'tone:ring;tonezone=fr'},
             playback, function(err, newPlayback) {
  if (err) {
    throw err;
  }
});
// answer the channel after 8 seconds
var timer = setTimeout(answer, 8000);
timers[channel.id] = timer;
```

Since this is a media operation and not *technically* a ringing indication, when we `answer` the channel, the tone playback will not stop! To stop playing back our French ringing tone, we issue a `stop` operation on the `playback` object. This actually maps to a `DELETE /playbacks/{playback_id}` operation. Notice that we use the fact that the `answer` callback closes on the original channel and `playback` variables to access them from the callback.

```
function answer() {
  console.log(util.format('Answering channel %s', channel.name));
  playback.stop(function(err) {
    if (err) {
      throw err;
    }
  });
  channel.answer(function(err) {
    if (err) {
      throw err;
    }
  });
  // hang up the channel in 1 seconds
  var timer = setTimeout(hangup, 1000);
  timers[channel.id] = timer;
}
```

Once answered, we'll schedule another timeout that will do the actual hanging up of the channel.

### channel-tones.js

The full source code for `channel-tones.js` is shown below:

#### channel-tones.js

```
/*jshint node: true*/
```

```

'use strict';

var ari = require('ari-client');
var util = require('util');

var timers = {};
ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

// handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }

  // handler for StasisStart event
  function stasisStart(event, channel) {
    console.log(util.format(
      'Channel %s has entered the application', channel.name));

    var playback = client.Playback();
    channel.play({media: 'tone:ring;tonezone=fr'},
      playback, function(err, newPlayback) {
        if (err) {
          throw err;
        }
      });
    // answer the channel after 8 seconds
    var timer = setTimeout(answer, 8000);
    timers[channel.id] = timer;

    // callback that will answer the channel
    function answer() {
      console.log(util.format('Answering channel %s', channel.name));
      playback.stop(function(err) {
        if (err) {
          throw err;
        }
      });
      channel.answer(function(err) {
        if (err) {
          throw err;
        }
      });
      // hang up the channel in 1 seconds
      var timer = setTimeout(hangup, 1000);
      timers[channel.id] = timer;
    }

    // callback that will hangup the channel
    function hangup() {
      console.log(util.format('Hanging up channel %s', channel.name));
      channel.hangup(function(err) {
        if (err) {
          throw err;
        }
      });
    }
  }
}

```



```

// handler for StasisEnd event
function stasisEnd(event, channel) {
  console.log(util.format(
    'Channel %s just left our application', channel.name));
  var timer = timers[channel.id];
  if (timer) {
    clearTimeout(timer);
    delete timers[channel.id];
  }
}

client.on('StasisStart', stasisStart);
client.on('StasisEnd', stasisEnd);

client.start('channel-tones');
}

```

### channel-tones.js in action

The following shows the output of the `channel-tones.js` script when a PJSIP channel for alice enters the application:

```

Channel PJSIP/alice-00000000 has entered the application
Answering channel PJSIP/alice-00000000
Hanging up channel PJSIP/alice-00000000
Channel PJSIP/alice-00000000 just left our application

```

### Example: Playing back a sound file

This example ARI application will do the following:

1. When a channel enters the Stasis application, initiate a playback of howler monkeys on the channel. Fly my pretties, FLY!
2. If the user has not hung up their phone in panic, it will hang up the channel when the howler monkeys return victorious - or rather, when ARI notifies the application that the playback has finished via the `PlaybackFinished` event.

### Dialplan

For this example, we need to just drop the channel into Stasis, specifying our application:

#### extensions.conf

```

exten => 1000,1,NoOp()
same =>      n,Stasis(channel-playback-monkeys)
same =>      n,Hangup()

```

### Python

Much like the `channel-tones.py` example, we'll start off by initiating a playback on the channel. Instead of specifying a tone scheme, however, we'll specify a scheme of sound with a resource of `tt-monkeys`. Unlike the tones, this media *does* have a well defined ending - the end of the sound file! So we'll subscribe for the `PlaybackFinished` event and tell `ari-py` to call `playback_finished` when our monkeys are done attacking.

```

playback_id = str(uuid.uuid4())
playback = channel.playWithId(playbackId=playback_id,
                              media='sound:tt-monkeys')
playback.on_event('PlaybackFinished', playback_finished)

```

Unfortunately, `ari-py` doesn't let us pass arbitrary data to a callback function in the same fashion as a Python timer. Nuts. Luckily, the `Playback` object has a property, `target_uri`, that tells us which object it just finished playing to. Using that, we can get the `channel` object back from Asterisk so we can hang it up.

```
def playback_finished(playback, ev):
    """Callback when the monkeys have finished howling"""

    target_uri = playback.json.get('target_uri')
    channel_id = target_uri.replace('channel:', '')
    channel = client.channels.get(channelId=channel_id)

    print "Monkeys successfully vanquished %s; hanging them up" %
channel.json.get('name')
    channel.hangup()
```

Note that unlike the `channel-tones.py` example, this application eschews the use of Python timers and simply responds to ARI events as they happen. This means we don't have to do much in our `StasisEnd` event, and we have to track less state.

#### **channel-playback-monkeys.py**

The full source code for `channel-playback-monkeys.py` is shown below:

```
#!/usr/bin/env python

import ari
import logging
import uuid

logging.basicConfig(level=logging.ERROR)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

def stasis_end_cb(channel, ev):
    """Handler for StasisEnd event"""

    print "Channel %s just left our application" % channel.json.get('name')

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    def playback_finished(playback, ev):
        """Callback when the monkeys have finished howling"""

        target_uri = playback.json.get('target_uri')
        channel_id = target_uri.replace('channel:', '')
        channel = client.channels.get(channelId=channel_id)

        print "Monkeys successfully vanquished %s; hanging them up" %
channel.json.get('name')
        channel.hangup()

    channel = channel_obj.get('channel')
    print "Monkeys! Attack %s!" % channel.json.get('name')

    playback_id = str(uuid.uuid4())
    playback = channel.playWithId(playbackId=playback_id,
                                  media='sound:tt-monkeys')
    playback.on_event('PlaybackFinished', playback_finished)

client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('StasisEnd', stasis_end_cb)

client.run(apps='channel-playback-monkeys')
```

#### channel-playback-monkeys.py in action

The following shows the output of the `channel-playback-monkeys.py` script when a PJSIP channel for `alice` enters the application:

```
Monkeys! Attack PJSIP/alice-00000000!
Monkeys successfully vanquished PJSIP/alice-00000000; hanging them up
Channel PJSIP/alice-00000000 just left our application
```

#### JavaScript (Node.js)

Much like the `channel-tones.js` example, we'll start off by initiating a playback on the channel. Instead of specifying a `tone` scheme, however, we'll specify a scheme of `sound` with a resource of `tt-monkeys`. Unlike the tones, this media *does* have a well defined ending - the end of the sound file! So

we'll subscribe for the `PlaybackFinished` event and tell `ari-client` to call `playbackFinished` when our monkeys are done attacking. Notice that we use `client.Playback()` to generate a `playback` object with a pre-existing `id` so we can scope the `PlaybackFinished` event to the `playback` we just created.

```
var playback = client.Playback();
channel.play({media: 'sound:tt-monkeys'},
             playback, function(err, newPlayback) {
  if (err) {
    throw err;
  }
});
playback.on('PlaybackFinished', playbackFinished);
```

Notice that we use the fact that the `playbackFinished` callback closes over the original `channel` variable to perform a `hangup` operation using that object directly.

```
function playbackFinished(event, completedPlayback) {
  console.log(util.format(
    'Monkeys successfully vanquished %s; hanging them up',
    channel.name));
  channel.hangup(function(err) {
    if (err) {
      throw err;
    }
  });
}
```

Note that unlike the `channel-tones.js` example, this application eschews the use of JavaScript timeouts and simply responds to ARI events as they happen. This means we don't have to do much in our `StasisEnd` event, and we have to track less state.

#### **channel-playback-monkeys.js**

The full source code for `channel-playback-monkeys.js` is shown below:

```

/*jshint node: true*/
'use strict';

var ari = require('ari-client');
var util = require('util');

ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

// handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }

  // handler for StasisStart event
  function stasisStart(event, channel) {
    console.log(util.format(
      'Monkeys! Attack %s!', channel.name));

    var playback = client.Playback();
    channel.play({media: 'sound:tt-monkeys'},
      playback, function(err, newPlayback) {
        if (err) {
          throw err;
        }
      });
    playback.on('PlaybackFinished', playbackFinished);

    function playbackFinished(event, completedPlayback) {
      console.log(util.format(
        'Monkeys successfully vanquished %s; hanging them up',
        channel.name));
      channel.hangup(function(err) {
        if (err) {
          throw err;
        }
      });
    }
  }
}

// handler for StasisEnd event
function stasisEnd(event, channel) {
  console.log(util.format(
    'Channel %s just left our application', channel.name));
}

client.on('StasisStart', stasisStart);
client.on('StasisEnd', stasisEnd);

client.start('channel-playback-monkeys');
}

```

#### channel-playback-monkeys.js in action

The following shows the output of the `channel-playback-monkeys.js` script when a PJSIP channel for alice enters the application:

```
Monkeys! Attack PJSIP/alice-00000000!  
Monkeys successfully vanquished PJSIP/alice-00000000; hanging them up  
Channel PJSIP/alice-00000000 just left our application
```

## ARI and Channels: Handling DTMF

### Handling DTMF events

DTMF events are conveyed via the `ChannelDtmfReceived` event. The event contains the channel that pressed the DTMF key, the digit that was pressed, and the duration of the digit.

While this concept is relatively straight forward, handling DTMF is quite common in applications, as it is the primary mechanism that phones have to inform a server to perform some action. This includes manipulating media, initiating call features, performing transfers, dialling, and just about every thing in between. As such, the examples on this page focus less on simply handling the event and more on using the DTMF in a relatively realistic fashion.

On This Page
<ul style="list-style-type: none"><li>• Handling DTMF events</li><li>• Example: A simple automated attendant<ul style="list-style-type: none"><li>• Dialplan</li><li>• Python<ul style="list-style-type: none"><li>• Playing the menu<ul style="list-style-type: none"><li>• Cancelling the menu</li><li>• Timing out</li></ul></li><li>• Handling the DTMF options</li><li>• <code>channel-aa.py</code></li><li>• <code>channel-aa.py</code> in action</li></ul></li><li>• JavaScript (Node.js)<ul style="list-style-type: none"><li>• Playing the menu<ul style="list-style-type: none"><li>• Cancelling the menu</li><li>• Timing out</li></ul></li><li>• Handling the DTMF options</li><li>• <code>channel-aa.js</code></li><li>• <code>channel-aa.js</code> in action</li></ul></li></ul></li></ul>

### Example: A simple automated attendant

This example mimics the automated attendant/IVR dialplan example. It does the following:

- Plays a menu to the user which is cancelled when the user takes some action.
- If the user presses 1 or 2, the digit is repeated to the user and the menu restarted.
- If the user presses an invalid digit, a prompt informing the user that the digit was invalid is played to the user and the menu restarted.
- If the user fails to press anything within some period of time, a prompt asking the user if they are still present is played to the user and the menu restarted.



For this example, you will need the following:

1. The **extra** sound package from Asterisk. You can install this using the `menuselect` tool.
2. If using the Python example, `ari-py` version 0.1.3 or later.
3. If using the JavaScript example, `ari-client` version 0.1.4 or later.

### Dialplan

As usual, a very simple dialplan is sufficient for this example. The dialplan takes the channel and places it into the Stasis application `channel-aa`.

#### extensions.conf

```
exten => 1000,1,NoOp( )
same =>      n,Stasis(channel-aa)
same =>      n,Hangup( )
```

## Python

As this example is a bit larger, how the code is written and structured is broken up into two phases:

1. Constructing the menu and handling its state as the user presses buttons.
2. Actually handling the button presses from the user.

The full source code for this example immediately follows the walk through.

### Playing the menu

Unlike Playback, which can chain multiple sounds together and play them back in one continuous operation, ARI treats all sound files being played as separate operations. It will queue each sound file up to be played on the channel, and hand back the caller an object to control the operation of that single sound file. The menu announcement for the attendant has the following requirements:

1. Playback the options for the user
2. If the user presses a DTMF key, cancel the playback of the options and handle the request
3. If the user presses an invalid DTMF key, let them know and restart the menu
4. If the user doesn't press anything, wait 10 seconds, ask them if they are still present, and restart the menu

The second requirement makes this a bit more challenging: when the user presses a DTMF key, we want to cancel whatever sound file is currently being played back and immediately handle their request. We thus have to maintain some state in our application about what sound file is currently being played so that we can cancel the correct playback. We also don't want to queue up all of the sounds immediately - we'd have to walk through all of the queued up sounds and cancel each one - that'd be annoying! Instead, we only want to start the next sound in our prompt when the previous has completed.

To start, we'll define in a list at the top of our script the sounds that make up the initial menu prompt:

```
sounds = ['press-1', 'or', 'press-2']
```

Since we'll want to maintain some state, we'll create a small object to do that for us. In Python, tuples are immutable - and we'll want to mutate the state in callbacks when certain operations happen. As such, it makes sense to use a small class for this with two properties:

1. The current sound being played
2. Whether or not we should consider the menu complete

It's useful to have both pieces of data, as we may cancel the menu half-way through and want to take one set of actions, or we may complete the menu and all the sounds and start a different set of actions.

```
class MenuState(object):
    """A small tracking object for the channel in the menu"""

    def __init__(self, current_sound, complete):
        self.current_sound = current_sound
        self.complete = complete
```

To start, we'll write a function, `play_intro_menu`, that starts the menu on a channel. It will simply initialize the state of the menu, and get the ball rolling on the channel by calling `queue_up_sound`.

```
def play_intro_menu(channel):
    """Play our intro menu to the specified channel
    Since we want to interrupt the playback of the menu when the user presses
    a DTMF key, we maintain the state of the menu via the MenuState object.
    A menu completes in one of two ways:
    (1) The user hits a key
    (2) The menu finishes to completion
    In the case of (2), a timer is started for the channel. If the timer pops,
    a prompt is played back and the menu restarted.
    Keyword Arguments:
    channel The channel in the IVR
    """
    menu_state = MenuState(0, False)

    def queue_up_sound(channel, menu_state):
        ...

    queue_up_sound(channel, menu_state)
```



`queue_up_sound` will be responsible for starting the next sound file on the channel and handling the manipulation of that sound file. Since there's a fair amount of checking that goes into this, we'll put the actual act of starting the sound in `play_next_sound`, which will return the `Playback` object from ARI. We'll prep the `menu_state` object for the next sound file playback, and pass it to the `PlaybackFinished` handler for the current sound being played back to the channel.

```
def queue_up_sound(channel, menu_state):
    """Start up the next sound and handle whatever happens

    Keywords Arguments:
    channel    The channel in the IVR
    menu_state The current state of the menu
    """

    current_playback = play_next_sound(menu_state)

    if not current_playback:
        return
    menu_state.current_sound += 1
    current_playback.on_event('PlaybackFinished', on_playback_finished,
                              callback_args=[menu_state])
```

`play_next_sound` will do two things:

1. If we shouldn't play another sound - either because we've run out of sounds to play or because the menu is now "complete", we bail and return `None`.
2. If we should play back a sound, start it up on the channel and return the `Playback` object.

```
def play_next_sound(menu_state):
    """Play the next sound, if we should

    Keyword Arguments:
    menu_state The current state of the IVR

    Returns:
    None if no playback should occur
    A playback object if a playback was started
    """

    if (menu_state.current_sound == len(sounds) or menu_state.complete):
        return None
    try:
        current_playback = channel.play(media='sound:%s' %
sounds[menu_state.current_sound])
    except:
        current_playback = None
    return current_playback
```

Our playback finished handler is very simple: since we've already incremented the state of the menu, we just call `queue_up_sound` again:

```
def on_playback_finished(playback, ev, menu_state):
    """Callback handler for when a playback is finished

    Keyword Arguments:
    playback    The playback object that finished
    ev          The PlaybackFinished event
    menu_state  The current state of the menu
    """

    queue_up_sound(channel, menu_state)
```

To recap, our `play_intro_menu` function has three nested functions:

1. `queue_up_sound` - starts a sound on a channel, increments the state of the menu, and subscribes for the `PlaybackFinished` event.
2. `play_next_sound` - if possible, actually starts the sound. Called from `queue_up_sound`.

3. `on_playback_finished` - called when `PlaybackFinished` is received for the current playback, and call `queue_up_sound` to start the next sound in the menu.

This will play back the menu sounds, but it doesn't handle cancelling the menu, time-outs, or other conditions. To do that, we're going to need more information from Asterisk.

### ***Cancelling the menu***

When the user presses a DTMF key, we want to stop the current playback and end the menu. To do that, we'll need to subscribe for DTMF events from the channel. We'll define a new handler function, `cancel_menu`, and tell `ari.py` to call it when a DTMF key is received via the `ChannelDtmfReceived` event. We don't really care about the digit here - we just want to cancel the menu. In the handler function, we'll set `menu_state.complete` to `True`, then tell the `current_playback` to stop.

We should also stop the menu when the channel is hung up. Since the `cancel_menu`, so we'll subscribe to the `StasisEnd` event here and call `cancel_menu` from it as well:

```
def queue_up_sound(channel, menu_state):
    """Start up the next sound and handle whatever happens

    Keywords Arguments:
    channel      The channel in the IVR
    menu_state   The current state of the menu
    """

    current_playback = play_next_sound(menu_state)

    def cancel_menu(channel, ev, current_playback, menu_state):
        """Cancel the menu, as the user did something"""
        menu_state.complete = True
        try:
            current_playback.stop()
        except:
            pass
        return

    if not current_playback:
        return
    menu_state.current_sound += 1
    current_playback.on_event('PlaybackFinished', on_playback_finished,
                              callback_args=[menu_state])

    # If the user hits a key or hangs up, cancel the menu operations
    channel.on_event('ChannelDtmfReceived', cancel_menu,
                    callback_args=[current_playback, menu_state])
    channel.on_event('StasisEnd', cancel_menu,
                    callback_args=[current_playback, menu_state])
```

### ***Timing out***

Now we can cancel the menu, but we also need to restart it if the user doesn't do anything. We can use a Python timer to start a timer if we're finished playing sounds *and* we got to the end of the sound prompt list. We don't want to start the timer if the user pressed a DTMF key - in that case, we would have stopped the menu early and we should be off handling their DTMF key press. The timer will call `menu_timeout`, which will play back a "are you still there?" prompt, then restart the menu.

```

def queue_up_sound(channel, menu_state):
    """Start up the next sound and handle whatever happens
    Keywords Arguments:
    channel    The channel in the IVR
    menu_state The current state of the menu
    """

    def menu_timeout(channel):
        """Callback called by a timer when the menu times out"""
        print 'Channel %s stopped paying attention...' % channel.json.get('name')
        channel.play(media='sound:are-you-still-there')
        play_intro_menu(channel)

    def cancel_menu(channel, ev, current_playback, menu_state):
        """Cancel the menu, as the user did something"""
        menu_state.complete = True
        try:
            current_playback.stop()
        except:
            pass
        return

    current_playback = play_next_sound(menu_state)
    if not current_playback:
        if menu_state.current_sound == len(sounds):
            # Menu played, start a timer!
            timer = threading.Timer(10, menu_timeout, [channel])
            channel_timers[channel.id] = timer
            timer.start()
        return

    menu_state.current_sound += 1
    current_playback.on_event('PlaybackFinished', on_playback_finished,
                             callback_args=[menu_state])

    # If the user hits a key or hangs up, cancel the menu operations
    channel.on_event('ChannelDtmfReceived', cancel_menu,
                    callback_args=[current_playback, menu_state])
    channel.on_event('StasisEnd', cancel_menu,
                    callback_args=[current_playback, menu_state])

```

Now that we've introduced timers, we know we're going to need to stop them if the user does something. We'll store the timers in a dictionary indexed by channel ID, so we can get them from various parts of the script:

```
channel_timers = {}
```

### Handling the DTMF options

While we now have code that plays back the menu to the user, we actually have to implement the attendant menu still. This is slightly easier than playing the menu. We can register for the `ChannelDtmfReceived` event in the `StasisStart` event handler. In that callback, we need to do the following:

1. Cancel any timers associated with the channel. Note that we don't need to stop the playback of the menu, as the menu function `queue_up_sound` already registers a handler for that event and cancels the menu when it gets any digit.
2. Actually handle the digit, if the digit is a 1 or a 2.
3. If the digit isn't supported, play a prompt informing the user that their option was invalid, and re-play the menu.

The following implements these three items, deferring processing of the valid options to separate functions.

```

def on_dtmf_received(channel, ev):
    """Our main DTMF handler for a channel in the IVR

    Keyword Arguments:
    channel The channel in the IVR
    digit The DTMF digit that was pressed
    """

    # Since they pressed something, cancel the timeout timer
    cancel_timeout(channel)
    digit = int(ev.get('digit'))

    print 'Channel %s entered %d' % (channel.json.get('name'), digit)
    if digit == 1:
        handle_extension_one(channel)
    elif digit == 2:
        handle_extension_two(channel)
    else:
        print 'Channel %s entered an invalid option!' % channel.json.get('name')
        channel.play(media='sound:option-is-invalid')
        play_intro_menu(channel)

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    channel = channel_obj.get('channel')
    print "Channel %s has entered the application" % channel.json.get('name')

    channel.on_event('ChannelDtmfReceived', on_dtmf_received)
    play_intro_menu(channel)

```

Cancelling the timer is done in a fashion similar to other examples. If the channel has a Python timer associated with it, we cancel the timer and remove it from the dictionary.

```

def cancel_timeout(channel):
    """Cancel the timeout timer for the channel

    Keyword Arguments:
    channel The channel in the IVR
    """
    timer = channel_timers.get(channel.id)
    if timer:
        timer.cancel()
        del channel_timers[channel.id]

```

Finally, we need to actually do *something* when the user presses a 1 or a 2. We could do anything here - but in our case, we're merely going to play back the number that they pressed and restart the menu.

```

def handle_extension_one(channel):
    """Handler for a channel pressing '1'

    Keyword Arguments:
    channel The channel in the IVR
    """
    channel.play(media='sound:you-entered')
    channel.play(media='digits:1')
    play_intro_menu(channel)

def handle_extension_two(channel):
    """Handler for a channel pressing '2'

    Keyword Arguments:
    channel The channel in the IVR
    """
    channel.play(media='sound:you-entered')
    channel.play(media='digits:2')
    play_intro_menu(channel)

```

### channel-aa.py

The full source for channel-aa.py is shown below:

#### channel-aa.py

```

#!/usr/bin/env python

import ari
import logging
import threading

logging.basicConfig(level=logging.ERROR)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

# Note: this uses the 'extra' sounds package
sounds = ['press-1', 'or', 'press-2']

channel_timers = {}

class MenuState(object):
    """A small tracking object for the channel in the menu"""

    def __init__(self, current_sound, complete):
        self.current_sound = current_sound
        self.complete = complete

def play_intro_menu(channel):
    """Play our intro menu to the specified channel

    Since we want to interrupt the playback of the menu when the user presses
    a DTMF key, we maintain the state of the menu via the MenuState object.
    A menu completes in one of two ways:
    (1) The user hits a key
    (2) The menu finishes to completion

```

In the case of (2), a timer is started for the channel. If the timer pops, a prompt is played back and the menu restarted.

Keyword Arguments:

```
channel The channel in the IVR
"""
```

```
menu_state = MenuState(0, False)
```

```
def play_next_sound(menu_state):
    """Play the next sound, if we should
```

```
Keyword Arguments:
```

```
menu_state The current state of the IVR
```

```
Returns:
```

```
None if no playback should occur
```

```
A playback object if a playback was started
```

```
"""
```

```
if (menu_state.current_sound == len(sounds) or menu_state.complete):
    return None
```

```
try:
```

```
    current_playback = channel.play(media='sound:%s' %
sounds[menu_state.current_sound])
```

```
except:
```

```
    current_playback = None
```

```
return current_playback
```

```
def on_playback_finished(playback, ev, menu_state):
```

```
    """Callback handler for when a playback is finished
```

```
Keyword Arguments:
```

```
playback The playback object that finished
```

```
ev The PlaybackFinished event
```

```
menu_state The current state of the menu
```

```
"""
```

```
queue_up_sound(channel, menu_state)
```

```
def queue_up_sound(channel, menu_state):
```

```
    """Start up the next sound and handle whatever happens
```

```
Keywords Arguments:
```

```
channel The channel in the IVR
```

```
menu_state The current state of the menu
```

```
"""
```

```
def menu_timeout(channel):
```

```
    """Callback called by a timer when the menu times out"""
```

```
    print 'Channel %s stopped paying attention...' % channel.json.get('name')
```

```
    channel.play(media='sound:are-you-still-there')
```

```
    play_intro_menu(channel)
```

```
def cancel_menu(channel, ev, current_playback, menu_state):
```

```
    """Cancel the menu, as the user did something"""
```

```
    menu_state.complete = True
```

```
    try:
```

```
        current_playback.stop()
```

```
    except:
```

```

        pass
    return

    current_playback = play_next_sound(menu_state)
    if not current_playback:
        if menu_state.current_sound == len(sounds):
            # Menu played, start a timer!
            timer = threading.Timer(10, menu_timeout, [channel])
            channel_timers[channel.id] = timer
            timer.start()
        return

    menu_state.current_sound += 1
    current_playback.on_event('PlaybackFinished', on_playback_finished,
                              callback_args=[menu_state])

    # If the user hits a key or hangs up, cancel the menu operations
    channel.on_event('ChannelDtmfReceived', cancel_menu,
                    callback_args=[current_playback, menu_state])
    channel.on_event('StasisEnd', cancel_menu,
                    callback_args=[current_playback, menu_state])

    queue_up_sound(channel, menu_state)

def handle_extension_one(channel):
    """Handler for a channel pressing '1'

    Keyword Arguments:
    channel The channel in the IVR
    """
    channel.play(media='sound:you-entered')
    channel.play(media='digits:1')
    play_intro_menu(channel)

def handle_extension_two(channel):
    """Handler for a channel pressing '2'

    Keyword Arguments:
    channel The channel in the IVR
    """
    channel.play(media='sound:you-entered')
    channel.play(media='digits:2')
    play_intro_menu(channel)

def cancel_timeout(channel):
    """Cancel the timeout timer for the channel

    Keyword Arguments:
    channel The channel in the IVR
    """
    timer = channel_timers.get(channel.id)
    if timer:
        timer.cancel()
        del channel_timers[channel.id]

```

```

def on_dtmf_received(channel, ev):
    """Our main DTMF handler for a channel in the IVR

    Keyword Arguments:
    channel The channel in the IVR
    digit   The DTMF digit that was pressed
    """

    # Since they pressed something, cancel the timeout timer
    cancel_timeout(channel)
    digit = int(ev.get('digit'))

    print 'Channel %s entered %d' % (channel.json.get('name'), digit)
    if digit == 1:
        handle_extension_one(channel)
    elif digit == 2:
        handle_extension_two(channel)
    else:
        print 'Channel %s entered an invalid option!' % channel.json.get('name')
        channel.play(media='sound:option-is-invalid')
        play_intro_menu(channel)

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    channel = channel_obj.get('channel')
    print "Channel %s has entered the application" % channel.json.get('name')

    channel.on_event('ChannelDtmfReceived', on_dtmf_received)
    play_intro_menu(channel)

def stasis_end_cb(channel, ev):
    """Handler for StasisEnd event"""

    print "%s has left the application" % channel.json.get('name')
    cancel_timeout(channel)

client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('StasisEnd', stasis_end_cb)

```



```
client.run(apps='channel-aa')
```

### channel-aa.py in action

The following shows the output of `channel-aa.py` when a PJSIP channel presses 1, 2, 8, then times out. Finally they hang up.

```
Channel PJSIP/alice-00000001 has entered the application
Channel PJSIP/alice-00000001 entered 1
Channel PJSIP/alice-00000001 entered 2
Channel PJSIP/alice-00000001 entered 8
Channel PJSIP/alice-00000001 entered an invalid option!
Channel PJSIP/alice-00000001 stopped paying attention...
PJSIP/alice-00000001 has left the application
```

### JavaScript (Node.js)

As this example is a bit larger, how the code is written and structured is broken up into two phases:

1. Constructing the menu and handling its state as the user presses buttons.
2. Actually handling the button presses from the user.

The full source code for this example immediately follows the walk through.

#### Playing the menu

Unlike Playback, which can chain multiple sounds together and play them back in one continuous operation, ARI treats all sound files being played as separate operations. It will queue each sound file up to be played on the channel, and hand back the caller an object to control the operation of that single sound file. The menu announcement for the attendant has the following requirements:

1. Playback the options for the user
2. If the user presses a DTMF key, cancel the playback of the options and handle the request
3. If the user presses an invalid DTMF key, let them know and restart the menu
4. If the user doesn't press anything, wait 10 seconds, ask them if they are still present, and restart the menu

The second requirement makes this a bit more challenging: when the user presses a DTMF key, we want to cancel whatever sound file is currently being played back and immediately handle their request. We thus have to maintain some state in our application about what sound file is currently being played so that we can cancel the correct playback. We also don't want to queue up all of the sounds immediately - we'd have to walk through all of the queued up sounds and cancel each one - that'd be annoying! Instead, we only want to start the next sound in our prompt when the previous has completed.

To start, we'll define an object to represent the menu at the top of our script that defines sounds that make up the initial menu prompt as well as valid DTMF options for the menu:

```
var menu = {
  // valid menu options
  options: [1, 2],
  // note: this uses the 'extra' sounds package
  sounds: ['sound:press-1', 'sound:or', 'sound:press-2']
};
```

To start with, we'll register a callback to handle a `StasisStart` and `StasisEnd` event on any channel that enters into our application:

```

function stasisStart(event, channel) {
  console.log('Channel %s has entered the application', channel.name);

  channel.on('ChannelDtmfReceived', dtmfReceived);

  channel.answer(function(err) {
    if (err) {
      throw err;
    }
    playIntroMenu(channel);
  });
}

// Handler for StasisEnd event
function stasisEnd(event, channel) {
  console.log('Channel %s has left the application', channel.name);

  // clean up listeners
  channel.removeListener('ChannelDtmfReceived', dtmfReceived);
  cancelTimeout(channel);
}

```

Note that we register a callback to handle ChannelDtmfReceived events on a channel entering our application in StasisStart and then unregister that callback on StasisEnd. For long running, non-trivial applications, this allows the JavaScript garbage collector to clean up our callback. This is important since every channel entering into our application will register its own copy of the callback which is not be garbage collected until it is unregistered.

We'll cover the DTMF callback handler shortly, but first we'll cover writing functions to handle playing the menu prompt

First we'll write a function to initialize a new instance of our menu; playIntroMenu.

Since we'll want to maintain some state, we'll create a small object to do that for us. This object will keep track of the following:

1. The current sound being played
2. The current Playback object being played
3. Whether or not this menu instance is done

It's useful to have this data, as we may cancel the menu half-way through and want to take one set of actions, or we may play all the sounds that make up the menu prompt and start a different set of actions.

```

var state = {
  currentSound: menu.sounds[0],
  currentPlayback: undefined,
  done: false
};

```

playIntroMenu will start the menu on a channel. It will simply initialize the state of the menu, and get the ball rolling on the channel by calling queueUpSound which is a nested function within playIntroMenu.

```

function playIntroMenu(channel) {
  var state = {
    currentSound: menu.sounds[0],
    currentPlayback: undefined,
    done: false
  };

  channel.on('ChannelDtmfReceived', cancelMenu);
  channel.on('StasisEnd', cancelMenu);
  queueUpSound();
  ...
}

```

We'll cover cancelMenu shortly, but first let's discuss queueUpSound. queueUpSound will be responsible for starting the next sound file on the channel and handling the manipulation of that sound file. queueUpSound is also responsible for starting a timeout once all sounds for the menu prompt have completed to handle reminding the user that they must choose a menu option. We'll cover that part shortly but first, we'll cover handling progress

through the sounds that make up the menu prompt. We first initiate playback on the current sound in the sequence. We then register a callback to handle that playback finishing, which will trigger `queueUpSound` to be called again, moving on to the next sound in the sequence. Finally, we update the state object to reflect the next sound to be played in the menu prompt sequence.

```
function queueUpSound() {
  if (!state.done) {
    // have we played all sounds in the menu?
    if (!state.currentSound) {
      var timer = setTimeout(stillThere, 10 * 1000);
      timers[channel.id] = timer;
    } else {
      var playback = client.Playback();
      state.currentPlayback = playback;

      channel.play({media: state.currentSound}, playback, function(err) {
        // ignore errors
      });
      playback.once('PlaybackFinished', function(event, playback) {
        queueUpSound();
      });

      var nextSoundIndex = menu.sounds.indexOf(state.currentSound) + 1;
      state.currentSound = menu.sounds[nextSoundIndex];
    }
  }
}
```

Notice that when registering our `PlaybackFinished` callback handler, we use the `once` method on the resource instance instead of `on`. This ensures that the callback will be invoked once and then automatically be unregistered. Since a `PlaybackFinished` event will only be invoked once for a given `Playback` instance, it makes sense to use this method which will also enable the callback to be garbage collected once it has been invoked.

`queueUpSound` will play back the menu sounds, but it doesn't handle cancelling the menu, time-outs, or other conditions. To do that, we're going to need more information from Asterisk.

### Canceling the menu

When the user presses a DTMF key, we want to stop the current playback and end the menu. To do that, we'll need to subscribe for DTMF events from the channel. We'll define a new handler function, `cancelMenu`, and tell `ari-client` to call it when a DTMF key is received via the `ChannelDtmfReceived` event. We don't really care about the digit here - we just want to cancel the menu. In the handler function, we'll set `state.done` to `true`, then tell the `currentPlayback` to stop.

We should also stop the menu when the channel is hung up. To do this we'll subscribe to the `StasisEnd` event as well and register `cancelMenu` as its callback handler:

```
function cancelMenu() {
  state.done = true;
  if (state.currentPlayback) {
    state.currentPlayback.stop(function(err) {
      // ignore errors
    });
  }

  // remove listeners as future calls to playIntroMenu will create new ones
  channel.removeListener('ChannelDtmfReceived', cancelMenu);
  channel.removeListener('StasisEnd', cancelMenu);
}
```

Note that once the `cancelMenu` callback is invoked, we unregister both the `ChannelDtmfReceived` and `StasisEnd` events. This is performed so that once this particular menu instance stops, we do not leave registered callbacks behind that will never be garbage collected.

### Timing out

Now we can cancel the menu, but we also need to restart it if the user doesn't do anything. We can use a JavaScript timeout to start a timer if we're finished playing sounds *and* we got to the end of the sound prompt sequence. We don't want to start the timer if the user pressed a DTMF key - in that case, we would have stopped the menu early and we should be off handling their DTMF key press. The timer will call `stillThere`, which will play back a

"are you still there?" prompt, then restart the menu.

```
function stillThere() {
  console.log('Channel %s stopped paying attention...', channel.name);

  channel.play({media: 'sound:are-you-still-there'}, function(err) {
    if (err) {
      throw err;
    }

    playIntroMenu(channel);
  });
}
```

Now that we've introduced timers, we know we're going to need to stop them if the user does something. We'll store the timers in an object indexed by channel ID, so we can get them from various parts of the script:

```
var timers = {};
```

### Handling the DTMF options

While we now have code that plays back the menu to the user, we actually have to implement the attendant menu still. Earlier in our example we registered a callback handler for a ChannelDtmfReceived event on a channel that enters into our application. In that callback, we need to do the following:

1. Cancel any timers associated with the channel. Note that we don't need to stop the playback of the menu, as the menu function `queueUpSound` already registers a handler for that event and cancels the menu when it gets any digit.
2. Actually handle the digit, if the digit is a 1 or a 2.
3. If the digit isn't supported, play a prompt informing the user that their option was invalid, and re-play the menu.

The following implements these three items, deferring processing of the valid options to a separate function.

```
function dtmfReceived(event, channel) {
  clearTimeout(channel);
  var digit = parseInt(event.digit);

  console.log('Channel %s entered %d', channel.name, digit);

  // will be non-zero if valid
  var valid = ~menu.options.indexOf(digit);
  if (valid) {
    handleDtmf(channel, digit);
  } else {
    console.log('Channel %s entered an invalid option!', channel.name);

    channel.play({media: 'sound:option-is-invalid'}, function(err, playback) {
      if (err) {
        throw err;
      }

      playIntroMenu(channel);
    });
  }
}
```

Canceling the timer is done in a fashion similar to other examples. If the channel has a JavaScript timeout associated with it, we cancel the timer and remove it from the object.

```
function cancelTimeout(channel) {
  var timer = timers[channel.id];

  if (timer) {
    clearTimeout(timer);
    delete timers[channel.id];
  }
}
```

Finally, we need to actually do *something* when the user presses a 1 or a 2. We could do anything here - but in our case, we're merely going to play back the number that they pressed and restart the menu.

```
function handleDtmf(channel, digit) {
  var parts = ['sound:you-entered', util.format('digits:%s', digit)];
  var done = 0;

  var playback = client.Playback();
  channel.play({media: 'sound:you-entered'}, playback, function(err) {
    // ignore errors
    channel.play({media: util.format('digits:%s', digit)}, function(err) {
      // ignore errors
      playIntroMenu(channel);
    });
  });
}
```

#### channel-aa.js

The full source for channel-aa.js is shown below:

```
channel-aa.js
/*jshint node:true*/
'use strict';

var ari = require('ari-client');
var util = require('util');

ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

var menu = {
  // valid menu options
  options: [1, 2],
  // note: this uses the 'extra' sounds package
  sounds: ['sound:press-1', 'sound:or', 'sound:press-2']
};

var timers = {};

// Handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }

  client.on('StasisStart', stasisStart);
  client.on('StasisEnd', stasisEnd);
```

```

// Handler for StasisStart event
function stasisStart(event, channel) {
  console.log('Channel %s has entered the application', channel.name);

  channel.on('ChannelDtmfReceived', dtmfReceived);

  channel.answer(function(err) {
    if (err) {
      throw err;
    }

    playIntroMenu(channel);
  });
}

// Handler for StasisEnd event
function stasisEnd(event, channel) {
  console.log('Channel %s has left the application', channel.name);

  // clean up listeners
  channel.removeListener('ChannelDtmfReceived', dtmfReceived);
  cancelTimeout(channel);
}

// Main DTMF handler
function dtmfReceived(event, channel) {
  cancelTimeout(channel);
  var digit = parseInt(event.digit);

  console.log('Channel %s entered %d', channel.name, digit);

  // will be non-zero if valid
  var valid = ~menu.options.indexOf(digit);
  if (valid) {
    handleDtmf(channel, digit);
  } else {
    console.log('Channel %s entered an invalid option!', channel.name);

    channel.play({media: 'sound:option-is-invalid'}, function(err, playback) {
      if (err) {
        throw err;
      }

      playIntroMenu(channel);
    });
  }
}

/**
 * Play our intro menu to the specified channel
 *
 * Since we want to interrupt the playback of the menu when the user presses
 * a DTMF key, we maintain the state of the menu via the MenuState object.
 * A menu completes in one of two ways:
 * (1) The user hits a key
 * (2) The menu finishes to completion
 *
 * In the case of (2), a timer is started for the channel. If the timer pops,
 * a prompt is played back and the menu restarted.

```

```

**/
function playIntroMenu(channel) {
  var state = {
    currentSound: menu.sounds[0],
    currentPlayback: undefined,
    done: false
  };

  channel.on('ChannelDtmfReceived', cancelMenu);
  channel.on('StasisEnd', cancelMenu);
  queueUpSound();

  // Cancel the menu, as the user did something
  function cancelMenu() {
    state.done = true;
    if (state.currentPlayback) {
      state.currentPlayback.stop(function(err) {
        // ignore errors
      });
    }

    // remove listeners as future calls to playIntroMenu will create new ones
    channel.removeListener('ChannelDtmfReceived', cancelMenu);
    channel.removeListener('StasisEnd', cancelMenu);
  }

  // Start up the next sound and handle whatever happens
  function queueUpSound() {
    if (!state.done) {
      // have we played all sounds in the menu?
      if (!state.currentSound) {
        var timer = setTimeout(stillThere, 10 * 1000);
        timers[channel.id] = timer;
      } else {
        var playback = client.Playback();
        state.currentPlayback = playback;

        channel.play({media: state.currentSound}, playback, function(err) {
          // ignore errors
        });
        playback.once('PlaybackFinished', function(event, playback) {
          queueUpSound();
        });

        var nextSoundIndex = menu.sounds.indexOf(state.currentSound) + 1;
        state.currentSound = menu.sounds[nextSoundIndex];
      }
    }
  }

  // plays are-you-still-there and restarts the menu
  function stillThere() {
    console.log('Channel %s stopped paying attention...', channel.name);

    channel.play({media: 'sound:are-you-still-there'}, function(err) {
      if (err) {
        throw err;
      }
    }
  }
}

```

```

        playIntroMenu(channel);
    });
}

// Cancel the timeout for the channel
function cancelTimeout(channel) {
    var timer = timers[channel.id];

    if (timer) {
        clearTimeout(timer);
        delete timers[channel.id];
    }
}

// Handler for channel pressing valid option
function handleDtmf(channel, digit) {
    var parts = ['sound:you-entered', util.format('digits:%s', digit)];
    var done = 0;

    var playback = client.Playback();
    channel.play({media: 'sound:you-entered'}, playback, function(err) {
        // ignore errors
        channel.play({media: util.format('digits:%s', digit)}, function(err) {
            // ignore errors
            playIntroMenu(channel);
        });
    });
}
}

```



```
client.start('channel-aa');  
}
```

#### **channel-aa.js in action**

The following shows the output of `channel-aa.js` when a PJSIP channel presses 1, 2, 8, then times out. Finally they hang up.

```
Channel PJSIP/alice-00000001 has entered the application  
Channel PJSIP/alice-00000001 entered 1  
Channel PJSIP/alice-00000001 entered 2  
Channel PJSIP/alice-00000001 entered 8  
Channel PJSIP/alice-00000001 entered an invalid option!  
Channel PJSIP/alice-00000001 stopped paying attention...  
PJSIP/alice-00000001 has left the application
```

# Introduction to ARI and Bridges

## Asterisk Bridges

In Asterisk, bridges can be thought of as a container for channels that form paths of communication between the channels contained within them. They can be used to pass media back and forth between the channels, as well as to play media to the various channels in a variety of ways.



### More Information

For more information on bridges in Asterisk, see [Bridges](#).

## Bridges in a Stasis Application

### Bridge Types

When a bridge is created through ARI, there are a number of attributes that can be specified that determine how the bridge mixes media between its participants. These include:

- `mixing` - specify that media should be passed between all channels in the bridge. This attribute cannot be used with `holding`.
- `dtmf_events` - specify that media should be decoded within Asterisk so that DTMF can be recognized. If this is not specified, then DTMF events may not be raised due to the media being passed directly between the channels in the bridge. This attribute only impacts how media is mixed when the `mixing` attribute is used.
- `proxy_media` - specify that media should always go through Asterisk, even if it could be redirected between clients. This attribute only impacts how media is mixed when the `mixing` attribute is used.

- `holding` - specify that the channels in the bridge should be entertained with some media. Channels in the bridge have two possible roles: a participant or an announcer. Media between participant channels is not shared; media from an announcer channel is played to all participant channels.

Depending on the combination of attributes selected when a bridge is created, different mixing technologies may be used for the participants in the bridge. Asterisk will attempt to use the most performant mixing technology that it can based on the channel types in the bridge, subject to the attributes specified when the bridge was created.

### Subscription Model

Unlike channels, bridges in a Stasis application are not automatically subscribed to for events. In order to receive events concerning events for a given bridge, the [applications](#) resource must be used to subscribe to the bridge via the `POST - /applications/{app_name}/subscription` operation. Events related to channels entering and leaving bridges will be sent without the need to subscribe to them since they are related to a channel in a Stasis application.

## Example: Interacting with Bridges

For this example, we're going to write an ARI application that will do the following:

1. When it connects, it will print out the names of all existing holding bridges. If there are no existing holding bridges, it will create one.
2. When a channel enters into its Stasis application, it will be added to the holding bridge and music on hold will be started on the bridge.

### Dialplan

The dialplan for this will be very straight forward: a simple extension that drops a channel into Stasis.

## On This Page

- [Asterisk Bridges](#)
- [Bridges in a Stasis Application](#)
  - [Bridge Types](#)
  - [Subscription Model](#)
- [Example: Interacting with Bridges](#)
  - [Dialplan](#)
  - [Python](#)
    - [b](#)
    - [b](#)
  - [JavaScript \(Node.js\)](#)
    - [b](#)
    - [b](#)

## Bridges in Depth

- [ARI and Bridges: Holding Bridges](#)
- [ARI and Bridges: Basic Mixing Bridges](#)
- [ARI and Bridges: Bridge Operations](#)

## extensions.conf

```
[default]

exten => 1000,1,NoOp()
same =>     n,Answer()
same =>     n,Stasis(bridge-hold)
same =>     n,Hangup()
```

## Python

For our Python examples, we will rely primarily on the `ari-py` library. Because the `ari` library will emit useful information using Python logging, we should go ahead and set that up as well - for now, a `basicConfig` with `ERROR` messages displayed should be sufficient. Finally, we'll need to get a client made by initiating a connection to Asterisk. This occurs using the `ari.connect` method, where we have to specify three things:

1. The HTTP base URI of the Asterisk server to connect to. Here, we assume that this is running on the same machine as the script, and that we're using the default port for Asterisk's HTTP server - 8088.
2. The username of the ARI user account to connect as. In this case, we're specifying it as `asterisk`.
3. The password for the ARI user account. In this case, that's `asterisk`.



Modify the connection credentials as appropriate for your server, although many examples will use these credentials.

**Please don't use these credentials in production systems!**

```
#!/usr/bin/env python

import ari
import logging

logging.basicConfig(level=logging.ERROR)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')
```

Once we've made our connection, our first task is to look for an existing holding bridge - if there is no existing holding bridge - we need to create it. The `bridges` resource has an operation for listing existing bridges - `GET /bridges`. Using `ari-py` we need to use the operation nickname - `list`. We can then use another `bridges` resource operation to create a holding bridge if none was found - `POST /bridges`. Using `ari-py`, we need to use the operation nickname - `create`.

```
# find or create a holding bridge
bridges = [candidate for candidate in client.bridges.list() if
            candidate.json['bridge_type'] == 'holding']
if bridges:
    bridge = bridges[0]
    print "Using bridge %s" % bridge.id
else:
    bridge = client.bridges.create(type='holding')
    print "Created bridge %s" % bridge.id
```

The `GET /channels` operation returns back a list of `Bridge` resources. Those resources, however, are returned as JSON from the operation, and while the `ari-py` library converts the `uniqueid` of those into an attribute on the object, it leaves the rest of them in the JSON dictionary.

Our next step involves adding channels that enter our Stasis application to the bridge we either found or created and signaling when a channel leaves our Stasis application. To do that, we need to subscribe for the `StasisStart` and `StasisEnd` events:

```
client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('StasisEnd', stasis_end_cb)
```

We need two handler functions - `stasis_start_cb` for the `StasisStart` event and `stasis_end_cb` for the `StasisEnd` event:

```
def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    channel = channel_obj.get('channel')
    print "Channel %s just entered our application, adding it to bridge %s" % (
        channel.json.get('name'), bridge.id)
    channel.answer()
    bridge.addChannel(channel=channel.id)
    bridge.startMoh()

def stasis_end_cb(channel, ev):
    """Handler for StasisEnd event"""

    print "Channel %s just left our application" % channel.json.get('name')
```

Finally, we need to tell the `client` to run our application. Once we call `client.run`, the websocket connection will be made and our application will wait on events infinitely. We can use `Ctrl+C` to kill it and break the connection.

```
client.run(apps='bridge-hold')
```

### ***bridge-hold.py***

The full source code for `bridge-hold.py` is shown below:

```

#!/usr/bin/env python

import ari
import logging

logging.basicConfig(level=logging.ERROR)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

# find or create a holding bridge
bridges = [candidate for candidate in client.bridges.list() if
            candidate.json['bridge_type'] == 'holding']
if bridges:
    bridge = bridges[0]
    print "Using bridge %s" % bridge.id
else:
    bridge = client.bridges.create(type='holding')
    print "Created bridge %s" % bridge.id

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    channel = channel_obj.get('channel')
    print "Channel %s just entered our application, adding it to bridge %s" % (
        channel.json.get('name'), bridge.id)

    channel.answer()
    bridge.addChannel(channel=channel.id)
    bridge.startMoh()

def stasis_end_cb(channel, ev):
    """Handler for StasisEnd event"""

    print "Channel %s just left our application" % channel.json.get('name')

client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('StasisEnd', stasis_end_cb)

client.run(apps='bridge-hold')

```

### ***bridge-hold.py in action***

Here, we see the output from the `bridge-hold.py` script when a PJSIP channel for endpoint 'alice' enters into the application:

```

asterisk:~$ python bridge-hold.py
Created bridge 79f3ad78-b124-4d7b-a629-a53b7e7f50cd
Channel PJSIP/alice-00000001 just entered our application, adding it to bridge
79f3ad78-b124-4d7b-a629-a53b7e7f50cd
Channel PJSIP/alice-00000001 just left our application

```

### **JavaScript (Node.js)**

For our JavaScript examples, we will rely primarily on the Node.js `ari-client` library. We'll need to get a client made by initiating a connection to Asterisk. This occurs using the `ari.connect` method, where we have to specify four things:

1. The HTTP base URI of the Asterisk server to connect to. Here, we assume that this is running on the same machine as the script,

- and that we're using the default port for Asterisk's HTTP server - 8088.
2. The username of the ARI user account to connect as. In this case, we're specifying it as `asterisk`.
  3. The password for the ARI user account. In this case, that's `asterisk`.
  4. A callback that will be called with an error if one occurred, followed by an instance of an ARI client.



Modify the connection credentials as appropriate for your server, although many examples will use these credentials.

**Please don't use these credentials in production systems!**

```
/*jshint node:true*/
'use strict';

var ari = require('ari-client');
var util = require('util');

ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

// handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }
}
```

Once we've made our connection, our first task is to look for an existing holding bridge - if there is no existing holding bridge - we need to create it. The `bridges` resource has an operation for listing existing bridges - `GET /bridges`. Using `ari-client` we need to use the operation nickname - `list`. We can then use another `bridges` resource operation to create a holding bridge if none was found - `POST /bridges`. Using `ari-client`, we need to use the operation nickname - `create`.

```
// find or create a holding bridge
var bridge = null;
client.bridges.list(function(err, bridges) {
  if (err) {
    throw err;
  }
  bridge = bridges.filter(function(candidate) {
    return candidate.bridge_type === 'holding';
  })[0];
  if (bridge) {
    console.log(util.format('Using bridge %s', bridge.id));
  } else {
    client.bridges.create({type: 'holding'}, function(err, newBridge) {
      if (err) {
        throw err;
      }
      bridge = newBridge;
      console.log(util.format('Created bridge %s', bridge.id));
    });
  }
});
```

The `GET /channels` operation returns back a an error if it occurred and a list of `Bridge` resources. `ari-client` will return a JavaScript object for each `Bridge` resource. Properties such as `bridge_type` can be accessed on the object directly.

Our next step involves adding channels that enter our Stasis application to the bridge we either found or created and signaling when a channel leaves our Stasis application. To do that, we need to subscribe for the `StasisStart` and `StasisEnd` events:

```
client.on('StasisStart', stasisStart);
client.on('StasisEnd', stasisEnd);
```

We need two callback functions - `stasisStart` for the `StasisStart` event and `stasisEnd` for the `StasisEnd` event:

```

// handler for StasisStart event
function stasisStart(event, channel) {
  console.log(util.format(
    'Channel %s just entered our application, adding it to bridge %s',
    channel.name,
    bridge.id));

  channel.answer(function(err) {
    if (err) {
      throw err;
    }

    bridge.addChannel({channel: channel.id}, function(err) {
      if (err) {
        throw err;
      }

      bridge.startMoh(function(err) {
        if (err) {
          throw err;
        }
      });
    });
  });
}

// handler for StasisEnd event
function stasisEnd(event, channel) {
  console.log(util.format(
    'Channel %s just left our application', channel.name));
}

```

Finally, we need to tell the client to start our application. Once we call `client.start`, a websocket connection will be established and the client will emit Node.js events as events come in through the websocket. We can use `Ctrl+C` to kill it and break the connection.

```
client.start('bridge-hold');
```

### **bridge-hold.js**

The full source code for `bridge-hold.js` is shown below:

```

/*jshint node:true*/
'use strict';

var ari = require('ari-client');
var util = require('util');

ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

// handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }

  // find or create a holding bridge
  var bridge = null;
  client.bridges.list(function(err, bridges) {
    if (err) {

```

```

    throw err;
  }

  bridge = bridges.filter(function(candidate) {
    return candidate.bridge_type === 'holding';
  })[0];

  if (bridge) {
    console.log(util.format('Using bridge %s', bridge.id));
  } else {
    client.bridges.create({type: 'holding'}, function(err, newBridge) {
      if (err) {
        throw err;
      }

      bridge = newBridge;
      console.log(util.format('Created bridge %s', bridge.id));
    });
  }
});

// handler for StasisStart event
function stasisStart(event, channel) {
  console.log(util.format(
    'Channel %s just entered our application, adding it to bridge %s',
    channel.name,
    bridge.id));

  channel.answer(function(err) {
    if (err) {
      throw err;
    }

    bridge.addChannel({channel: channel.id}, function(err) {
      if (err) {
        throw err;
      }

      bridge.startMoh(function(err) {
        if (err) {
          throw err;
        }
      });
    });
  });
}

// handler for StasisEnd event
function stasisEnd(event, channel) {
  console.log(util.format(
    'Channel %s just left our application', channel.name));
}

client.on('StasisStart', stasisStart);
client.on('StasisEnd', stasisEnd);

```



```
client.start('bridge-hold');  
}
```

### ***bridge-hold.js in action***

Here, we see the output from the `bridge-hold.js` script when a PJSIP channel for endpoint 'alice' enters into the application:

```
asterisk:~$ node bridge-hold.js  
Created bridge 41208316-174c-4e40-90bb-c45cca6579d4  
Channel PJSIP/alice-00000001 just entered our application, adding it to bridge  
41208316-174c-4e40-90bb-c45cca6579d4  
Channel PJSIP/alice-00000001 just left our application
```

## ARI and Bridges: Holding Bridges

### Holding Bridges

Holding bridges are a special type of bridge in Asterisk. The purpose of a holding bridge is to provide a consistent way to place channels when you want the person on the other end of the channel to wait. Asterisk will mix the media to the channel depending on the type of **role** the channel has within the bridge. Two types of roles are supported:

- **participant** - the default role for channels in a holding bridge. Media from the bridge is played directly to the channels; however, media from the channels is not played to any other participant.
- **announcer** - if a channel joins a holding bridge as an announcer, the bridge will not play media to the channel. However, all media from the channel will be played to all **participant** channels in the bridge simultaneously.

### Adding a channel as a participant

To add a channel as a participant to a holding bridge, you can either not specify a **role** (as the **participant** role is the default role for holding bridges), or you can specify the **participant** role directly:

```
POST /bridges/{bridge_id}/addChannel?channel=12345
POST
/bridges/{bridge_id}/addChannel?channel=12345&role=participant
```

### Adding a channel as an announcer

To add a channel as an announcer to a holding bridge, you must specify a **role** of **announcer**:

```
POST /bridges/{bridge_id}/addChannel?channel=56789&role=announcer
```



#### When is an Announcer channel useful?

If you want to simply play back a media file to all participants in a holding bridge, e.g., "your call is important to us, please keep waiting", you can simply initiate a `/play` operation on the holding bridge itself. That will perform a playback to all participants in the same fashion as an announcer channel.

An announcer channel is particularly useful when there is someone actually on the other end of the channel, as opposed to a pre-recorded message. For example, you may have a call queue supervisor who wants to let everyone who is waiting for an agent that response times are especially long, but to hold on for a bit longer. Jumping into the holding bridge as an announcer adds a small bit of humanity to the dreaded call queue experience!

### Music on hold, media playback, recording, and other such things

When dealing with holding bridges, given the particular media rules and channel roles involved, there are some additional catches that you have to be aware of when manipulating the bridge:

1. Playing music on hold to the bridge will play it for all participants, as well playing media to the bridge. However, you can only do **one** of those operations - you cannot play media to a holding bridge while you are simultaneously playing music on hold to the bridge. Initiating a `/play` operation on a holding bridge should only be done after stopping the music on hold; likewise, starting music on hold on a bridge with a `/play` operation currently in progress will fail.
2. Recording a holding bridge - while possible - is not terribly interesting. Participant media is dropped - so at best, you'll only record the entertainment that was played to the participants.

#### There can be only one!

You cannot have an announcer channel in a holding bridge at the same time that you perform a `play` operation or have music on hold playing to the bridge. Holding bridges do **not** mix the media between announcers. Since media from the `play` operation has to go to all participants, as does your announcer channel's media, the holding bridge will become quite confused about your application's intent.

### On This Page

- [Holding Bridges](#)
  - [Adding a channel as a participant](#)
  - [Adding a channel as an announcer](#)
    - [M](#)
    - [T](#)
- [Example: Infinite wait area](#)
  - [Dialplan](#)
  - [Python](#)
    - [b](#)
    - [b](#)
  - [JavaScript \(Node.js\)](#)
    - [b](#)
    - [b](#)

## Example: Infinite wait area

Now that we all know that holding bridges are perfect for building what many callers fear - the dreaded waiting area of doom - let's make one! This example ARI application will do the following:

1. When a channel enters into the Stasis application, it will be put into a existing holding bridge or a newly created one if none exist.
2. Music on hold will be played on the bridge.
3. Periodically, the `thnk-u-for-patience` sound will be played to the bridge thanking the users for their patience, which they will need since this holding bridge will never progress beyond this point!
4. When a channel leaves a holding bridge, if no other channels remain, the bridge will be destroyed.

This example will use a similar structure to the bridge-hold python example. Unlike that example, however, it will use some form of a timer to perform our periodic announcement to the holding bridge, and when all the channels have left the infinite wait area, we'll destroy the holding bridge (cleaning up resources is always good!)

## Dialplan

For this example, we need to just drop the channel into Stasis, specifying our application:

### extensions.conf

```
exten => 1000,1,NoOp()  
same =>      n,Stasis(bridge-infinite-wait)  
same =>      n,Hangup()
```

## Python

When a channel enters our Stasis application, we first look for an existing holding bridge or create one if none is found. When we create a new bridge, we start music on hold in the bridge and create a timer that will call a callback after 30 seconds. That callback temporarily stops the music on hold, and starts a play operation on the bridge that thanks everyone for their patience. When the play operation finishes, it resumes music on hold.

```

# find or create a holding bridge
holding_bridge = None

# Announcer timer
announcer_timer = None

def find_or_create_bridge():
    """Find our infinite wait bridge, or create a new one

    Returns:
    The one and only holding bridge
    """

    global holding_bridge
    global announcer_timer

    if holding_bridge:
        return holding_bridge

    bridges = [candidate for candidate in client.bridges.list() if
                candidate.json['bridge_type'] == 'holding']
    if bridges:
        bridge = bridges[0]
        print "Using bridge %s" % bridge.id
    else:
        bridge = client.bridges.create(type='holding')
        bridge.startMoh()
        print "Created bridge %s" % bridge.id

    def play_announcement(bridge):
        """Play an announcement to the bridge"""

        def on_playback_finished(playback, ev):
            """Handler for the announcement's PlaybackFinished event"""
            global announcer_timer
            global holding_bridge

            holding_bridge.startMoh()

            announcer_timer = threading.Timer(30, play_announcement,
                                              [holding_bridge])

            announcer_timer.start()

        bridge.stopMoh()
        print "Letting the everyone know we care..."
        thanks_playback = bridge.play(media='sound:thnk-u-for-patience')
        thanks_playback.on_event('PlaybackFinished', on_playback_finished)

    holding_bridge = bridge
    holding_bridge.on_event('ChannelLeftBridge', on_channel_left_bridge)
    # After 30 seconds, let everyone in the bridge know that we care
    announcer_timer = threading.Timer(30, play_announcement, [holding_bridge])
    announcer_timer.start()
    return bridge

```

The function that does this work, `find_or_create_bridge`, is called from our `StasisStart` event handler. The bridge that it returns will have the new channel added to it.

```

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    bridge = find_or_create_bridge()

    channel = channel_obj.get('channel')
    print "Channel %s just entered our application, adding it to bridge %s" % (
        channel.json.get('name'), holding_bridge.id)

    channel.answer()
    bridge.addChannel(channel=channel.id)

```

In the `find_or_create_bridge` function, we also subscribed for the `ChannelLeftBridge` event. We'll add a callback handler for this in that function as well. When the channel leaves the bridge, we'll check to see if there are no more channels in the bridge and - if so - destroy the bridge.

```

def on_channel_left_bridge(bridge, ev):
    """Handler for ChannelLeftBridge event"""
    global holding_bridge
    global announcer_timer

    channel = ev.get('channel')
    channel_count = len(bridge.json.get('channels'))

    print "Channel %s left bridge %s" % (channel.get('name'), bridge.id)
    if holding_bridge.id == bridge.id and channel_count == 0:
        if announcer_timer:
            announcer_timer.cancel()
            announcer_timer = None

        print "Destroying bridge %s" % bridge.id
        holding_bridge.destroy()
        holding_bridge = None

```

### bridge-infinite-wait.py

The full source code for `bridge-infinite-wait.py` is shown below:

```

bridge-infinite-wait.py

#!/usr/bin/env python

import ari
import logging
import threading

logging.basicConfig(level=logging.ERROR)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

# find or create a holding bridge
holding_bridge = None

# Announcer timer
announcer_timer = None

def find_or_create_bridge():

```

```

"""Find our infinite wait bridge, or create a new one

Returns:
The one and only holding bridge
"""

global holding_bridge
global announcer_timer

if holding_bridge:
    return holding_bridge

bridges = [candidate for candidate in client.bridges.list() if
            candidate.json['bridge_type'] == 'holding']
if bridges:
    bridge = bridges[0]
    print "Using bridge %s" % bridge.id
else:
    bridge = client.bridges.create(type='holding')
    bridge.startMoh()
    print "Created bridge %s" % bridge.id

def play_announcement(bridge):
    """Play an announcement to the bridge"""

    def on_playback_finished(playback, ev):
        """Handler for the announcement's PlaybackFinished event"""
        global announcer_timer
        global holding_bridge

        holding_bridge.startMoh()

        announcer_timer = threading.Timer(30, play_announcement,
                                          [holding_bridge])
        announcer_timer.start()

    bridge.stopMoh()
    print "Letting the everyone know we care..."
    thanks_playback = bridge.play(media='sound:thnk-u-for-patience')
    thanks_playback.on_event('PlaybackFinished', on_playback_finished)

def on_channel_left_bridge(bridge, ev):
    """Handler for ChannelLeftBridge event"""
    global holding_bridge
    global announcer_timer

    channel = ev.get('channel')
    channel_count = len(bridge.json.get('channels'))

    print "Channel %s left bridge %s" % (channel.get('name'), bridge.id)
    if holding_bridge.id == bridge.id and channel_count == 0:
        if announcer_timer:
            announcer_timer.cancel()
            announcer_timer = None

        print "Destroying bridge %s" % bridge.id
        holding_bridge.destroy()
        holding_bridge = None

```

```

holding_bridge = bridge
holding_bridge.on_event('ChannelLeftBridge', on_channel_left_bridge)

# After 30 seconds, let everyone in the bridge know that we care
announcer_timer = threading.Timer(30, play_announcement, [holding_bridge])
announcer_timer.start()

return bridge

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart event"""

    bridge = find_or_create_bridge()

    channel = channel_obj.get('channel')
    print "Channel %s just entered our application, adding it to bridge %s" % (
        channel.json.get('name'), holding_bridge.id)

    channel.answer()
    bridge.addChannel(channel=channel.id)

def stasis_end_cb(channel, ev):
    """Handler for StasisEnd event"""

    print "Channel %s just left our application" % channel.json.get('name')

client.on_channel_event('StasisStart', stasis_start_cb)
client.on_channel_event('StasisEnd', stasis_end_cb)

```

```
client.run(apps='bridge-infinite-wait')
```

### **bridge-infinite-wait.py in action**

```
Created bridge 950c4805-c33c-4895-ad9a-2798055e4939  
Channel PJSIP/alice-00000000 just entered our application, adding it to bridge  
950c4805-c33c-4895-ad9a-2798055e4939  
Letting the everyone know we care...  
Channel PJSIP/alice-00000000 left bridge 950c4805-c33c-4895-ad9a-2798055e4939  
Destroying bridge 950c4805-c33c-4895-ad9a-2798055e4939  
Channel PJSIP/alice-00000000 just left our application
```

### **JavaScript (Node.js)**

When a channel enters our Stasis application, we first look for an existing holding bridge or create one if none is found. When we create a new bridge, we start music on hold in the bridge and create a timer that will call a callback after 30 seconds. That callback temporarily stops the music on hold, and starts a play operation on the bridge that thanks everyone for their patience. When the play operation finishes, it resumes music on hold.

In all cases, we add the channel to the bridge via the `joinBridge` function.



```

console.log('Channel %s just entered our application', channel.name);

// find or create a holding bridge
client.bridges.list(function(err, bridges) {
  if (err) {
    throw err;
  }

  var bridge = bridges.filter(function(candidate) {
    return candidate.bridge_type === 'holding';
  })[0];

  if (bridge) {
    console.log('Using bridge %s', bridge.id);
    joinBridge(bridge);
  } else {
    client.bridges.create({type: 'holding'}, function(err, newBridge) {
      if (err) {
        throw err;
      }
      console.log('Created bridge %s', newBridge.id);
      newBridge.startMoh(function(err) {
        if (err) {
          throw err;
        }
      });
      joinBridge(newBridge);

      timer = setTimeout(play_announcement, 30000);

      // callback that will let our users know how much we care
      function play_announcement() {
        console.log('Letting everyone know we care...');
        newBridge.stopMoh(function(err) {
          if (err) {
            throw err;
          }
        });

        var playback = client.Playback();
        newBridge.play({media: 'sound:thnk-u-for-patience'},
          playback, function(err, playback) {
            if (err) {
              throw err;
            }
          });
        playback.once('PlaybackFinished', function(event, playback) {
          newBridge.startMoh(function(err) {
            if (err) {
              throw err;
            }
          });
          timer = setTimeout(play_announcement, 30000);
        });
      });
    });
  }
});
}

```

The `joinBridge` function involves registered a callback for the `ChannelLeftBridge` event and adds the channel to the bridge.

```
function joinBridge(bridge) {
  channel.once('ChannelLeftBridge', function(event, instances) {
    channelLeftBridge(event, instances, bridge);
  });

  bridge.addChannel({channel: channel.id}, function(err) {
    if (err) {
      throw err;
    }
  });
  channel.answer(function(err) {
    if (err) {
      throw err;
    }
  });
}
```

Notice that we use an anonymous function to pass the bridge as an extra parameter to the `ChannelLeftBridge` callback so we can keep the handler at the same level as `joinBridge` and avoid another indentation level of callbacks. Finally, we can handle destroying the bridge when the last channel contained in it has left:

```
// Handler for ChannelLeftBridge event
function channelLeftBridge(event, instances, bridge){
  var holdingBridge = instances.bridge;
  var channel = instances.channel;

  console.log('Channel %s left bridge %s', channel.name, bridge.id);

  if (holdingBridge.id === bridge.id &&
      holdingBridge.channels.length === 0) {

    if (timer) {
      clearTimeout(timer);
    }

    bridge.destroy(function(err) {
      if (err) {
        throw err;
      }
    });
  }
}
```

### **bridge-infinite-wait.js**

The full source code for `bridge-infinite-wait.js` is shown below:

```
bridge-infinite-wait.js
/*jshint node:true*/
'use strict';

var ari = require('ari-client');
var util = require('util');

var timer = null;
ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);
```

```

// handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }

  // handler for StasisStart event
  function stasisStart(event, channel) {
    console.log('Channel %s just entered our application', channel.name);

    // find or create a holding bridge
    client.bridges.list(function(err, bridges) {
      if (err) {
        throw err;
      }

      var bridge = bridges.filter(function(candidate) {
        return candidate.bridge_type === 'holding';
      })[0];

      if (bridge) {
        console.log('Using bridge %s', bridge.id);
        joinBridge(bridge);
      } else {
        client.bridges.create({type: 'holding'}, function(err, newBridge) {
          if (err) {
            throw err;
          }
          console.log('Created bridge %s', newBridge.id);
          newBridge.startMoh(function(err) {
            if (err) {
              throw err;
            }
          });
          joinBridge(newBridge);

          timer = setTimeout(play_announcement, 30000);

          // callback that will let our users know how much we care
          function play_announcement() {
            console.log('Letting everyone know we care...');
            newBridge.stopMoh(function(err) {
              if (err) {
                throw err;
              }
            });

            var playback = client.Playback();
            newBridge.play({media: 'sound:thnk-u-for-patience'},
              playback, function(err, playback) {
                if (err) {
                  throw err;
                }
              });
            playback.once('PlaybackFinished', function(event, playback) {
              newBridge.startMoh(function(err) {
                if (err) {
                  throw err;
                }
              });
            });
          }
        });
      }
    });
  }
}

```

```

        timer = setTimeout(play_announcement, 30000);
    });
    });
}
});
}
});

function joinBridge(bridge) {
    channel.once('ChannelLeftBridge', function(event, instances) {
        channelLeftBridge(event, instances, bridge);
    });

    bridge.addChannel({channel: channel.id}, function(err) {
        if (err) {
            throw err;
        }
    });
    channel.answer(function(err) {
        if (err) {
            throw err;
        }
    });
}

// Handler for ChannelLeftBridge event
function channelLeftBridge(event, instances, bridge) {
    var holdingBridge = instances.bridge;
    var channel = instances.channel;

    console.log('Channel %s left bridge %s', channel.name, bridge.id);

    if (holdingBridge.id === bridge.id &&
        holdingBridge.channels.length === 0) {

        if (timer) {
            clearTimeout(timer);
        }

        bridge.destroy(function(err) {
            if (err) {
                throw err;
            }
        });
    }
}

// handler for StasisEnd event
function stasisEnd(event, channel) {
    console.log('Channel %s just left our application', channel.name);
}

client.on('StasisStart', stasisStart);
client.on('StasisEnd', stasisEnd);

```

```
console.log('starting');
client.start('bridge-infinite-wait');
}
```

### **bridge-infinite-wait.js in action**

The following shows the output of the `bridge-infinite-wait.js` script when a PJSIP channel for `alice` enters the application:

```
Channel PJSIP/alice-00000001 just entered our application
Created bridge 31a4a193-36a7-412b-854b-cf2cf5f90bbd
Letting everyone know we care...
Channel PJSIP/alice-00000001 left bridge 31a4a193-36a7-412b-854b-cf2cf5f90bbd
Channel PJSIP/alice-00000001 just left our application
```

## ARI and Bridges: Basic Mixing Bridges

### Mixing Bridges

In a mixing bridge, Asterisk shares media between all the channels in the bridge. Depending on the attributes the bridge was created with and the types of channels in the bridge, a mixing bridge may attempt to share the media in a variety of ways. They are, in order of best performance to lowest performance:

- Direct packet sharing between devices - when there are two channels in a mixing bridge of similar types, it may be possible to have the media bypass Asterisk completely. In this type of bridge, the channels will pass media directly between each other, and Asterisk will simply monitor the state of the channels. However, because the media is not going through Asterisk, most features - such as recording, speech detection, DTMF, etc. - are not available. The `proxy_media` attribute or the `dtmf_events` attribute will prevent this mixing type from being used.
- Native packet sharing through Asterisk - when there are two channels in a mixing bridge of similar types, but the media cannot flow directly between the devices, Asterisk will attempt to mix the media between the channels by directly passing media from one channel to the other, and vice versa. The media itself is not decoded, and so - much like when the media is directly shared between the devices - Asterisk cannot use many features. The `proxy_media` attribute or the `dtmf_events` attribute will prevent this mixing type from being used.
- Two party mixing - when there are two channels in a mixing bridge, regardless of the channel type, Asterisk will decode the media from each channel and pass it to the other participant. This mixing technology allows for all the various features of Asterisk to be used on the channels while they are in the bridge, but does not necessarily incur any penalties from transcoding.
- Multi-party mixing - when there are more than two channels in a mixing bridge, Asterisk will transcode the media from each participant into signed linear, mix the media from all participants together into a new media frame, then write the media back out to all participants.

At all times, the bridge will attempt to mix the media in the most performant manner possible. As the situation in the bridge changes, Asterisk will switch the mixing technology to the best mixing technology available.

### What Can Happen in a Mixing Bridge

Action	Bridge Response
A bridge is created using <code>POST /bridges</code> , and Alice's channel - which supports having media be directly sent to another device - is added to the bridge using <code>POST /bridges/{bridge_id}/addChannel</code> .	Asterisk picks the basic two-party mixing technology. We don't know yet what other channel is going to join the bridge - it could be anything! - and so Asterisk picks the best one based on the information it currently has.
Bob's channel - which also supports having media be directly sent to another device - also joins the bridge via <code>POST /bridges/{bridge_id}/addChannel</code> .	We have two channels in the bridge now, so Asterisk re-evaluates how the media is mixed. Since both channels support having their media be sent directly to each other, and mixing media that way is more performant than the current mixing technology, Asterisk picks the direct media mixing technology and instructs the channels to tell their devices to send the media to each other.
Carol's channel - which is a DAHDI channel (poor Carol, calling from the PSTN) - is also added to the bridge via <code>POST /bridges/{bridge_id}/addChannel</code> .	Since we now have three channels in the bridge, Asterisk switches the mixing technology to multi-mix. Alice and Bob's media is sent back to Asterisk, and Asterisk mixes the media from Alice, Bob, and Carol together and then sends the new media to each channel.
Eventually, Alice hangs up, leaving only Bob and Carol in the bridge.	Since Alice left, Asterisk switches back to the basic two-party mixing technology. We can't use a native mixing technology, as Bob and Carol's channels are incompatible, but we can use a mixing technology that is less expensive than the multi-mix technology.

### Example: Implementing a basic dial

Dialing can be implemented by using the `POST - /channels` operation and putting both the resulting channel and the original Stasis channel in a mixing bridge to allow media to flow between them. An endpoint should be specified along with the originate operation as well as a Stasis application name. This will cause the dialed channel to enter Stasis, where it can be added to a mixing bridge. It's also a good idea to use Stasis application arguments to flag that the dialed channel was dialed using originate in order to handle it differently from the original channel once it enters into the Stasis application.

### On This Page

- Mixing Bridges
  - What Can Happen in a Mixing Bridge
- Example: Implementing a basic dial
  - Dialplan
  - Python
    - b
    - b
  - JavaScript (Node.js)
    - b
    - b

This example ARI application will do the following:

1. When a channel enters into the Stasis application, a call will be originated to the endpoint specified by the first command line argument to the script.
2. When that channel enters into the Stasis application, a mixing bridge will be created and the two channels will be put in it so that media can flow between them.
3. If either channel hangs up, the other channel will also be hung up.
4. Once the dialed channel exists the Stasis application, the mixing bridge will be destroyed.

## Dialplan

For this example, we'll use a Stasis application that species not only the application - `bridge-dial` - but also:

- Whether or not the channel is `inbound` or a `dialed` channel.
- If the channel is `inbound`, the endpoint to dial.

As an example, here is a dialplan that dials the `PJSIP/bob` endpoint:

```
extensions.conf
-----
exten => 1000,1,NoOp()
same =>      n,Stasis(bridge-dial,inbound,PJSIP/bob)
same =>      n,Hangup()
```

## Python

As is typical for an ARI application, we'll start off by implementing a callback handler for the `StasisStart` event. In this particular case, our callback handler will be called in two conditions:

1. When an inbound channel enters into the `Stasis` dialplan application. In that case, we'll want to initiate the outbound dial.
2. When an outbound channel answers. In that case, we'll want to defer processing to a different callback handler and - in that handler - initiate the bridging of the two channels.

In our `StasisStart` callback handler, we can expect to have two pieces of information passed to the application:

1. The "type" of channel entering the callback handler. In this case, we expect the type to be either `inbound` or `dialed`.
2. If the "type" of channel is `inbound`, we expect the second argument to be the endpoint to dial.

The following code shows the `StasisStart` callback handler for the `inbound` channel. Note that if the "type" is not `inbound`, we defer processing to another callback handler. We also tell the inbound channel to start ringing via the `ring` operation, and initiate an outbound dial by creating a new channel to the endpoint specified. Finally, we subscribe to the `StasisEnd` event for both channels, and instruct them to call a `safe_hangup` function on the opposing channel. This ensures that if either party hangs up, we hang up the person they were talking to. We'll show the implementation of that function shortly.



### Be careful of errors!

Note that we wrap the origination with a `try / except` block, in case the endpoint provided by the dialplan doesn't exist. When taking in input from a user or from the Asterisk dialplan, it is always good to be mindful of possible errors.

```

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart"""

    channel = channel_obj.get('channel')
    channel_name = channel.json.get('name')
    args = ev.get('args')

    if not args:
        print "Error: {} didn't provide any arguments!".format(channel_name)
        return

    if args and args[0] != 'inbound':
        # Only handle inbound channels here
        return

    if len(args) != 2:
        print "Error: {} didn't tell us who to dial".format(channel_name)
        channel.hangup()
        return

    print "{} entered our application".format(channel_name)
    channel.ring()

    try:
        print "Dialing {}".format(args[1])
        outgoing = client.channels.Originate(endpoint=args[1],
                                             app='bridge-dial',
                                             appArgs='dialed')

    except requests.HTTPError:
        print "Whoops, pretty sure %s wasn't valid" % args[1]
        channel.hangup()
        return

    channel.on_event('StasisEnd', lambda *args: safe_hangup(outgoing))
    outgoing.on_event('StasisEnd', lambda *args: safe_hangup(channel))

```

The `safe_hangup` function referenced above simply does a "safe" hangup on the channel provided. This is because it is entirely possible for both parties to hang up nearly simultaneously. Since our Python code is running in a separate process from Asterisk, we may be processing the hang up of the first party and instruct Asterisk to hang up the second party when they are already technically hung up! Again, it is always a good idea to view the processing of a communications application in an asynchronous fashion: we live in an asynchronous world, and a user can take an action at any moment in time.

```

def safe_hangup(channel):
    """Safely hang up the specified channel"""
    try:
        channel.hangup()
        print "Hung up {}".format(channel.json.get('name'))
    except requests.HTTPError as e:
        if e.response.status_code != requests.codes.not_found:
            raise e

```

We now have to handle the outbound channel when it answers. Currently, when it answers it will be immediately placed in our Stasis application, which will call the `stasis_start_cb` we previously defined. While we could have some additional `if / else` blocks in that handler, we can also just apply a `StasisStart` callback to the outbound channel after we create it, and handle its entrance separately.

When the outbound channel is answered, we need to do the following:

1. Answer the inbound channel.
2. Create a new mixing bridge.
3. Put both channels into the mixing bridge.
4. Ensure we destroy the mixing bridge when either channel leaves our application.



This is shown in the following code:

```
def outgoing_start_cb(channel_obj, ev):
    """StasisStart handler for our dialed channel"""

    print "{} answered; bridging with {}".format(outgoing.json.get('name'),
                                                  channel.json.get('name'))

    channel.answer()

    bridge = client.bridges.create(type='mixing')
    bridge.addChannel(channel=[channel.id, outgoing.id])

    # Clean up the bridge when done
    channel.on_event('StasisEnd', lambda *args:
                    safe_bridge_destroy(bridge))
    outgoing.on_event('StasisEnd', lambda *args:
                    safe_bridge_destroy(bridge))

    outgoing.on_event('StasisStart', outgoing_start_cb)
```

Note that the `safe_bridge_destroy` function is similar to the `safe_hangup` function, except that it attempts to safely destroy the mixing bridge, as opposed to hanging up the other party.

#### **bridge-dial.py**

The full source code for `bridge-dial.py` is shown below:

```
basic-dial.py
#!/usr/bin/env python

import logging
import requests
import ari

logging.basicConfig(level=logging.ERROR)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

def safe_hangup(channel):
    """Safely hang up the specified channel"""
    try:
        channel.hangup()
        print "Hung up {}".format(channel.json.get('name'))
    except requests.HTTPError as e:
        if e.response.status_code != requests.codes.not_found:
            raise e

def safe_bridge_destroy(bridge):
    """Safely destroy the specified bridge"""
    try:
        bridge.destroy()
    except requests.HTTPError as e:
        if e.response.status_code != requests.codes.not_found:
            raise e
```

```

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart"""

    channel = channel_obj.get('channel')
    channel_name = channel.json.get('name')
    args = ev.get('args')

    if not args:
        print "Error: {} didn't provide any arguments!".format(channel_name)
        return

    if args and args[0] != 'inbound':
        # Only handle inbound channels here
        return

    if len(args) != 2:
        print "Error: {} didn't tell us who to dial".format(channel_name)
        channel.hangup()
        return

    print "{} entered our application".format(channel_name)
    channel.ring()

    try:
        print "Dialing {}".format(args[1])
        outgoing = client.channels.originate(endpoint=args[1],
                                             app='bridge-dial',
                                             appArgs='dialed')

    except requests.HTTPError:
        print "Whoops, pretty sure %s wasn't valid" % args[1]
        channel.hangup()
        return

    channel.on_event('StasisEnd', lambda *args: safe_hangup(outgoing))
    outgoing.on_event('StasisEnd', lambda *args: safe_hangup(channel))

    def outgoing_start_cb(channel_obj, ev):
        """StasisStart handler for our dialed channel"""

        print "{} answered; bridging with {}".format(outgoing.json.get('name'),
                                                       channel.json.get('name'))

        channel.answer()

        bridge = client.bridges.create(type='mixing')
        bridge.addChannel(channel=[channel.id, outgoing.id])

        # Clean up the bridge when done
        channel.on_event('StasisEnd', lambda *args:
                        safe_bridge_destroy(bridge))
        outgoing.on_event('StasisEnd', lambda *args:
                        safe_bridge_destroy(bridge))

    outgoing.on_event('StasisStart', outgoing_start_cb)

client.on_channel_event('StasisStart', stasis_start_cb)

```

```
client.run(apps='bridge-dial')
```

### bridge-dial.py in action

The following shows the output of the `bridge-dial.py` script when a PJSIP channel for `alice` enters the application and dials a PJSIP channel for `bob`:

```
PJSIP/Alice-00000001 entered our application
Dialing PJSIP/Bob
PJSIP/Bob-00000002 answered: bridging with PJSIP/Alice-00000001
Hung up PJSIP/Bob-00000002
```

### JavaScript (Node.js)

This example shows how to use anonymous functions to call functions with extra parameters that would otherwise require a closer. This can be done to reduce the number of nested callbacks required to implement the flow of an application. First, we look for an application argument in our `StasisStart` event callback to ensure that we will only originate a call if the channel entering Stasis is a channel that dialed our application extension we defined in the `extensions.conf` file above. We then play a sound on the channel asking the caller to wait while they are being connected and call the `originate()` function to process down the application flow:

```
function stasisStart(event, channel) {
  // ensure the channel is not a dialed channel
  var dialed = event.args[0] === 'dialed';

  if (!dialed) {
    channel.answer(function(err) {
      if (err) {
        throw err;
      }

      console.log('Channel %s has entered our application', channel.name);

      var playback = client.Playback();
      channel.play({media: 'sound:pls-wait-connect-call'},
        playback, function(err, playback) {
          if (err) {
            throw err;
          }
        });

      originate(channel);
    });
  }
}
```

We then prepare an object with a locally generate `id` for the dialed channel and register event callbacks either channels hanging up and the dialed channel entering into the Stasis application. We then originate a call to the endpoint specified by the first command line argument to the script passing in a Stasis application argument of `dialed` so we can skip the dialed channel when the original `StasisStart` event callback fires for it:

```

function originate(channel) {
  var dialed = client.Channel();

  channel.on('StasisEnd', function(event, channel) {
    hangupDialed(channel, dialed);
  });

  dialed.on('ChannelDestroyed', function(event, dialed) {
    hangupOriginal(channel, dialed);
  });

  dialed.on('StasisStart', function(event, dialed) {
    joinMixingBridge(channel, dialed);
  });

  dialed.originate(
    {endpoint: process.argv[2], app: 'bridge-dial', appArgs: 'dialed'},
    function(err, dialed) {
      if (err) {
        throw err;
      }
    }
  );
}

```

We then handle either channel hanging up by hanging up the other channel. Note that we skip any errors that occur on hangup since it is possible that the channel we are attempting to hang up is the one that has already left and would result in an HTTP error as it is no longer a Stasis channel:

```

function hangupDialed(channel, dialed) {
  console.log(
    'Channel %s left our application, hanging up dialed channel %s',
    channel.name, dialed.name);

  // hangup the other end
  dialed.hangup(function(err) {
    // ignore error since dialed channel could have hung up, causing the
    // original channel to exit Stasis
  });
}

// handler for the dialed channel hanging up so we can gracefully hangup the
// other end
function hangupOriginal(channel, dialed) {
  console.log('Dialed channel %s has been hung up, hanging up channel %s',
    dialed.name, channel.name);

  // hangup the other end
  channel.hangup(function(err) {
    // ignore error since original channel could have hung up, causing the
    // dialed channel to exit Stasis
  });
}

```

We then handle the StasisStart event for the dialed channel by registered an event callback for the StasisEnd event on the dialed channel, answer that answer, creating a new mixing bridge, and finally calling a function to add the two channels to the new bridge:

```
function joinMixingBridge(channel, dialed) {
  var bridge = client.Bridge();

  dialed.on('StasisEnd', function(event, dialed) {
    dialedExit(dialed, bridge);
  });

  dialed.answer(function(err) {
    if (err) {
      throw err;
    }
  });

  bridge.create({type: 'mixing'}, function(err, bridge) {
    if (err) {
      throw err;
    }

    console.log('Created bridge %s', bridge.id);

    addChannelsToBridge(channel, dialed, bridge);
  });
}
```

We then handle the dialed channel exiting the Stasis application by destroying the mixing bridge:

```
function dialedExit(dialed, bridge) {
  console.log(
    'Dialed channel %s has left our application, destroying bridge %s',
    dialed.name, bridge.id);

  bridge.destroy(function(err) {
    if (err) {
      throw err;
    }
  });
}
```

Finally, the function that was called earlier by the callback handling the StasisStart event for the dialed channel adds the two channels to the mixing bridge which allows media to flow between the two channels:

```
function addChannelsToBridge(channel, dialed, bridge) {
  console.log('Adding channel %s and dialed channel %s to bridge %s',
    channel.name, dialed.name, bridge.id);

  bridge.addChannel({channel: [channel.id, dialed.id]}, function(err) {
    if (err) {
      throw err;
    }
  });
}
```

### bridge-dial.js

The full source code for `bridge-dial.js` is shown below:

```
bridge-dial.js
/*jshint node:true*/
```

```

'use strict';

var ari = require('ari-client');
var util = require('util');

// ensure endpoint was passed in to script
if (!process.argv[2]) {
  console.error('usage: node bridge-dial.js endpoint');
  process.exit(1);
}

ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

// handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }

  // handler for StasisStart event
  function stasisStart(event, channel) {
    // ensure the channel is not a dialed channel
    var dialed = event.args[0] === 'dialed';

    if (!dialed) {
      channel.answer(function(err) {
        if (err) {
          throw err;
        }

        console.log('Channel %s has entered our application', channel.name);

        var playback = client.Playback();
        channel.play({media: 'sound:pls-wait-connect-call'},
          playback, function(err, playback) {
            if (err) {
              throw err;
            }
          });

        originate(channel);
      });
    }
  }

  function originate(channel) {
    var dialed = client.Channel();

    channel.on('StasisEnd', function(event, channel) {
      hangupDialed(channel, dialed);
    });

    dialed.on('ChannelDestroyed', function(event, dialed) {
      hangupOriginal(channel, dialed);
    });

    dialed.on('StasisStart', function(event, dialed) {
      joinMixingBridge(channel, dialed);
    });
  }
}

```

```

dialled.originate(
  {endpoint: process.argv[2], app: 'bridge-dial', appArgs: 'dialled'},
  function(err, dialled) {
    if (err) {
      throw err;
    }
  });
}

// handler for original channel hanging up so we can gracefully hangup the
// other end
function hangupDialled(channel, dialled) {
  console.log(
    'Channel %s left our application, hanging up dialled channel %s',
    channel.name, dialled.name);

  // hangup the other end
  dialled.hangup(function(err) {
    // ignore error since dialled channel could have hung up, causing the
    // original channel to exit Stasis
  });
}

// handler for the dialled channel hanging up so we can gracefully hangup the
// other end
function hangupOriginal(channel, dialled) {
  console.log('Dialled channel %s has been hung up, hanging up channel %s',
    dialled.name, channel.name);

  // hangup the other end
  channel.hangup(function(err) {
    // ignore error since original channel could have hung up, causing the
    // dialled channel to exit Stasis
  });
}

// handler for dialled channel entering Stasis
function joinMixingBridge(channel, dialled) {
  var bridge = client.Bridge();

  dialled.on('StasisEnd', function(event, dialled) {
    dialledExit(dialled, bridge);
  });

  dialled.answer(function(err) {
    if (err) {
      throw err;
    }
  });

  bridge.create({type: 'mixing'}, function(err, bridge) {
    if (err) {
      throw err;
    }

    console.log('Created bridge %s', bridge.id);

    addChannelsToBridge(channel, dialled, bridge);
  });
}

```

```

    });
  }

  // handler for the dialed channel leaving Stasis
  function dialedExit(dialed, bridge) {
    console.log(
      'Dialed channel %s has left our application, destroying bridge %s',
      dialed.name, bridge.id);

    bridge.destroy(function(err) {
      if (err) {
        throw err;
      }
    });
  }
}

// handler for new mixing bridge ready for channels to be added to it
function addChannelsToBridge(channel, dialed, bridge) {
  console.log('Adding channel %s and dialed channel %s to bridge %s',
    channel.name, dialed.name, bridge.id);

  bridge.addChannel({channel: [channel.id, dialed.id]}, function(err) {
    if (err) {
      throw err;
    }
  });
}

client.on('StasisStart', stasisStart);

client.start('bridge-dial');
}

```

### bridge-dial.js in action

The following shows the output of the `bridge-dial.js` script when a PJSIP channel for `alice` enters the application and dials a PJSIP channel for `bob`:

```

Channel PJSIP/alice-00000001 has entered our application
Created bridge 30430e82-83ed-4242-9f37-1bc040f70724
Adding channel PJSIP/alice-00000001 and dialed channel PJSIP/bob-00000002 to bridge
30430e82-83ed-4242-9f37-1bc040f70724
Dialed channel PJSIP/bob-00000002 has left our application, destroying bridge
30430e82-83ed-4242-9f37-1bc040f70724
Dialed channel PJSIP/bob-00000002 has been hung up, hanging up channel
PJSIP/alice-00000001
Channel PJSIP/alice-00000001 left our application, hanging up dialed channel undefined

```



## ARI and Bridges: Bridge Operations

### Moving Between Bridges

Channels can be both added and removed from bridges via the `POST - /bridges/{bridgeId}/addChannel` and `POST - /bridges/{bridgeId}/removeChannel` operations. This allows channels to be put in a holding bridge while waiting for an application to continue to its next step for example. One example of this would be to put an incoming channel into a holding bridge playing music on hold while dialing another endpoint. Once that endpoint answers, the incoming channel can be moved from the holding bridge to a mixing bridge to establish an audio call between the two channels.

### Example: Dialing with Entertainment

This example ARI application will do the following:

1. When a channel enters into the Stasis application, it will be put in a holding bridge and a call will be originated to the endpoint specified by the first command line argument to the script.
2. When that channel enters into the Stasis application, the original channel will be removed from the holding bridge, a mixing bridge will be created, and the two channels will be put in it.
3. If either channel hangs up, the other channel will also be hung up.
4. Once the dialed channel exists the Stasis application, the mixing bridge will be destroyed.

### Dialplan

For this example, we need to just drop the channel into Stasis, specifying our application:

```
extensions.conf
exten => 1000,1,NoOp()
same =>      n,Stasis(bridge-move,inbound,PJSIP/bob)
same =>      n,Hangup()
```

### Python

A large part of the implementation of this particular example is similar to the `bridge-dial.py` example. However, instead of ringing the inbound channel, we'll instead create a holding bridge and place the channel in said holding bridge. Since a holding bridge can hold a number of channels, we'll reuse the same holding bridge for all of the channels that use the application. The method to obtain the holding bridge is `find_or_create_holding_bridge`, shown below:

### On This Page

- Moving Between Bridges
- Example: Dialing with Entertainment
  - Dialplan
  - Python
    - b
    - b
  - JavaScript (Node.js)
    - b
    - b

```

# Our one and only holding bridge
holding_bridge = None

def find_or_create_holding_bridge():
    """Find our infinite wait bridge, or create a new one

    Returns:
    The one and only holding bridge
    """
    global holding_bridge

    if holding_bridge:
        return holding_bridge

    bridges = [candidate for candidate in client.bridges.list() if
                candidate.json['bridge_type'] == 'holding']
    if bridges:
        bridge = bridges[0]
        print "Using bridge {}".format(bridge.id)
    else:
        bridge = client.bridges.create(type='holding')
        bridge.startMoh()
        print "Created bridge {}".format(bridge.id)

    holding_bridge = bridge
    return holding_bridge

```

When the inbound channel enters the application, we'll place it into our waiting bridge:

```

wait_bridge = find_or_create_holding_bridge()
wait_bridge.addChannel(channel=channel.id)

```

When the dialed channel answers, we can remove the inbound channel from the waiting bridge - since there is only one waiting bridge being used, we can use `find_or_create_holding_bridge` to obtain it. We then place it into a newly created mixing bridge along with the dialed channel, in the same fashion as the `bridge-dial.py` example.

```

print "{} answered; bridging with {}".format(outgoing.json.get('name'),
                                              channel.json.get('name'))

    wait_bridge = find_or_create_holding_bridge()
    wait_bridge.removeChannel(channel=channel.id)

    bridge = client.bridges.create(type='mixing')
    bridge.addChannel(channel=[channel.id, outgoing.id])

```

### bridge-move.py

The full source code for `bridge-move.py` is shown below:

```

#!/usr/bin/env python

import logging
import requests
import ari

logging.basicConfig(level=logging.ERROR)

```

```

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

# Our one and only holding bridge
holding_bridge = None

def find_or_create_holding_bridge():
    """Find our infinite wait bridge, or create a new one

    Returns:
    The one and only holding bridge
    """
    global holding_bridge

    if holding_bridge:
        return holding_bridge

    bridges = [candidate for candidate in client.bridges.list() if
                candidate.json['bridge_type'] == 'holding']
    if bridges:
        bridge = bridges[0]
        print "Using bridge {}".format(bridge.id)
    else:
        bridge = client.bridges.create(type='holding')
        bridge.startMoh()
        print "Created bridge {}".format(bridge.id)

    holding_bridge = bridge
    return holding_bridge

def safe_hangup(channel):
    """Safely hang up the specified channel"""
    try:
        channel.hangup()
        print "Hung up {}".format(channel.json.get('name'))
    except requests.HTTPError as e:
        if e.response.status_code != requests.codes.not_found:
            raise e

def safe_bridge_destroy(bridge):
    """Safely destroy the specified bridge"""
    try:
        bridge.destroy()
    except requests.HTTPError as e:
        if e.response.status_code != requests.codes.not_found:
            raise e

def stasis_start_cb(channel_obj, ev):
    """Handler for StasisStart"""

    channel = channel_obj.get('channel')
    channel_name = channel.json.get('name')
    args = ev.get('args')

    if not args:
        print "Error: {} didn't provide any arguments!".format(channel_name)

```

```

    return

if args and args[0] != 'inbound':
    # Only handle inbound channels here
    return

if len(args) != 2:
    print "Error: {} didn't tell us who to dial".format(channel_name)
    channel.hangup()
    return

wait_bridge = find_or_create_holding_bridge()
wait_bridge.addChannel(channel=channel.id)

try:
    outgoing = client.channels.originate(endpoint=args[1],
                                         app='bridge-move',
                                         appArgs='dialed')

except requests.HTTPError:
    print "Whoops, pretty sure %s wasn't valid" % args[1]
    channel.hangup()
    return

channel.on_event('StasisEnd', lambda *args: safe_hangup(outgoing))
outgoing.on_event('StasisEnd', lambda *args: safe_hangup(channel))

def outgoing_start_cb(channel_obj, ev):
    """StasisStart handler for our dialed channel"""

    print "{} answered; bridging with {}".format(outgoing.json.get('name'),
                                                  channel.json.get('name'))

    wait_bridge = find_or_create_holding_bridge()
    wait_bridge.removeChannel(channel=channel.id)

    bridge = client.bridges.create(type='mixing')
    bridge.addChannel(channel=[channel.id, outgoing.id])

    # Clean up the bridge when done
    channel.on_event('StasisEnd', lambda *args:
                    safe_bridge_destroy(bridge))
    outgoing.on_event('StasisEnd', lambda *args:
                    safe_bridge_destroy(bridge))

    outgoing.on_event('StasisStart', outgoing_start_cb)

client.on_channel_event('StasisStart', stasis_start_cb)

```

```
client.run(apps='bridge-move')
```

### bridge-move.py in action

The following shows the output of the `bridge-move.py` script when a PJSIP channel for `alice` enters the application and dials a PJSIP channel for `bob`:

```
PJSIP/Alice-00000001 entered our application
Dialing PJSIP/Bob
PJSIP/Bob-00000002 answered: bridging with PJSIP/Alice-00000001
Hung up PJSIP/Bob-00000002
```

### JavaScript (Node.js)

This example is very similar to `bridge-dial.js` with one main difference: the original Stasis channel is put in a holding bridge while the an originate operation is used to dial another channel. Once the dialed channel enters into the Stasis application, the original channel will be removed from the holding bridge, and both channels will finally be put into a mixing bridge. Once a channel enters into our Stasis application, we either find an existing holding bridge or create one:

```
function findOrCreateHoldingBridge(channel) {
  client.bridges.list(function(err, bridges) {
    var holdingBridge = bridges.filter(function(candidate) {
      return candidate.bridge_type === 'holding';
    })[0];

    if (holdingBridge) {
      console.log('Using existing holding bridge %s', holdingBridge.id);

      originate(channel, holdingBridge);
    } else {
      client.bridges.create({type: 'holding'}, function(err, holdingBridge) {
        if (err) {
          throw err;
        }

        console.log('Created new holding bridge %s', holdingBridge.id);

        originate(channel, holdingBridge);
      });
    }
  });
}
```

We then add the channel to the holding bridge and start music on hold before continuing with dialing we we did in the `bridge-dial.js` example:

```
holdingBridge.addChannel({channel: channel.id}, function(err) {
  if (err) {
    throw err;
  }
  holdingBridge.startMoh(function(err) {
    // ignore error
  });
});
```

Once the endpoint has answered and a mixing bridge has been created, we proceed by first removing the original channel from the holding bridge and then adding both channels to the mixing bridge as before:

```

function moveToMixingBridge(channel, dialed, mixingBridge, holdingBridge) {
  console.log('Adding channel %s and dialed channel %s to bridge %s',
    channel.name, dialed.name, mixingBridge.id);

  holdingBridge.removeChannel({channel: channel.id}, function(err) {
    if (err) {
      throw err;
    }

    mixingBridge.addChannel(
      {channel: [channel.id, dialed.id]}, function(err) {
        if (err) {
          throw err;
        }
      });
  });
}

```

Note that we need to keep track of one more variable as we go down the application flow to ensure we have a reference to both the holding and mixing bridge. Again we use anonymous functions to pass extra arguments to callback handlers to keep the nested callbacks to a minimum.

### bridge-move.js

The full source code for `bridge-move.js` is shown below:

```

bridge-move.js

/*jshint node:true*/
'use strict';

var ari = require('ari-client');
var util = require('util');

// ensure endpoint was passed in to script
if (!process.argv[2]) {
  console.error('usage: node bridge-move.js endpoint');
  process.exit(1);
}

ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

// handler for client being loaded
function clientLoaded (err, client) {
  if (err) {
    throw err;
  }

  // handler for StasisStart event
  function stasisStart(event, channel) {
    // ensure the channel is not a dialed channel
    var dialed = event.args[0] === 'dialed';

    if (!dialed) {
      channel.answer(function(err) {
        if (err) {
          throw err;
        }

        console.log('Channel %s has entered our application', channel.name);
      });
    }
  }
}

```

```

        findOrCreateHoldingBridge(channel);
    });
}
}

function findOrCreateHoldingBridge(channel) {
    client.bridges.list(function(err, bridges) {
        var holdingBridge = bridges.filter(function(candidate) {
            return candidate.bridge_type === 'holding';
        })[0];

        if (holdingBridge) {
            console.log('Using existing holding bridge %s', holdingBridge.id);

            originate(channel, holdingBridge);
        } else {
            client.bridges.create({type: 'holding'}, function(err, holdingBridge) {
                if (err) {
                    throw err;
                }

                console.log('Created new holding bridge %s', holdingBridge.id);

                originate(channel, holdingBridge);
            });
        }
    });
}

function originate(channel, holdingBridge) {
    holdingBridge.addChannel({channel: channel.id}, function(err) {
        if (err) {
            throw err;
        }

        holdingBridge.startMoh(function(err) {
            // ignore error
        });
    });

    var dialed = client.Channel();

    channel.on('StasisEnd', function(event, channel) {
        safeHangup(dialed);
    });

    dialed.on('ChannelDestroyed', function(event, dialed) {
        safeHangup(channel);
    });

    dialed.on('StasisStart', function(event, dialed) {
        joinMixingBridge(channel, dialed, holdingBridge);
    });

    dialed.originate(
        {endpoint: process.argv[2], app: 'bridge-move', appArgs: 'dialed'},
        function(err, dialed) {
            if (err) {

```

```

        throw err;
    }
    });
}

// safely hangs the given channel
function safeHangup(channel) {
    console.log('Hanging up channel %s', channel.name);

    channel.hangup(function(err) {
        // ignore error
    });
}

// handler for dialed channel entering Stasis
function joinMixingBridge(channel, dialed, holdingBridge) {
    var mixingBridge = client.Bridge();

    dialed.on('StasisEnd', function(event, dialed) {
        dialedExit(dialed, mixingBridge);
    });

    dialed.answer(function(err) {
        if (err) {
            throw err;
        }
    });

    mixingBridge.create({type: 'mixing'}, function(err, mixingBridge) {
        if (err) {
            throw err;
        }

        console.log('Created mixing bridge %s', mixingBridge.id);

        moveToMixingBridge(channel, dialed, mixingBridge, holdingBridge);
    });
}

// handler for the dialed channel leaving Stasis
function dialedExit(dialed, mixingBridge) {
    console.log(
        'Dialed channel %s has left our application, destroying mixing bridge %s',
        dialed.name, mixingBridge.id);

    mixingBridge.destroy(function(err) {
        if (err) {
            throw err;
        }
    });
}

// handler for new mixing bridge ready for channels to be added to it
function moveToMixingBridge(channel, dialed, mixingBridge, holdingBridge) {
    console.log('Adding channel %s and dialed channel %s to bridge %s',
        channel.name, dialed.name, mixingBridge.id);

    holdingBridge.removeChannel({channel: channel.id}, function(err) {
        if (err) {

```



```
        throw err;
    }

    mixingBridge.addChannel(
        {channel: [channel.id, dialed.id]}, function(err) {
            if (err) {
                throw err;
            }
        });
    });
}

client.on('StasisStart', stasisStart);
```

```
client.start('bridge-move');  
}
```

### bridge-move.js in action

The following shows the output of the `bridge-move.js` script when a PJSIP channel for `alice` enters the application and dials a PJSIP channel for `bob`:

```
Channel PJSIP/alice-00000001 has entered our application  
Created new holding bridge e58641af-2006-4c3d-bf9e-8817baa27381  
Created mixing bridge 5ae49fee-e353-4ad9-bfa7-f8306d9dfd1e  
Adding channel PJSIP/alice-00000001 and dialed channel PJSIP/bob-00000002 to bridge  
5ae49fee-e353-4ad9-bfa7-f8306d9dfd1e  
Dialed channel PJSIP/bob-00000002 has left our application, destroying mixing bridge  
5ae49fee-e353-4ad9-bfa7-f8306d9dfd1e  
Hanging up channel PJSIP/alice-00000001  
Hanging up channel undefined
```

# Introduction to ARI and Media Manipulation

## Media Control

ARI contains tools for manipulating media, such as playing sound files, playing tones, playing numbers and digits, recording media, deleting stored recordings, manipulating playbacks (e.g. rewind and fast-forward), and intercepting DTMF tones. Some channel-specific information and examples for [playing media](#) and [intercepting DTMF](#) have been covered on previous pages. While some information will be repeated here, the intention of this section is to delve deeper into media manipulation.

To frame the discussion, we will be creating a set of applications that mimic a minimal voice mail system.

## About the Code Samples

The following ARI client libraries are used in the code samples on these pages

- Python code samples use [ari-py](#)
- Node.js samples use [node-ari-client](#)

On This Page
<ul style="list-style-type: none"><li>• <a href="#">Media Control</a></li><li>• <a href="#">About the Code Samples</a></li><li>• <a href="#">State Machine</a></li></ul>

Media In Depth
<ul style="list-style-type: none"><li>• <a href="#">ARI and Media: Part 1 - Recording</a></li><li>• <a href="#">ARI and Media: Part 2 - Playbacks</a></li></ul>

All of the code presented here has been tested with Asterisk 13 and works as intended. That being said, the code samples given are intended more to demonstrate the capabilities of ARI than to be a best practices guide for writing an application or to illustrate watertight code. Error-handling is virtually non-existent in the code samples. For a real application, Python calls to ARI should likely be in `try-catch` blocks in case there is an error, and Node.js calls to ARI should provide callbacks that detect if there was an error.

The asynchronous nature of Node.js and the `node-ari-client` library is not always used in the safest ways in the code samples provided. For instance, there are code samples where DTMF presses cause media operations to take place, and the code does not await confirmation that the media operation has actually completed before accepting more DTMF presses. This could potentially result in the desired media operations happening out of order if many DTMF presses occur in rapid succession.

## State Machine

Voice mail, at its heart, is an IVR. IVRs are most easily represented using a finite state machine. The way a state machine works is that a program switches between pre-defined states based on events that occur. Certain events will cause program code within the state to take certain actions, and other events will result in a change to a different program state. Each state can almost be seen as a self-contained program.

To start our state machine, we will define what events might cause state transitions. If you think about a typical IVR, the events that can occur are DTMF key presses, and changes in the state of a call, such as hanging up. As such, we'll define a base set of events.

## event.py

[Expand source](#)

```
class Event(object):
    # DTMF digits
    DTMF_1 = "1"
    DTMF_2 = "2"
    DTMF_3 = "3"
    DTMF_4 = "4"
    DTMF_5 = "5"
    DTMF_6 = "6"
    DTMF_7 = "7"
    DTMF_8 = "8"
    DTMF_9 = "9"
    DTMF_0 = "0"
    # Use "octothorpe" so there is no confusion about "pound" or "hash"
    # terminology.
    DTMF_OCTOTHORPE = "#"
    DTMF_STAR = "*"
    # Call has hung up
    HANGUP = "hangup"
    # Playback of a file has completed
    PLAYBACK_COMPLETE = "playback_complete"
    # Mailbox has been emptied
    MAILBOX_EMPTY = "empty"
```

## event.js

[Expand source](#)

```
var Event = {
    // DTMF digits
    DTMF_1: "1",
    DTMF_2: "2",
    DTMF_3: "3",
    DTMF_4: "4",
    DTMF_5: "5",
    DTMF_6: "6",
    DTMF_7: "7",
    DTMF_8: "8",
    DTMF_9: "9",
    DTMF_0: "0",
    // Use "octothorpe" so there is no confusion about "pound" or "hash"
    // terminology.
    DTMF_OCTOTHORPE: "#",
    DTMF_STAR: "*",
    // Call has hung up
    HANGUP: "hangup",
    // Playback of a file has completed
    PLAYBACK_COMPLETE: "playback_complete",
    // Mailbox has been emptied
    MAILBOX_EMPTY: "empty"
}
module.exports = Event;
```

There is no hard requirement for our application that we define events as named constants, but doing so makes it easier for tools like pylint and jslint to find potential mistakes.

After we have defined our events, we need to create a state machine itself. The state machine keeps track of what the current state is, and which events cause state changes. Here is a simple implementation of a state machine

## state\_machine.py

[Expand source](#)

```
class StateMachine(object):
    def __init__(self):
        self.transitions = {}
        self.current_state = None

    def add_transition(self, src_state, event, dst_state):
        if not self.transitions.get(src_state.state_name):
            self.transitions[src_state.state_name] = {}

        self.transitions[src_state.state_name][event] = dst_state

    def change_state(self, event):
        self.current_state = self.transitions[self.current_state.state_name][event]
        self.current_state.enter()

    def start(self, initial_state):
        self.current_state = initial_state
        self.current_state.enter()
```

## state\_machine.js

[Expand source](#)

```
function StateMachine() {
    var transitions = {};
    var current_state = null;

    this.add_transition = function(src_state, event, dst_state) {
        if (!transitions.hasOwnProperty(src_state.state_name)) {
            transitions[src_state.state_name] = {};
        }
        transitions[src_state.state_name][event] = dst_state;
    }

    this.change_state = function(event) {
        current_state = transitions[current_state.state_name][event];
        current_state.enter();
    }

    this.start = function(initial_state) {
        current_state = initial_state;
        current_state.enter();
    }
}

module.exports = StateMachine;
```

The state machine code is pretty straightforward. The state machine has transitions added to it with the `add_transition()` method and can be started with the `start()` method. Our use of the state machine will always be to define all transitions, and then to start the state machine.

States within a state machine have certain duties that they must fulfill if they want to work well in the state machine we have devised

- A state must know what events should cause it to change states, though it does not need to know what state it will be transitioning to.
- A state must set up ARI event listeners each time the state is entered, and it must remove these before changing states.
- The state must define the following attributes:
  - `state_name`, a string that represents the name of the state.
  - `enter()`, a method that is called whenever the state is entered.

It should be noted that this state machine implementation is not necessarily ideal, since it requires the states to know what events cause it to change states. However, it will become clear later that for a simple voice mail system, this is not that big a deal. To see how we use this state machine, continue on

to the sub-pages.

## ARI and Media: Part 1 - Recording

### The Recording API

Recordings in ARI are divided into two main categories: live and stored. Live recordings are those that are currently being recorded on a channel or bridge, and stored recordings are recordings that have been completed and saved to the file system. The API for the `/recordings` resource can be found [here](#).

Live recordings can be manipulated as they are being made, with options to manipulate the flow of audio such as muting, pausing, stopping, or canceling the recording. Stored recordings are simply files on the file system on which Asterisk is installed. The location of stored recordings is in the `/recording` subdirectory of the configured `astspooldir` in `asterisk.conf`. By default, this places recordings in `/var/spool/asterisk/recording`.

Channels can have their audio recorded using the `/channels/{channelId}/record` resource, and Bridges can have their audio recorded using the `/bridges/{bridgeId}/record` resource.

### Voice Mail Application Skeleton

Our application to record voice mails will be based on the following skeleton. As we add new features, we will create new states for our state machine, and add a few extra lines to our application skeleton in order to link the states together appropriately.

On this Page
<ul style="list-style-type: none"><li>• <a href="#">The Recording API</a></li><li>• <a href="#">Voice Mail Application Skeleton</a></li><li>• <a href="#">Basic Voice Mail Recording</a></li><li>• <a href="#">Cancelling a Recording</a></li><li>• <a href="#">Operating on Stored Recordings</a></li><li>• <a href="#">Recording Bridges</a></li></ul>

**vm-record.py**[Expand source](#)

```
#!/usr/bin/env python

import ari
import logging
import time
import os
import sys

from event import Event
from state_machine import StateMachine
# As we add more states to our state machine, we'll import the necessary
# states here.

logging.basicConfig(level=logging.ERROR)
LOGGER = logging.getLogger(__name__)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

class VoiceMailCall(object):
    def __init__(self, ari_client, channel, mailbox):
        self.client = ari_client
        self.channel = channel
        self.vm_path = os.path.join('voicemail', mailbox, str(time.time()))
        self.setup_state_machine()

    def setup_state_machine(self):
        # This is where we will initialize states, create a state machine, add
        # state transitions to the state machine, and start the state machine.

def stasis_start_cb(channel_obj, event):
    channel = channel_obj['channel']
    channel_name = channel.json.get('name')
    mailbox = event.get('args')[0]

    print("Channel {0} recording voicemail for {1}".format(
        channel_name, mailbox))

    channel.answer()
    VoiceMailCall(client, channel, mailbox)

client.on_channel_event('StasisStart', stasis_start_cb)
client.run(apps=sys.argv[1])
```



**vm-record.js**[Expand source](#)

```
/*jshint node:true*/
'use strict';

var ari = require('ari-client');
var util = require('util');
var path = require('path');

var Event = require('./event');
var StateMachine = require('./state_machine');
// As we add new states to our state machine, this is where we will
// add the new required states.

ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

var VoiceMailCall = function(ari_client, channel, mailbox) {
  this.client = ari_client;
  this.channel = channel;
  this.vm_path = path.join('voicemail', mailbox, Date.now().toString());

  this.setup_state_machine = function() {
    // This is where we will initialize states, create a state machine, add
    // state transitions to the state machine, and start the state machine.
  }

  this.setup_state_machine();
}

function clientLoaded(err, client) {
  if (err) {
    throw err;
  }

  client.on('StasisStart', stasisStart);

  function stasisStart(event, channel) {
    var mailbox = event.args[0]
    channel.answer(function(err) {
      if (err) {
        throw err;
      }
      new VoiceMailCall(client, channel, mailbox);
    });
  }

  client.start(process.argv[2]);
}
```

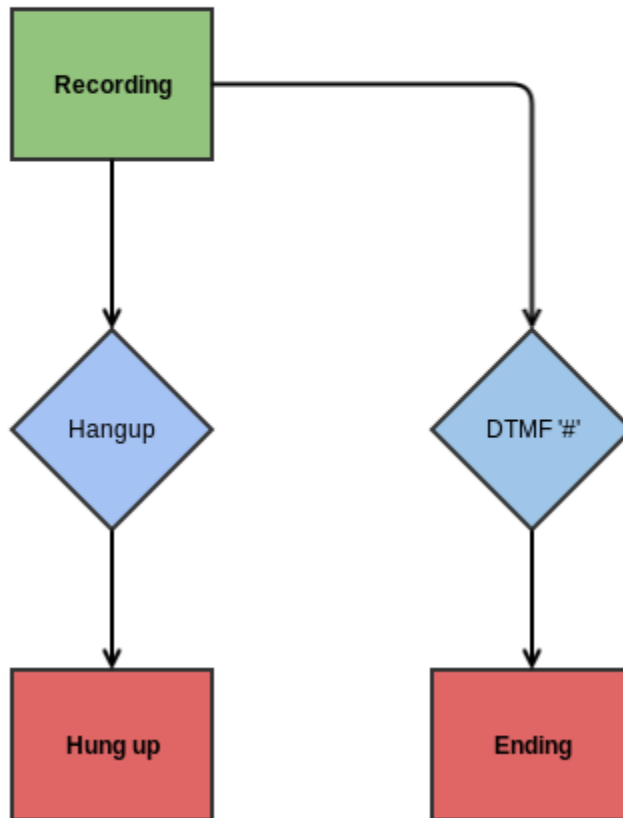
With a few modifications, this same application skeleton can be adapted for use with non-voice mail applications. The biggest voice mail-specific thing being done here is the calculation of the path for voice mail recordings based on the application argument. The intended use of this application is something like the following:

```
[default]
exten => _3XX,1,NoOp()
    same => n,Stasis(vm-record, ${EXTEN})
    same => n,Hangup()
```

This way, when calling any three-digit extension that begins with the number '3', the user will call into the application with the mailbox dialed (e.g. dialling "305" will allow the user to leave a message for mailbox "305").

### Basic Voice Mail Recording

We've seen a lot of the underlying concepts for our application, so let's actually make something useful now. We'll start with a very simple application that allows callers to record a message upon entering the application. When the caller has completed recording the message, the caller may press the '#' key or may hang up to accept the recording. Here is a state machine diagram for the application:



Notice that even though DTMF '#' and a caller hangup result in the same end result that there are two separate states that are transitioned into. This is because our code needs to behave differently at the end of a voice mail call based on whether the channel has been hung up or not. In short, the "Ending" state in the diagram will forcibly hang up the channel, whereas the "Hung up" state does not need to do so.

For this, we will be defining three states: recording, hungup, and ending. The following is the code for the three states:

```
from event import Event

class RecordingState(object):
    state_name = "recording"

    def __init__(self, call):
        self.call = call
        self.hangup_event = None
        self.dtmf_event = None
        self.recording = None

    def enter(self):
        print "Entering recording state"
        self.hangup_event = self.call.channel.on_event('ChannelHangupRequest',
            self.on_hangup)
        self.dtmf_event = self.call.channel.on_event('ChannelDtmfReceived',
            self.on_dtmf)
        self.recording = self.call.channel.record(name=self.call.vm_path,
            format='wav',
            beep=True,
            ifExists='overwrite')
        print "Recording voicemail at {0}".format(self.call.vm_path)

    def cleanup(self):
        print "Cleaning up event handlers"
        self.dtmf_event.close()
        self.hangup_event.close()

    def on_hangup(self, channel, event):
        print "Accepted recording {0} on hangup".format(self.call.vm_path)
        self.cleanup()
        self.call.state_machine.change_state(Event.HANGUP)

    def on_dtmf(self, channel, event):
        digit = event.get('digit')
        if digit == '#':
            rec_name = self.recording.json.get('name')
            print "Accepted recording {0} on DTMF #".format(rec_name)
            self.cleanup()
            self.recording.stop()
            self.call.state_machine.change_state(Event.DTMF_OCTOTHORPE)
```

## recording\_state.js

[Expand source](#)

```
var Event = require('./event')

function RecordingState(call) {
  this.state_name = "recording";

  this.enter = function() {
    var recording = call.client.LiveRecording(call.client, {name: call.vm_path});
    console.log("Entering recording state");
    call.channel.on("ChannelHangupRequest", on_hangup);
    call.channel.on("ChannelDtmfReceived", on_dtmf);
    call.channel.record({name: recording.name, format: 'wav', beep: true, ifExists:
'overwrite'}, recording);

    function cleanup() {
      call.channel.removeListener('ChannelHangupRequest', on_hangup);
      call.channel.removeListener('ChannelDtmfReceived', on_dtmf);
    }

    function on_hangup(event, channel) {
      console.log("Accepted recording %s on hangup", recording.name);
      cleanup();
      call.state_machine.change_state(Event.HANGUP);
    }

    function on_dtmf(event, channel) {
      switch (event.digit) {
        case '#':
          console.log("Accepted recording", call.vm_path);
          cleanup();
          recording.stop(function(err) {
            call.state_machine.change_state(Event.DTMF_OCTOTHORPE);
          });
          break;
      }
    }
  }
}

module.exports = RecordingState;
```

When entered, the state sets up listeners for hangup and DTMF events on the channel, since those are the events that will cause the state to change. In all cases, before a state change occurs, the `cleanup()` function is invoked to remove event listeners. This way, the event listeners set by the recording state will not accidentally still be set up when the next state is entered. This same `cleanup()` method will be used for all states we create that set up ARI event listeners.

The `stop` method causes a live recording to finish and be saved to the file system. Notice that the `on_hangup()` method does not attempt to stop the live recording. This is because when a channel hangs up, any live recordings on that channel are automatically stopped and stored.

The other two states in the state machine are much simpler, since they are terminal states and do not need to watch for any events.

### ending\_state.py

[Expand source](#)

```
class EndingState(object):
    state_name = "ending"

    def __init__(self, call):
        self.call = call

    def enter(self):
        channel_name = self.call.channel.json.get('name')
        print "Ending voice mail call from {0}".format(channel_name)
        self.call.channel.hangup()
```

### ending\_state.js

[Expand source](#)

```
function EndingState(call) {
    this.state_name = "ending";

    this.enter = function() {
        channel_name = call.channel.name;
        console.log("Ending voice mail call from", channel_name);
        call.channel.hangup();
    }
}

module.exports = EndingState;
```

### hangup\_state.py

[Expand source](#)

```
class HungUpState(object):
    state_name = "hungup"

    def __init__(self, call):
        self.call = call

    def enter(self):
        channel_name = self.call.channel.json.get('name')
        print "Channel {0} hung up".format(channel_name)
```

### hangup\_state.js

[Expand source](#)

```
function HungUpState(call) {
    this.state_name = "hungup";

    this.enter = function() {
        channel_name = call.channel.name;
        console.log("Channel %s hung up", channel_name);
    }
}

module.exports = HungUpState;
```

These two states are two sides to the same coin. The `EndingState` is used to end the call by hanging up the channel, and the `HungUpState` is used to terminate the state machine when the caller has hung up. You may find yourself wondering why a `HungUpState` is needed at all. For our application, it does not do much, but it's a great place to perform post-call logic if your application demands it. See the second reader exercise on this page to see an example of that.

Using the application skeleton we set up earlier, we can make the following modifications to accommodate our state machine:

### vm-call.py

[Expand source](#)

```
# At the top of the file
from recording_state import RecordingState
from ending_state import EndingState
from hungup_state import HungUpState

# Inside our VoiceMailCall class
def setup_state_machine(self):
    hungup_state = HungUpState(self)
    recording_state = RecordingState(self)
    ending_state = EndingState(self)

    self.state_machine = StateMachine()
    self.state_machine.add_transition(recording_state, Event.DTMF_OCTOTHORPE,
                                     ending_state)
    self.state_machine.add_transition(recording_state, Event.HANGUP,
                                     hungup_state)
    self.state_machine.start(recording_state)
```

### vm-call.js

[Expand source](#)

```
// At the top of the file
var RecordingState = require('./recording_state');
var EndingState = require('./ending_state');
var HungUpState = require('./hungup_state');

// Inside our VoiceMailCall function
this.setup_state_machine = function() {
    var hungup_state = new HungUpState(self)
    var recording_state = new RecordingState(self)
    var ending_state = new EndingState(self)

    this.state_machine = new StateMachine()
    this.state_machine.add_transition(recording_state, Event.DTMF_OCTOTHORPE,
    ending_state)
    this.state_machine.add_transition(recording_state, Event.HANGUP, hungup_state)
    this.state_machine.start(recording_state)
}
```

The following is a sample output of a user calling the application and pressing the '#' key when finished recording

```
Channel PJSIP/200-00000003 recording voicemail for 305
Entering recording state
Recording voicemail at voicemail/305/1411497846.53
Accepted recording voicemail/305/1411497846.53
Cleaning up event handlers
Ending voice mail call from PJSIP/200-00000003
```

## Reader Exercise 1

Currently, the voicemails being recorded are all kept in a single "folder" for a specific mailbox. See if you can change the code to record messages in an "INBOX" folder on the mailbox instead.

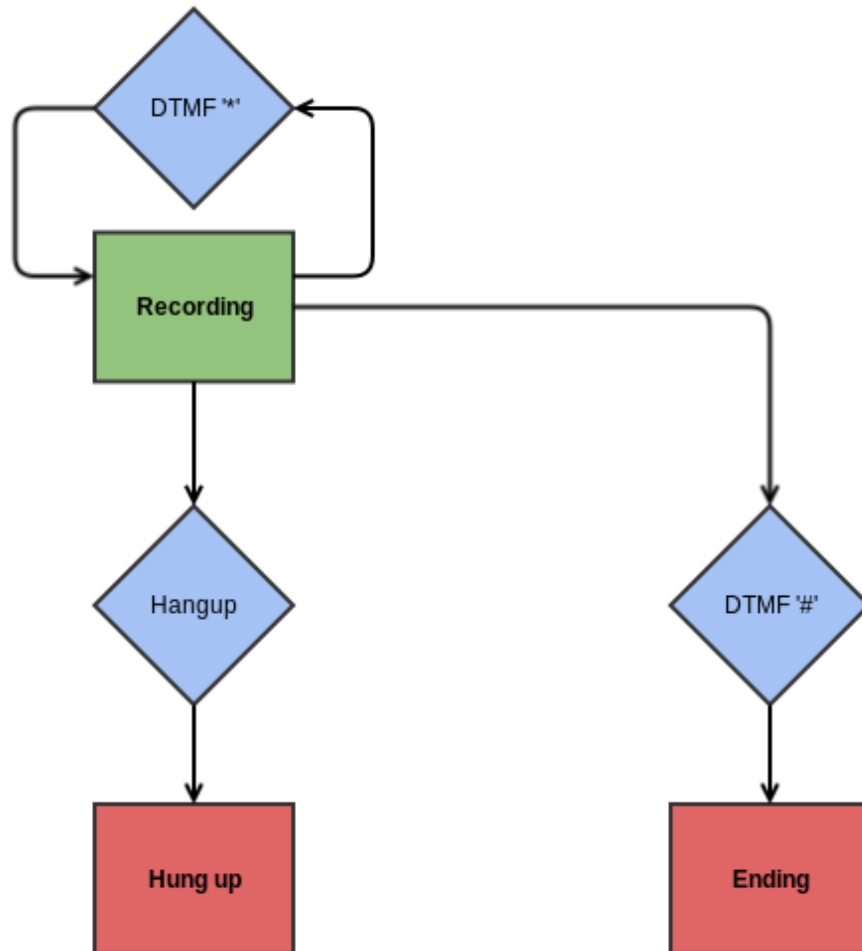
## Reader Exercise 2

`EndingState` and `HungUpState` don't do much of anything at the moment. States like these can be great in voice mail applications for updating the message-waiting state of mailboxes on a system. If you're feeling industrious, read the API for the `/mailboxes` resource in ARI. Try to change `HungUpState` and `EndingState` to update the message-waiting status of a mailbox when a new message is left. To keep the exercise simple, for now you can assume the following:

- The number of "old" messages in a mailbox is always 0
- Since you are more concerned with alerting the mailbox owner that there exist new messages, do not try to count the messages in the mailbox. Instead, just state that there is 1 new message.

### Cancelling a Recording

Now we have a simple application set up to record a message, but it's pretty bare at the moment. Let's start expanding some. One feature we can add is the ability to press a DTMF key while recording a voice mail to cancel the current recording and re-start the recording process. We'll use the DTMF '\*' key to accomplish this. The updated state machine diagram looks like the following:



All that has changed is that there is a new transition, which means a minimal change to our current code to facilitate the change. In our `recording_state` file, we will rewrite the `on_dtmf` method as follows:

### recording\_state.py

[Expand source](#)

```
def on_dtmf(self, channel, event):
    digit = event.get('digit')
    if digit == '#':
        rec_name = self.recording.json.get('name')
        print "Accepted recording {0} on DTMF #".format(rec_name)
        self.cleanup()
        self.recording.stop()
        self.call.state_machine.change_state(Event.DTMF_OCTOTHORPE)
    # NEW CONTENT
    elif digit == '*':
        rec_name = self.recording.json.get('name')
        print "Canceling recording {0} on DTMF *".format(rec_name)
        self.cleanup()
        self.recording.cancel()
        self.call.state_machine.change_state(Event.DTMF_STAR)
```

### recording\_state.js

[Expand source](#)

```
function on_dtmf(event, channel) {
    switch (event.digit) {
    case '#':
        console.log("Accepted recording", call.vm_path);
        cleanup();
        recording.stop(function(err) {
            call.state_machine.change_state(Event.DTMF_OCTOTHORPE);
        });
        break;
    // NEW CONTENT
    case '*':
        console.log("Canceling recording", call.vm_path);
        cleanup();
        recording.cancel(function(err) {
            call.state_machine.change_state(Event.DTMF_STAR);
        });
        break;
    }
}
```

The first part of the method is the same as it was before, but we have added extra handling for when the user presses the '\*' key. The `cancel()` method for live recordings causes the live recording to be stopped and for it not to be stored on the file system.

We also need to add our new transition while setting up our state machine. Our `VoiceMailCall::setup_state_machine()` method now looks like:



## vm-call.py

[Expand source](#)

```
def setup_state_machine(self):
    hungup_state = HungUpState(self)
    recording_state = RecordingState(self)
    ending_state = EndingState(self)
    self.state_machine = StateMachine()
    self.state_machine.add_transition(recording_state, Event.DTMF_OCTOTHORPE,
                                     ending_state)

    self.state_machine.add_transition(recording_state, Event.HANGUP,
                                     hungup_state)

    self.state_machine.add_transition(recording_state, Event.DTMF_STAR,
                                     recording_state)

    self.state_machine.start(recording_state)
```

## vm-call.js

[Expand source](#)

```
this.setup_state_machine = function() {
    var hungup_state = new HungUpState(this);
    var recording_state = new RecordingState(this);
    var ending_state = new EndingState(this);

    this.state_machine = new StateMachine();
    this.state_machine.add_transition(recording_state, Event.DTMF_OCTOTHORPE,
    ending_state);
    this.state_machine.add_transition(recording_state, Event.HANGUP, hungup_state);
    this.state_machine.add_transition(recording_state, Event.DTMF_STAR, recording_state);
    this.state_machine.start(recording_state);
}
```

This is exactly the same as it was, except for the penultimate line adding the `Event.DTMF_STAR` transition. Here is sample output for when a user calls in, presses '\*' twice, and then presses '#' to complete the call

```
Channel PJSIP/200-00000007 recording voicemail for 305
Entering recording state
Recording voicemail at voicemail/305/1411498790.65
Canceling recording voicemail/305/1411498790.65 on DTMF *
Cleaning up event handlers
Entering recording state
Recording voicemail at voicemail/305/1411498790.65
Canceling recording voicemail/305/1411498790.65 on DTMF *
Cleaning up event handlers
Entering recording state
Recording voicemail at voicemail/305/1411498790.65
Accepted recording voicemail/305/1411498790.65 on DTMF #
Cleaning up event handlers
Ending voice mail call from PJSIP/200-00000007
```

### Reader Exercise 3

We have covered the `stop()` and `cancel()` methods, but live recordings provide other methods as well. In particular, there are `pause()`, which causes the live recording to temporarily stop recording audio, and `unpause()`, which causes the live recording to resume recording audio.

Modify the `RecordingState` to allow a DTMF digit of your choice to toggle pausing and unpausing the live recording.

### Reader Exercise 4

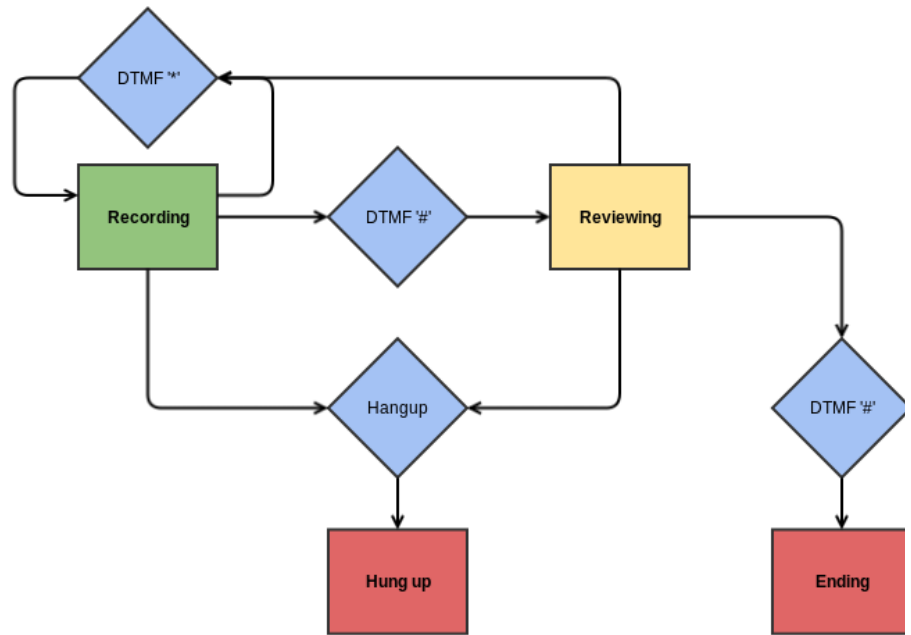
Our application provides the ability to cancel recordings and re-record them, but it gives no ability to cancel the recording and end the call.

Modify the `RecordingState` to allow for a DTMF digit of your choice to cancel the recording and end the call.

## Operating on Stored Recordings

So far, we've recorded a channel, stopped a live recording, and cancelled a live recording. Now let's turn our attention to operations that can be performed

on stored recordings. An obvious operation to start with is to play back the stored recording. We're going to make another modification to our voice mail recording application that adds a "reviewing" state after a voicemail is recorded. In this state, a user that has recorded a voice mail will hear the recorded message played back to him/her. The user may press the '#' key or hang up in order to accept the recorded message, or the user may press '\*' to erase the stored recording and record a new message in its place. Below is the updated state diagram with the new "reviewing" state added.



To realize this, here is the code for our new "reviewing" state:

```
import uuid

class ReviewingState(object):
    state_name = "reviewing"

    def __init__(self, call):
        self.call = call
        self.playback_id = None
        self.hangup_event = None
        self.playback_finished = None
        self.dtmf_event = None
        self.playback = None

    def enter(self):
        self.playback_id = str(uuid.uuid4())
        print "Entering reviewing state"
        self.hangup_event = self.call.channel.on_event("ChannelHangupRequest",
            self.on_hangup)
        self.playback_finished = self.call.client.on_event(
            'PlaybackFinished', self.on_playback_finished)
        self.dtmf_event = self.call.channel.on_event('ChannelDtmfReceived',
            self.on_dtmf)
        self.playback = self.call.channel.playWithId(
            playbackId=self.playback_id, media="recording:{0}".format(
                self.call.vm_path))

    def cleanup(self):
        self.playback_finished.close()
        if self.playback:
            self.playback.stop()
        self.dtmf_event.close()
        self.hangup_event.close()

    def on_hangup(self, channel, event):
        print "Accepted recording {0} on hangup".format(self.call.vm_path)
        self.cleanup()
        self.call.state_machine.change_state(Event.HANGUP)

    def on_playback_finished(self, event):
        if self.playback_id == event.get('playback').get('id'):
            self.playback = None

    def on_dtmf(self, channel, event):
        digit = event.get('digit')
        if digit == '#':
            print "Accepted recording {0} on DTMF #".format(self.call.vm_path)
            self.cleanup()
            self.call.state_machine.change_state(Event.DTMF_OCTOTHORPE)
        elif digit == '*':
            print "Discarding stored recording {0} on DTMF *".format(self.call.vm_path)
            self.cleanup()
            self.call.client.recordings.deleteStored(
                recordingName=self.call.vm_path)
            self.call.state_machine.change_state(Event.DTMF_STAR)
```

**reviewing\_state.js**[› Expand source](#)

```
var Event = require('./event');

function ReviewingState(call) {
  this.state_name = "reviewing";

  this.enter = function() {
    var playback = call.client.Playback();
    var url = "recording:" + call.vm_path;
    console.log("Entering reviewing state");
    call.channel.on("ChannelHangupRequest", on_hangup);
    call.channel.on("ChannelDtmfReceived", on_dtmf);
    call.client.on("PlaybackFinished", on_playback_finished);
    call.channel.play({media: url}, playback);

    function cleanup() {
      call.channel.removeListener('ChannelHangupRequest', on_hangup);
      call.channel.removeListener('ChannelDtmfReceived', on_dtmf);
      call.client.removeListener('PlaybackFinished', on_playback_finished);
      if (playback) {
        playback.stop();
      }
    }

    function on_hangup(event, channel) {
      console.log("Accepted recording %s on hangup", call.vm_path);
      playback = null;
      cleanup();
      call.state_machine.change_state(Event.HANGUP);
    }

    function on_playback_finished(event) {
      if (playback && (playback.id === event.playback.id)) {
        playback = null;
      }
    }

    function on_dtmf(event, channel) {
      switch (event.digit) {
        case '#':
          console.log("Accepted recording", call.vm_path);
          cleanup();
          call.state_machine.change_state(Event.DTMF_OCTOTHORPE);
          break;
        case '*':
          console.log("Canceling recording", call.vm_path);
          cleanup();
          call.client.recordings.deleteStored({recordingName: call.vm_path});
          call.state_machine.change_state(Event.DTMF_STAR);
          break;
      }
    }
  }
}

module.exports = ReviewingState;
```

The code for this state is similar to the code from `RecordingState`. The big difference is that instead of recording a message, it is playing back a stored recording. Stored recordings can be played using the channel's `play()` method (or as we have used in the python code, `playWithId()`). If the URI of the media to be played is prefixed with the "recording:" scheme, then Asterisk knows to search for the specified file where recordings are stored. More information on playing back files on channels, as well as a detailed list of media URI schemes can be found [here](#). Note the method that is called when a DTMF "\*" is received. The `deleteStored()` method can be used on the `/recordings` resource of the ARI client to delete a stored recording from the file system on which Asterisk is running.

One more thing to point out is the code that runs in `on_playback_finished()`. When reviewing a voicemail recording, the message may finish playing back before the user decides what to do with it. If this happens, we detect that the playback has finished so that we do not attempt to stop an already-finished playback once the user decides how to proceed.

We need to get this new state added into our state machine, so we make the following modifications to our code to allow for the new state to be added:

#### vm-call.py

› [Expand source](#)

```
#At the top of the file
from reviewing_state import ReviewingState

#In VoiceMailCall::setup_state_machine
def setup_state_machine(self):
    hungup_state = HungUpState(self)
    recording_state = RecordingState(self)
    ending_state = EndingState(self)
    reviewing_state = ReviewingState(self)
    self.state_machine = StateMachine()
    self.state_machine.add_transition(recording_state, Event.DTMF_OCTOTHORPE,
                                     reviewing_state)
    self.state_machine.add_transition(recording_state, Event.HANGUP,
                                     hungup_state)
    self.state_machine.add_transition(recording_state, Event.DTMF_STAR,
                                     recording_state)
    self.state_machine.add_transition(reviewing_state, Event.HANGUP,
                                     hungup_state)
    self.state_machine.add_transition(reviewing_state, Event.DTMF_OCTOTHORPE,
                                     ending_state)
    self.state_machine.add_transition(reviewing_state, Event.DTMF_STAR,
                                     recording_state)
    self.state_machine.start(recording_state)
```

## vm-call.js

› Expand source

```
// At the top of the file
var ReviewingState = require('./reviewing_state');

// In VoicemailCall::setup_state_machine
this.setup_state_machine = function() {
  var hungup_state = new HungUpState(this);
  var recording_state = new RecordingState(this);
  var ending_state = new EndingState(this);
  var reviewing_state = new ReviewingState(this);

  this.state_machine = new StateMachine();
  this.state_machine.add_transition(recording_state, Event.DTMF_OCTOTHORPE,
reviewing_state);
  this.state_machine.add_transition(recording_state, Event.HANGUP, hungup_state);
  this.state_machine.add_transition(recording_state, Event.DTMF_STAR, recording_state);
  this.state_machine.add_transition(reviewing_state, Event.DTMF_OCTOTHORPE,
ending_state);
  this.state_machine.add_transition(reviewing_state, Event.HANGUP, hungup_state);
  this.state_machine.add_transition(reviewing_state, Event.DTMF_STAR, recording_state);
  this.state_machine.start(recording_state);
}
```

The following is the output from a sample call. The user records audio, then presses '#'. Upon hearing the recording, the user decides to record again, so the user presses '\*'. After re-recording, the user presses '#'. The user hears the new version of the recording played back and is satisfied with it, so the user presses '#' to accept the recording.

```
Channel PJSIP/200-00000009 recording voicemail for 305
Entering recording state
Recording voicemail at voicemail/305/1411501058.42
Accepted recording voicemail/305/1411501058.42 on DTMF #
Cleaning up event handlers
Entering reviewing state
Discarding stored recording voicemail/305/1411501058.42 on DTMF *
Entering recording state
Recording voicemail at voicemail/305/1411501058.42
Accepted recording voicemail/305/1411501058.42 on DTMF #
Cleaning up event handlers
Entering reviewing state
Accepted recording voicemail/305/1411501058.42 on DTMF #
Ending voice mail call from PJSIP/200-00000009
```

## Reader Exercise 5

In the previous section we introduced the ability to delete a stored recording. Stored recordings have a second operation available to them: **copying**. The `copy()` method of a stored recording can be used to copy the stored recording from one location to another.

For this exercise modify `ReviewingState` to let a DTMF key of your choice copy the message to a different mailbox on the system. When a user presses this DTMF key, the state machine should transition into a new state called "copying." The "copying" state should gather DTMF from the user to determine which mailbox the message should be copied to. If '#' is entered, then the message is sent to the mailbox the user has typed in. If '\*' is entered, then the copying operation is cancelled. Both a '#' and a '\*' should cause the state machine to transition back into `ReviewingState`.

As an example, let's say that you have set DTMF '0' to be the key that the user presses in `ReviewingState` to copy the message. The user presses '0'. The user then presses '3' '2' '0' '#'. The message should be copied to mailbox "320", and the user should start hearing the message played back again. Now let's say the user presses '0' to copy the message again. The user then presses '3' '2' '1' '0' '\*'. The message should not be copied to any mailbox, and the user should start hearing the message played back again.

## Recording Bridges

This discussion of recordings has focused on recording channel audio. It's important to note that bridges also have an option to be recorded. What's the difference? Recording a channel's audio records only the audio coming **from** a channel. Recording a bridge records the mixed audio coming from all channels into the bridge. This means that if you are attempting to do something like record a conversation between participants in a phone call, you would want to record the audio in the bridge rather than on either of the channels involved.

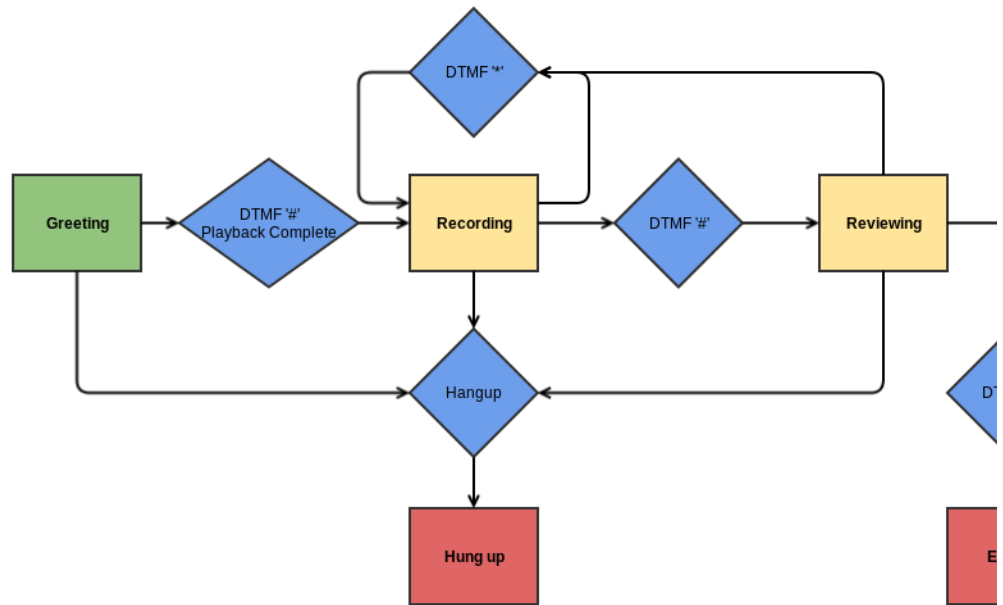
Once recording is started on a bridge, the operations available for the live recording and the resulting stored recording are exactly the same as for live recordings and stored recordings on a channel. Since the API for recording a bridge and recording a channel are so similar, this page will not provide any

examples of recording bridge audio.

## ARI and Media: Part 2 - Playbacks

### Querying for sounds

In our voice mail application we have been creating, we have learned the ins and outs of creating and manipulating live and stored recordings. Let's make the voice mail application more user-friendly now by adding some playbacks of installed sounds. The voice mail application has some nice capabilities, but it is not very user-friendly yet. Let's modify the current application to play a greeting to the user when they call into the application. This is the updated state machine:



#### On this Page

- [Querying for sounds](#)
- [Controlling playbacks](#)
- [Playbacks on bridges](#)

To make the new "greeting" state more interesting, we are going to add some safety to this state by ensuring that the sound we want to play is installed on the system. The `/sounds` resource in ARI provides methods to list the sounds installed on the system, as well as the ability to get specific sound files.

Asterisk searches for sounds in the `/sounds/` subdirectory of the configured `astdatadir` option in `asterisk.conf`. By default, Asterisk will search for sounds in `/var/lib/asterisk/sounds`. When Asterisk starts up, it indexes the installed sounds and keeps an in-data representation of those sound files. When an ARI application asks Asterisk for details about a specific sound or for a list of sounds on the system, Asterisk consults its in-memory index instead of searching the file system directly. This has some trade-offs. When querying for sound information, this in-memory indexing makes the operations much faster. On the other hand, it also means that Asterisk has to be "poked" to re-index the sounds if new sounds are added to the file system after Asterisk is running. The Asterisk CLI command "module reload sounds" provides a means of having Asterisk re-index the sounds on the system so that they are available to ARI.

For our greeting, we will play the built-in sound "vm-intro". Here is the code for our new state:



**greeting\_state.py**[› Expand source](#)

```
from event import Event

def sounds_installed(client):
    try:
        client.sounds.get(soundId='vm-intro')
    except:
        print "Required sound 'vm-intro' not installed. Aborting"
        raise

class GreetingState(object):
    state_name = "greeting"

    def __init__(self, call):
        self.call = call
        self.hangup_event = None
        self.playback_finished = None
        self.dtmf_event = None
        self.playback = None
        sounds_installed(call.client)

    def enter(self):
        print "Entering greeting state"
        self.hangup_event = self.call.channel.on_event('ChannelHangupRequest',
            self.on_hangup)
        self.playback_finished = self.call.client.on_event(
            'PlaybackFinished', self.on_playback_finished)
        self.dtmf_event = self.call.channel.on_event('ChannelDtmfReceived',
            self.on_dtmf)
        self.playback = self.call.channel.play(media="sound:vm-intro")

    def cleanup(self):
        self.playback_finished.close()
        self.dtmf_event.close()
        self.hangup_event.close()

    def on_hangup(self, channel, event):
        print "Abandoning voicemail recording on hangup"
        self.cleanup()
        self.call.state_machine.change_state(Event.HANGUP)

    def on_playback_finished(self, playback):
        self.cleanup()
        self.call.state_machine.change_state(Event.PLAYBACK_COMPLETE)

    def on_dtmf(self, channel, event):
        digit = event.get('digit')
        if digit == '#':
            print "Cutting off greeting on DTMF #"
            # Let on_playback_finished take care of state change
            self.playback.stop()
```

**greeting\_state.js**[› Expand source](#)

```

var Event = require('./event');

function sounds_installed(client) {
  client.sounds.get({soundId: 'vm-intro'}, function(err) {
    if (err) {
      console.log("Required sound 'vm-intro' not installed. Aborting");
      throw err;
    }
  });
}

function GreetingState(call) {
  this.state_name = "greeting";
  sounds_installed(call.client);

  this.enter = function() {
    var playback = call.client.Playback();
    console.log("Entering greeting state");
    call.channel.on("ChannelHangupRequest", on_hangup);
    call.channel.on("ChannelDtmfReceived", on_dtmf);
    call.client.on("PlaybackFinished", on_playback_finished);
    call.channel.play({media: 'sound:vm-intro'}, playback);

    function cleanup() {
      call.channel.removeListener('ChannelHangupRequest', on_hangup);
      call.channel.removeListener('ChannelDtmfReceived', on_dtmf);
      call.client.removeListener('PlaybackFinished', on_playback_finished);
      if (playback) {
        playback.stop();
      }
    }

    function on_hangup(event, channel) {
      console.log("Abandoning voicemail recording on hangup");
      cleanup();
      call.state_machine.change_state(Event.HANGUP);
    }

    function on_playback_finished(event) {
      if (playback && playback.id === event.playback.id) {
        cleanup();
        call.state_machine.change_state(Event.PLAYBACK_COMPLETE);
      }
    }

    function on_dtmf(event, channel) {
      switch (event.digit) {
        case '#':
          console.log("Skipping greeting");
          cleanup();
          call.state_machine.change_state(Event.DTMF_OCTOTHORPE);
        }
      }
    }
  }

  module.exports = GreetingState;
}

```

The `sounds.get()` method employed here allows for a single sound to be retrieved based on input parameters. Here, we simply specify the name of the recording we want to ensure that it exists in some form on the system. By checking for the sound's existence in the initialization of `GreetingState`, we

can abort the call early if the sound is not installed.

And here is our updated state machine:

### vm-call.py

[Expand source](#)

```
#At the top of the file
from greeting_state import GreetingState

#In VoiceMailCall::setup_state_machine()
def setup_state_machine(self):
    hungup_state = HungUpState(self)
    recording_state = RecordingState(self)
    ending_state = EndingState(self)
    reviewing_state = ReviewingState(self)
    greeting_state = GreetingState(self)
    self.state_machine = StateMachine()
    self.state_machine.add_transition(recording_state, Event.DTMF_OCTOTHORPE,
                                     reviewing_state)
    self.state_machine.add_transition(recording_state, Event.HANGUP,
                                     hungup_state)
    self.state_machine.add_transition(recording_state, Event.DTMF_STAR,
                                     recording_state)
    self.state_machine.add_transition(reviewing_state, Event.HANGUP,
                                     hungup_state)
    self.state_machine.add_transition(reviewing_state, Event.DTMF_OCTOTHORPE,
                                     ending_state)
    self.state_machine.add_transition(reviewing_state, Event.DTMF_STAR,
                                     recording_state)
    self.state_machine.add_transition(greeting_state, Event.HANGUP,
                                     hungup_state)
    self.state_machine.add_transition(greeting_state,
                                     Event.PLAYBACK_COMPLETE,
                                     recording_state)
    self.state_machine.start(greeting_state)
```

## vm-call.js

› [Expand source](#)

```
//At the top of the file
var GreetingState = require('./greeting_state');

//In VoicemailCall::setup_state_machine()
this.setup_state_machine = function() {
    var hungup_state = new HungUpState(this);
    var recording_state = new RecordingState(this);
    var ending_state = new EndingState(this);
    var reviewing_state = new ReviewingState(this);
    var greeting_state = new GreetingState(this);
    this.state_machine = new StateMachine();
    this.state_machine.add_transition(recording_state, Event.DTMF_OCTOTHORPE,
reviewing_state);
    this.state_machine.add_transition(recording_state, Event.HANGUP, hungup_state);
    this.state_machine.add_transition(recording_state, Event.DTMF_STAR, recording_state);
    this.state_machine.add_transition(reviewing_state, Event.DTMF_OCTOTHORPE,
ending_state);
    this.state_machine.add_transition(reviewing_state, Event.HANGUP, hungup_state);
    this.state_machine.add_transition(reviewing_state, Event.DTMF_STAR, recording_state);
    this.state_machine.add_transition(greeting_state, Event.HANGUP, hungup_state);
    this.state_machine.add_transition(greeting_state, Event.PLAYBACK_COMPLETE,
recording_state);
    this.state_machine.start(greeting_state);
}
```

Here is a sample run where the user cuts off the greeting by pressing the '#' key, records a greeting and presses the '#' key, and after listening to the recording presses the '#' key once more.

```
Channel PJSIP/200-0000000b recording voicemail for 305
Entering greeting state
Cutting off greeting on DTMF #
Entering recording state
Recording voicemail at voicemail/305/1411503204.75
Accepted recording voicemail/305/1411503204.75 on DTMF #
Cleaning up event handlers
Entering reviewing state
Accepted recording voicemail/305/1411503204.75 on DTMF #
Ending voice mail call from PJSIP/200-0000000b
```

### Reader Exercise 1

Our current implementation of `GreetingState` does not take language into consideration. The `sounds_installed` method checks for the existence of the sound file, but it does not ensure that we have the sound file in the language of the channel that is in our application.

For this exercise, modify `sounds_installed()` to also check if the retrieved sound exists in the language of the calling channel. The channel's language can be retrieved using the `getChannelVar()` method on a channel to retrieve the value of variable "CHANNEL(language)". The sound returned by `sounds.get()` contains an array of `FormatLang` objects that are a pair of format and language strings. If the sound exists, but not in the channel's language, then throw an exception.

### Controlling playbacks

So far in our voice mail application, we have stopped playbacks, but there are a lot more interesting operations that can be done on them, such as reversing and fast-forwarding them. Within the context of recording a voicemail, these operations are pretty useless, so we will shift our focus now to the other side of voicemail: listening to recorded voicemails.

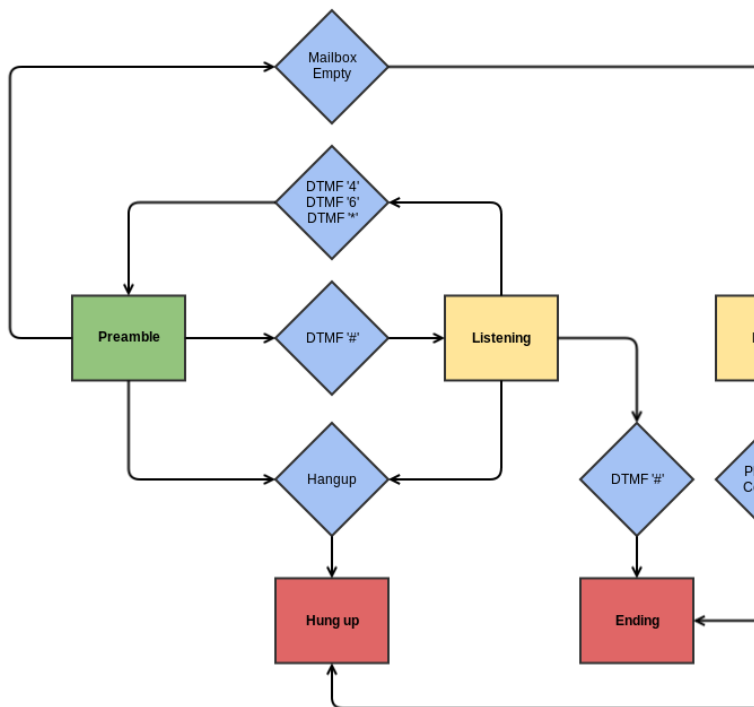
For this, we will write a new application. This new application will allow a caller to listen to the voicemails that are stored in a specific mailbox. When the caller enters the application, a prompt is played to the caller saying which message number the caller is hearing. When the message number finishes playing (or if the caller interrupts the playback with '#'), then the caller hears the specified message in the voicemail box. While listening to the voicemail, the caller can do several things:

- Press the '1' key to go back 3 seconds in the current message playback.
- Press the '2' key to pause or unpause the current message playback.
- Press the '3' key to go forward 3 seconds in the current message playback.
- Press the '4' key to play to the previous message.

- Press the '5' key to restart the current message playback.
- Press the '6' key to play to the next message.
- Press the '\*' key to delete the current message and play the next message.
- Press the '#' key to end the call.

If all messages in a mailbox are deleted or if the mailbox contained no messages to begin with, then "no more messages" is played back to the user, and the call is completed.

This means defining a brand new state machine. To start with, we'll define three new states. The "preamble" state is the initial state of the state machine, where the current message number is played back to the listener. The "listening" state is where the voice mail message is played back to the listener. The "empty" state is where no more messages remain in the mailbox. Here is the state machine we will be using:



Notice that while in the listening state, DTMF '4', '6', and '\*' all change the state to the preamble state. This is so that the new message number can be played back to the caller before the next message is heard. Also notice that the preamble state is responsible for determining if the state should change to empty. This keeps the logic in the listening state more straight-forward since it is already having to deal with a lot of DTMF events. It also gracefully handles the case where a caller calls into the application when the caller has no voicemail messages.

Here is the implementation of the application.

```

vm-playback.py > Expand source

#!/usr/bin/env python

import ari
import logging
import sys

from state_machine import StateMachine
from ending_state import EndingState
from hungup_state import HungUpState
from listening_state import ListeningState
from preamble_state import PreambleState
from empty_state import EmptyState
from event import Event

logging.basicConfig(level=logging.ERROR)
LOGGER = logging.getLogger(__name__)

client = ari.connect('http://localhost:8088', 'asterisk', 'asterisk')

```

```

class VoiceMailCall(object):
    def __init__(self, ari_client, channel, mailbox):
        self.client = ari_client
        self.channel = channel

        self.voicemails = []
        recordings = ari_client.recordings.listStored()
        vm_number = 1
        for rec in recordings:
            if rec.json['name'].startswith('voicemail/{0}'.format(mailbox)):
                self.voicemails.append((vm_number, rec.json['name']))
                vm_number += 1

        self.current_voicemail = 0
        self.setup_state_machine()

    def setup_state_machine(self):
        hungup_state = HungUpState(self)
        ending_state = EndingState(self)
        listening_state = ListeningState(self)
        preamble_state = PreambleState(self)
        empty_state = EmptyState(self)
        self.state_machine = StateMachine()
        self.state_machine.add_transition(listening_state, Event.DTMF_4,
                                         preamble_state)
        self.state_machine.add_transition(listening_state, Event.DTMF_6,
                                         preamble_state)
        self.state_machine.add_transition(listening_state, Event.HANGUP,
                                         hungup_state)
        self.state_machine.add_transition(listening_state, Event.DTMF_OCTOTHORPE,
                                         ending_state)
        self.state_machine.add_transition(listening_state, Event.DTMF_STAR,
                                         preamble_state)
        self.state_machine.add_transition(preamble_state, Event.DTMF_OCTOTHORPE,
                                         listening_state)
        self.state_machine.add_transition(preamble_state,
                                         Event.PLAYBACK_COMPLETE,
                                         listening_state)
        self.state_machine.add_transition(preamble_state, Event.MAILBOX_EMPTY,
                                         empty_state)
        self.state_machine.add_transition(preamble_state, Event.HANGUP,
                                         hungup_state)
        self.state_machine.add_transition(empty_state, Event.HANGUP,
                                         hungup_state)
        self.state_machine.add_transition(empty_state,
                                         Event.PLAYBACK_COMPLETE,
                                         ending_state)
        self.state_machine.start(preamble_state)

    def next_message(self):
        self.current_voicemail += 1
        if self.current_voicemail == len(self.voicemails):
            self.current_voicemail = 0

    def previous_message(self):
        self.current_voicemail -= 1
        if self.current_voicemail < 0:

```

```
        self.current_voicemail = len(self.voicemails) - 1

def delete_message(self):
    del self.voicemails[self.current_voicemail]
    if self.current_voicemail == len(self.voicemails):
        self.current_voicemail = 0

def get_current_voicemail_number(self):
    return self.voicemails[self.current_voicemail][0]

def get_current_voicemail_file(self):
    return self.voicemails[self.current_voicemail][1]

def mailbox_empty(self):
    return len(self.voicemails) == 0

def stasis_start_cb(channel_obj, event):
    channel = channel_obj['channel']
    channel_name = channel.json.get('name')
    mailbox = event.get('args')[0]
    print("Channel {0} recording voicemail for {1}".format(
        channel_name, mailbox))
    channel.answer()
    VoiceMailCall(client, channel, mailbox)
```

```
client.on_channel_event('StasisStart', stasis_start_cb)
client.run(apps=sys.argv[1])
```

### vm-playback.js

[Expand source](#)

```
/*jshint node:true*/
'use strict';

var ari = require('ari-client');
var util = require('util');
var path = require('path');

var Event = require('./event');
var StateMachine = require('./state_machine');
var EndingState = require('./ending_state');
var HungUpState = require('./hungup_state');
var PreambleState = require('./preamble_state');
var EmptyState = require('./empty_state');
var ListeningState = require('./listening_state');

ari.connect('http://localhost:8088', 'asterisk', 'asterisk', clientLoaded);

var VoiceMailCall = function(ari_client, channel, mailbox) {
  this.client = ari_client;
  this.channel = channel;
  var current_voicemail = 0;
  var voicemails = [];

  this.setup_state_machine = function() {
    var hungup_state = new HungUpState(this);
    var ending_state = new EndingState(this);
    var listening_state = new ListeningState(this);
    var preamble_state = new PreambleState(this);
    var empty_state = new EmptyState(this);
    this.state_machine = new StateMachine(this);
    this.state_machine.add_transition(listening_state, Event.DTMF_4, preamble_state);
    this.state_machine.add_transition(listening_state, Event.DTMF_6, preamble_state);
    this.state_machine.add_transition(listening_state, Event.HANGUP, hungup_state);
    this.state_machine.add_transition(listening_state, Event.DTMF_OCTOTHORPE,
ending_state);
    this.state_machine.add_transition(listening_state, Event.DTMF_STAR,
preamble_state);
    this.state_machine.add_transition(preamble_state, Event.DTMF_OCTOTHORPE,
listening_state);
    this.state_machine.add_transition(preamble_state, Event.PLAYBACK_COMPLETE,
listening_state);
    this.state_machine.add_transition(preamble_state, Event.MAILBOX_EMPTY,
empty_state);
    this.state_machine.add_transition(preamble_state, Event.HANGUP, hungup_state);
    this.state_machine.add_transition(empty_state, Event.HANGUP, hungup_state);
    this.state_machine.add_transition(empty_state, Event.PLAYBACK_COMPLETE,
ending_state);
    this.state_machine.start(preamble_state);
  }

  this.next_message = function() {
    current_voicemail++;
  }
}
```



```

        if (current_voicemail === voicemails.length) {
            current_voicemail = 0;
        }
    }

    this.previous_message = function() {
        current_voicemail--;
        if (current_voicemail < 0) {
            current_voicemail = voicemails.length - 1;
        }
    }

    this.delete_message = function() {
        voicemails.splice(current_voicemail, 1);
        if (current_voicemail === voicemails.length) {
            current_voicemail = 0;
        }
    }

    this.get_current_voicemail_file = function() {
        return voicemails[current_voicemail]['file'];
    }

    this.mailbox_empty = function() {
        return voicemails.length === 0;
    }

    var self = this;
    ari_client.recordings.listStored(function (err, recordings) {
        var vm_number = 1;
        var regex = new RegExp('^voicemail/' + mailbox);
        for (var i = 0; i < recordings.length; i++) {
            var rec_name = recordings[i].name;
            if (rec_name.search(regex) !== -1) {
                console.log("Found voicemail", rec_name);
                voicemails.push({'number': vm_number, 'file': rec_name});
                vm_number++;
            }
        }
        self.setup_state_machine();
    });
}

function clientLoaded(err, client) {
    if (err) {
        throw err;
    }
    client.on('StasisStart', stasisStart);
    function stasisStart(event, channel) {
        var mailbox = event.args[0]
        channel.answer(function(err) {
            if (err) {
                throw err;
            }
            new VoiceMailCall(client, channel, mailbox);
        });
    }
}

```

```

    }
    client.start(process.argv[2]);
}

```

Quite a bit of this is similar to what we were using for our voice mail recording application. The biggest difference here is that the call has many more methods defined since playing back voice mails is more complicated than recording a single one.

Now that we have the state machine defined and the application written, let's actually write the required new states. First of the new states is the "preamble" state.

### preamble\_state.py

[Expand source](#)

```

from event import Event
import uuid

def sounds_installed(client):
    try:
        client.sounds.get(soundId='vm-message')
    except:
        print "Required sound 'vm-message' not installed. Aborting"
        raise

class PreambleState(object):
    state_name = "preamble"

    def __init__(self, call):
        self.call = call
        self.hangup_event = None
        self.playback_finished = None
        self.dtmf_event = None
        self.playback = None
        sounds_installed(call.client)

    def enter(self):
        print "Entering preamble state"
        if self.call.mailbox_empty():
            self.call.state_machine.change_state(Event.MAILBOX_EMPTY)
            return
        self.hangup_event = self.call.channel.on_event("ChannelHangupRequest",
                                                       self.on_hangup)
        self.playback_finished = self.call.client.on_event(
            'PlaybackFinished', self.on_playback_finished)
        self.dtmf_event = self.call.channel.on_event('ChannelDtmfReceived',
                                                    self.on_dtmf)
        self.initialize_playbacks()

    def initialize_playbacks(self):
        self.current_playback = 0
        current_voicemail = self.call.get_current_voicemail_number()
        self.sounds_to_play = [
            {
                'id': str(uuid.uuid4()),
                'media': 'sound:vm-message'
            },
            {
                'id': str(uuid.uuid4()),
                'media': 'number:{0}'.format(current_voicemail)
            }
        ]

```

```

    ]
    self.start_playback()

def start_playback(self):
    current_sound = self.sounds_to_play[self.current_playback]
    self.playback = self.call.channel.playWithId(
        playbackId=current_sound['id'],
        media=current_sound['media']
    )

def cleanup(self):
    self.playback_finished.close()
    if self.playback:
        self.playback.stop()
    self.dtmf_event.close()
    self.hangup_event.close()

def on_hangup(self, channel, event):
    self.playback = None
    self.cleanup()
    self.call.state_machine.change_state(Event.HANGUP)

def on_playback_finished(self, event):
    current_sound = self.sounds_to_play[self.current_playback]
    if current_sound['id'] == event.get('playback').get('id'):
        self.playback = None
        self.current_playback += 1
        if self.current_playback == len(self.sounds_to_play):
            self.cleanup()
            self.call.state_machine.change_state(Event.PLAYBACK_COMPLETE)
        else:
            self.start_playback()

def on_dtmf(self, channel, event):
    digit = event.get('digit')

```

```
    if digit == '#':
        self.cleanup()
        self.call.state_machine.change_state(Event.DTMF_OCTOTHORPE)
```

### preamble\_state.js

[Expand source](#)

```
var Event = require('./event');

function sounds_installed(client) {
    client.sounds.get({soundId: 'vm-message'}, function(err) {
        if (err) {
            console.log("Required sound 'vm-message' not installed. Aborting");
            throw err;
        }
    });
}

function PreambleState(call) {
    this.state_name = "preamble";
    this.enter = function() {
        var current_playback;
        var sounds_to_play;
        var playback;
        console.log("Entering preamble state");
        if (call.mailbox_empty()) {
            call.state_machine.change_state(Event.MAILBOX_EMPTY);
            return;
        }
        call.channel.on("ChannelHangupRequest", on_hangup);
        call.client.on("PlaybackFinished", on_playback_finished);
        call.channel.on("ChannelDtmfReceived", on_dtmf);
        initialize_playbacks();

        function initialize_playbacks() {
            current_playback = 0;
            sounds_to_play = [
                {
                    'playback': call.client.Playback(),
                    'media': 'sound:vm-message'
                },
                {
                    'playback': call.client.Playback(),
                    'media': 'number:' + call.get_current_voicemail_number()
                }
            ];
            start_playback();
        }

        function start_playback() {
            current_sound = sounds_to_play[current_playback];
            playback = current_sound['playback'];
            call.channel.play({media: current_sound['media']}, playback);
        }

        function cleanup() {
            call.channel.removeListener('ChannelHangupRequest', on_hangup);
            call.channel.removeListener('ChannelDtmfReceived', on_dtmf);
            call.client.removeListener('PlaybackFinished', on_playback_finished);
        }
    };
}
```

```

        if (playback) {
            playback.stop();
        }
    }

    function on_hangup(event, channel) {
        playback = null;
        cleanup();
        call.state_machine.change_state(Event.HANGUP);
    }

    function on_playback_finished(event) {
        var current_sound = sounds_to_play[current_playback];
        if (playback && (playback.id === event.playback.id)) {
            playback = null;
            current_playback++;
            if (current_playback === sounds_to_play.length) {
                cleanup();
                call.state_machine.change_state(Event.PLAYBACK_COMPLETE);
            } else {
                start_playback();
            }
        }
    }

    function on_dtmf(event, channel) {
        switch(event.digit) {
            case '#':
                cleanup();
                call.state_machine.change_state(Event.DTMF_OCTOTHORPE);
        }
    }
}

```

```
module.exports = PreambleState;
```

`PreambleState` should look similar to the `GreetingState` introduced previously on this page. The biggest difference is that the code is structured to play multiple sound files instead of just a single one. Note that it is acceptable to call `channel.play()` while a playback is playing on a channel in order to queue a second playback. For our application though, we have elected to play the second sound only after the first has completed. The reason for this is that if there is only a single active playback at any given time, then it becomes easier to clean up the current state when an event occurs that causes a state change.

Next, here is the "empty" state code:

**empty\_state.py**[› Expand source](#)

```
from event import Event
import uuid

def sounds_installed(client):
    try:
        client.sounds.get(soundId='vm-nomore')
    except:
        print "Required sound 'vm-nomore' not installed. Aborting"
        raise

class EmptyState(object):
    state_name = "empty"

    def __init__(self, call):
        self.call = call
        self.playback_id = None
        self.hangup_event = None
        self.playback_finished = None
        self.playback = None
        sounds_installed(call.client)

    def enter(self):
        self.playback_id = str(uuid.uuid4())
        print "Entering empty state"
        self.hangup_event = self.call.channel.on_event("ChannelHangupRequest",
            self.on_hangup)
        self.playback_finished = self.call.client.on_event(
            'PlaybackFinished', self.on_playback_finished)
        self.playback = self.call.channel.playWithId(
            playbackId=self.playback_id, media="sound:vm-nomore")

    def cleanup(self):
        self.playback_finished.close()
        if self.playback:
            self.playback.stop()
        self.hangup_event.close()

    def on_hangup(self, channel, event):
        # Setting playback to None stops cleanup() from trying to stop the
        # playback.
        self.playback = None
        self.cleanup()
        self.call.state_machine.change_state(Event.HANGUP)

    def on_playback_finished(self, event):
        if self.playback_id == event.get('playback').get('id'):
            self.playback = None
            self.cleanup()
            self.call.state_machine.change_state(Event.PLAYBACK_COMPLETE)
```

**empty\_state.js**[Expand source](#)

```
var Event = require('./event');

function sounds_installed(client) {
  client.sounds.get({soundId: 'vm-nomore'}, function(err) {
    if (err) {
      console.log("Required sound 'vm-nomore' not installed. Aborting");
      throw err;
    }
  });
}

function EmptyState(call) {
  this.state_name = "empty";

  this.enter = function() {
    console.log("Entering empty state");
    playback = call.client.Playback();
    call.channel.on("ChannelHangup", on_hangup);
    call.client.on("PlaybackFinished", on_playback_finished);
    call.channel.play({media: 'sound:vm-nomore'}, playback);

    function cleanup() {
      call.channel.removeListener('ChannelHangupRequest', on_hangup);
      call.channel.removeListener('PlaybackFinished', on_playback_finished);
      if (playback) {
        playback.stop();
      }
    }

    function on_hangup(event, channel) {
      playback = null;
      cleanup();
      call.state_machine.change_state(Event.HANGUP);
    }

    function on_playback_finished(event) {
      if (playback && playback.id === event.playback.id) {
        playback = null;
        cleanup();
        call.state_machine.change_state(Event.PLAYBACK_COMPLETE);
      }
    }
  }
}

module.exports = EmptyState;
```

This state does not introduce anything we haven't seen already, so let's move on to the "listening" state code:

**listening\_state.py**[Expand source](#)

```
from event import Event
import uuid

class ListeningState(object):
    state_name = "listening"
```



```

def __init__(self, call):
    self.call = call
    self.playback_id = None
    self.hangup_event = None
    self.playback_finished = None
    self.dtmf_event = None
    self.playback = None

def enter(self):
    self.paused = False
    self.playback_id = str(uuid.uuid4())
    print "Entering listening state"
    self.hangup_event = self.call.channel.on_event("ChannelHangupRequest",
        self.on_hangup)
    self.playback_finished = self.call.client.on_event(
        'PlaybackFinished', self.on_playback_finished)
    self.dtmf_event = self.call.channel.on_event('ChannelDtmfReceived',
        self.on_dtmf)

    self.playback = self.call.channel.playWithId(
        playbackId=self.playback_id, media="recording:{0}".format(
            self.call.get_current_voicemail_file()))

def cleanup(self):
    self.playback_finished.close()
    if self.playback:
        self.playback.stop()
    self.dtmf_event.close()
    self.hangup_event.close()

def on_hangup(self, channel, event):
    self.cleanup()
    self.call.state_machine.change_state(Event.HANGUP)

def on_playback_finished(self, event):
    if self.playback_id == event.get('playback').get('id'):
        self.playback = None

def on_dtmf(self, channel, event):
    digit = event.get('digit')
    if digit == '1':
        if self.playback:
            self.playback.control(operation='reverse')
    elif digit == '2':
        if not self.playback:
            return
        if self.paused:
            self.playback.control(operation='unpause')
            self.paused = False
        else:
            self.playback.control(operation='pause')
            self.paused = True
    elif digit == '3':
        if self.playback:
            self.playback.control(operation='forward')
    elif digit == '4':
        self.cleanup()
        self.call.previous_message()
        self.call.state_machine.change_state(Event.DTMF_4)

```

```
elif digit == '5':
    if self.playback:
        self.playback.control(operation='restart')
elif digit == '6':
    self.cleanup()
    self.call.next_message()
    self.call.state_machine.change_state(Event.DTMF_6)
elif digit == '#':
    self.cleanup()
    self.call.state_machine.change_state(Event.DTMF_OCTOTHORPE)
elif digit == '*':
    print ("Deleting stored recording {0}".format(
        self.call.get_current_voicemail_file()))
    self.cleanup()
    self.call.client.recordings.deleteStored(
```

```
        recordingName=self.call.get_current_voicemail_file())
self.call.delete_message()
self.call.state_machine.change_state(Event.DTMF_STAR)
```

## listening\_state.js

[Expand source](#)

```
var Event = require('./event');

var ListeningState = function(call) {
  this.state_name = "listening";
  this.call = call;
}

function ListeningState(call) {
  this.state_name = "listening";
  this.enter = function() {
    var playback = call.client.Playback();
    var url = "recording:" + call.get_current_voicemail_file();

    console.log("Entering Listening state");
    call.channel.on("ChannelHangupRequest", on_hangup);
    call.channel.on("ChannelDtmfReceived", on_dtmf);
    call.client.on("PlaybackFinished", on_playback_finished);
    console.log("Playing file %s", url);
    call.channel.play({media: url}, playback, function(err) {
      if (err) {
        console.error(err);
      }
    });

    function cleanup() {
      call.channel.removeListener('ChannelHangupRequest', on_hangup);
      call.channel.removeListener('ChannelDtmfReceived', on_dtmf);
      call.client.removeListener('PlaybackFinished', on_playback_finished);
      if (playback) {
        playback.stop();
      }
    }

    function on_hangup(event, channel) {
      playback = null;
      cleanup();
      call.state_machine.change_state(Event.HANGUP);
    }

    function on_playback_finished(event) {
      if (playback && (playback.id === event.playback.id)) {
        playback = null;
      }
    }

    function on_dtmf(event, channel) {
      switch (event.digit) {
        case '1':
          if (playback) {
            playback.control({operation: 'reverse'});
          }
          break;
      }
    }
  }
}
```

```

    case '2':
        if (!playback) {
            break;
        }
        if (paused) {
            playback.control({operation: 'unpause'});
            paused = false;
        } else {
            playback.control({operation: 'pause'});
            paused = true;
        }
        break;
    case '3':
        if (playback) {
            playback.control({operation: 'forward'});
        }
        break;
    case '4':
        cleanup();
        call.previous_message();
        call.state_machine.change_state(Event.DTMF_4);
        break;
    case '5':
        if (playback) {
            playback.control({operation: 'restart'});
        }
        break;
    case '6':
        cleanup();
        call.next_message();
        call.state_machine.change_state(Event.DTMF_6);
        break;
    case '#':
        cleanup();
        call.state_machine.change_state(Event.DTMF_OCTOTHORPE);
        break;
    case '*':
        console.log("Deleting stored recording",
call.get_current_voicemail_file());
        cleanup();

call.client.recordings.deleteStored({recordingName:call.get_current_voicemail_file()});
        call.delete_message();
        call.state_machine.change_state(Event.DTMF_STAR);
    }
}
}

```

```
}  
  
module.exports = ListeningState;
```

`ListeningState` is where we introduce new playback control concepts. Playbacks have their controlling operations wrapped in a single method, `control()`, rather than having lots of separate operations. All control operations (reverse, pause, unpause, forward, and restart) are demonstrated by this state.

### Reader Exercise 2

You may have noticed while exploring the playbacks API that the `control()` method takes no parameters other than the operation. This means that certain properties of operations are determined when the playback is started on the channel.

For this exercise, modify the `ListeningState` so that DTMF '1' and '3' reverse or forward the playback by 5000 milliseconds instead of the default 3000 milliseconds.

### Playbacks on bridges

Just as channels allow for playbacks to be performed on them, bridges also have the capacity to have sounds, recordings, tones, numbers, etc. played back on them. The difference is that all participants in the bridge will hear the playback instead of just a single channel. In bridging situations, it can be useful to play certain sounds to an entire bridge (e.g. Telling participants the call is being recorded), but it can also be useful to play sounds to specific participants (e.g. Telling a caller he has joined a conference bridge). A playback on a bridge can be stopped or controlled exactly the same as a playback on a channel.

### Reader Exercise 3

If you've read through the Recording and Playbacks pages, then you should have a good grasp on the operations available, as well as a decent state machine implementation. For our final exercise, instead of adding onto the existing voice mail applications, create a new application that uses some of the recording and playback operations that you have learned about.

You will be creating a rudimentary call queue application. The queue application will have two types of participants: agents and callers. Agents and callers call into the same `Stasis` application and are distinguished based on arguments to the `Stasis` application (e.g. A caller might call `Stasis(queue,caller)` and an agent might call `Stasis(queue,agent)` from the dialplan).

- When an agent calls into the `Stasis` application, the agent is placed into an agent queue.
  - While in the agent queue, the agent should hear music on hold. Information about how to play music on hold to a channel can be found here.
- When a caller calls into the `Stasis` application, the caller is placed into a caller queue.
  - While in the caller queue, the caller should hear music on hold.
  - Every 30 seconds, the caller should hear an announcement. If the caller is at the front of the queue, the "queue-youarenext" sound should play. If the caller is not at the front of the queue, then the caller should hear the "queue-thereare" sound, then the number of callers ahead in the queue, then the "queue-callswaiting" sound.
- If there is a caller in the caller queue and an agent in the agent queue, then the caller and agent at the front of their respective queues are removed and placed into a bridge with each other. For more information on bridging channels, see [his page](#). If this happens while a caller has a sound file playing, then this should cause the sound file to immediately stop playing in order to bridge the call.
- Once bridged, the agent can perform the following:
  - Pressing '0' should start recording the bridge. Upon recording the bridge, the caller should hear a beep sound.
  - Pressing '5' while the call is being recorded should pause the recording. Pressing '5' a second time should unpause the recording.
  - Pressing '7' while the call is being recorded should stop the recording.
  - Pressing '9' while the call is being recorded should discard the recording.
  - Pressing '#' at any time (whether the call is being recorded or not) should end the call, causing the agent to be placed into the back of the agent queue and the caller to be hung up.
  - If the agent hangs up, then the caller should also be hung up.
- Once bridged, if the caller hangs up, then the agent should be placed into the back of the agent queue.

Best of luck!

# The Asterisk Resource

## The Asterisk Resource

While the [primary purpose of ARI](#) is to allow developers to build their own communications applications using Asterisk as a media engine, there are other resources in the API that are useful outside of this use case. One of these is the `asterisk` resource. This resource not only provides information about the running Asterisk instance, but also exposes resources and operations that allow an external system to manipulate the overall Asterisk system.

### On This Page

- [The Asterisk Resource](#)
- [Retrieving System Information](#)

### More Detail

- [ARI Push Configuration](#)

## Retrieving System Information

The `asterisk` resource provides the ability to retrieve basic information about the running Asterisk process. This includes:

- Information about how Asterisk was compiled
- Information about Asterisk's configuration
- Current status of the Asterisk process
- Information about the system the Asterisk process is running on

An example of this is shown below:


```
$ curl -X GET -u asterisk:SECRET https://localhost:8088/ari/asterisk/info
{
  "status":
  {
    "startup_time": "2015-07-16T21:01:37.273-0500",
    "last_reload_time": "2015-07-16T21:01:37.273-0500"
  },
  "build":
  {
    "user": "mjordan",
    "options": "AST_DEVMODE, LOADABLE_MODULES, OPTIONAL_API, TEST_FRAMEWORK",
    "machine": "x86_64",
    "os": "Linux",
    "kernel": "3.13.0-24-generic",
    "date": "2015-07-11 15:51:57 UTC"
  },
  "system":
  {
    "version": "GIT-master-0b2cbeaM",
    "entity_id": "ec:f4:bb:67:a6:d0"
  },
  "config":
  {
    "default_language": "en",
    "name": "mjordan-laptop",
    "setid":
    {
      "user": "asterisk",
      "group": "asterisk"
    }
  }
}
```

# ARI Push Configuration

## Overview

Asterisk typically retrieves its configuration information by *pulling* it from some configuration source - whether that be a static configuration file or a relational database. This page describes an alternative way to provide configuration information to Asterisk using a *push* model through ARI. Note that only modules whose configuration is managed by the [Sorcery](#) data abstraction framework in Asterisk can make use of this mechanism. Predominately, this implies configuration of the PJSIP stack.

On This Page
<ul style="list-style-type: none"><li>• Overview</li><li>• Push Configuration Workflow</li><li>• Asterisk Configuration</li><li>• Pushing PJSIP Configuration<ul style="list-style-type: none"><li>• Original PJSIP Configuration</li><li>• New Configuration<ul style="list-style-type: none"><li>• F</li><li>• S</li><li>• A</li></ul></li><li>• Pushing Configuration</li><li>• Deleting Configuration</li></ul></li></ul>

Version Information
 This feature was introduced in ARI version 1.8.0, or Asterisk 13.5.0 or later.

## Push Configuration Workflow

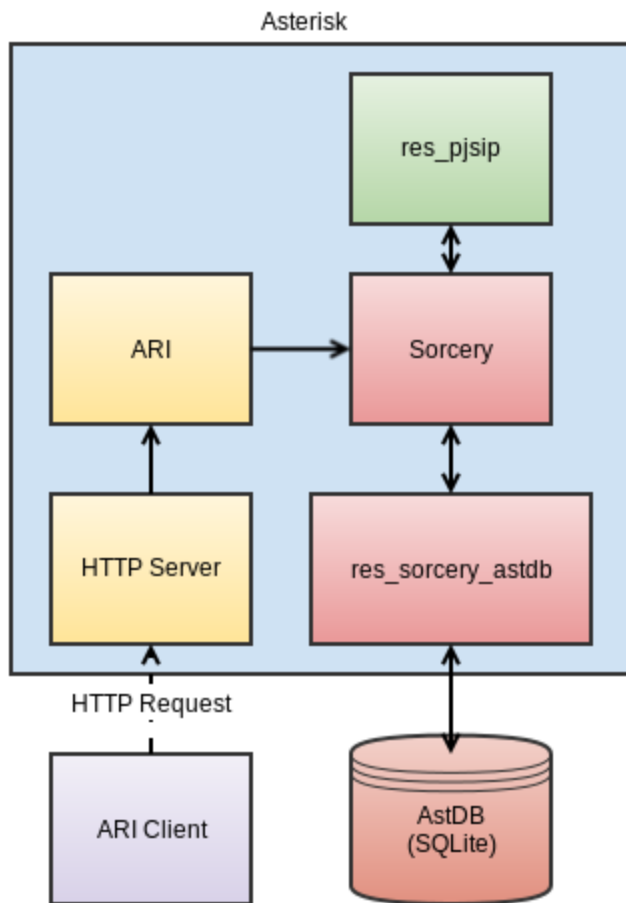
With push configuration, an external process uses ARI to perform a configuration operation. The configuration operation could be any one of the four classic operations for persistent storage - [Create](#), [Retrieve](#), [Update](#), or [Delete](#). For the purposes of this workflow, we'll assume that the operation is to create a configuration change in Asterisk.

The ARI client makes a `PUT` request, where the body contains the configuration object to create, encoded in JSON. This is first handled by ARI, which performs basic validation on the inbound request and its contents. Once the request is validated, ARI asks the Sorcery framework to create the actual object.

Sorcery requires three pieces of information, at a minimum, to create an object:

- The overall class of configuration types. This is usually a module or namespace that provides multiple types of objects to be created, e.g., 'res\_pjsip'.
- The type of configuration object to create, e.g., 'endpoint'.
- A unique identifier (amongst objects of the same type) for the object, e.g., 'alice'.

Once Sorcery has determined that it knows how to create the type of object, it creates it using the provided data in the JSON body. If some piece of data in the body can't be converted to an attribute on the object, the inbound request is rejected.



If the object is created successfully, Sorcery then has to determine what to do with it. While we've just had a piece of configuration pushed to Asterisk, Sorcery is responsible for storing it in some permanent - or semi-permanent - storage. For this, it looks to its configuration in `sorcery.conf`. We'll assume that our object should be created in the AstDB, a SQLite database. In that case, Asterisk pushes the newly created object to `res_sorcery_astdb`, which is the Sorcery driver for the AstDB. This module then writes the information to the SQLite database.

When the PJSIP stack next needs the object - such as when an `INVITE` request is received that maps to Alice's endpoint - it queries Sorcery for the object. At this point, from Sorcery's perspective, the retrieval of the configuration information is exactly the same as if a static configuration file or a relational database was providing the information, and it returns the pushed configuration information.

## Asterisk Configuration

To make use of push configuration, you **must** configure Sorcery to persist the pushed configuration somewhere. If you don't want the information to persist beyond reloads, you can use the in-memory Sorcery driver, `res_sorcery_memory`. The example below assumes that we will push configuration to the PJSIP stack, and that we want information to persist even if Asterisk is restarted. For that reason, we'll use the AstDB.

```

sorcery.conf

[res_pjsip]

endpoint=astdb,ps_endpoints
auth=astdb,ps_auths
aor=astdb,ps_aors
domain_alias=astdb,ps_domain_aliases
contact=astdb,ps_contacts
system=astdb,ps_systems

[res_pjsip_endpoint_identifier_ip]
identify=astdb,ps_endpoint_id_ips

[res_pjsip_outbound_registration]
registration=astdb,ps_registrations

```

⚠ You **must** configure `sorcery.conf` for this feature to work. The standard data provider Sorcery uses for PJSIP objects is the static configuration file driver. This driver does not support creation, updating, or deletion of objects - which means only the `GET` request will succeed. Any of the following drivers will work to support push configuration:

- `res_sorcery_memory`
- `res_sorcery_astdb`



- res\_sorcery\_realtime

## Pushing PJSIP Configuration

This walk-through will show how we can use the `asterisk` resource in ARI to push a PJSIP endpoint into the AstDB, and then later remove the endpoint.

### Original PJSIP Configuration

Assume we have the following static PJSIP configuration file that defines an endpoint for Alice:

```
pjsip.conf

[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0:5060

[transport-tcp]
type=transport
protocol=tcp
bind=0.0.0.0:5060

[alice]
type=aor
support_path=yes
remove_existing=yes
max_contacts=1

[alice]
type=auth
auth_type=userpass
username=alice
password=secret

[alice]
type=endpoint
from_user=alice
allow=!all,g722,ulaw,alaw
ice_support=yes
force_rport=yes
rewrite_contact=yes
rtp_symmetric=yes
context=default
auth=alice
aors=alice
```

If we then ask Asterisk what endpoints we have, it will show us something like the following:

## Asterisk CLI

```
*CLI> pjsip show endpoints
Endpoint: <Endpoint/CID.....> <State.....>
<Channels.>
  I/OAuth:
<AuthId/UserName.....>
  Aor: <Aor.....> <MaxContact>
  Contact: <Aor/ContactUri.....> <Status.....>
<RTT(ms)..>
  Transport: <TransportId.....> <Type> <cos> <tos>
<BindAddress.....>
  Identify:
<Identify/Endpoint.....>
  Match: <ip/cidr.....>
  Channel: <ChannelId.....> <State.....>
<Time(sec)>
  Exten: <DialedExten.....> CLCID: <ConnectedLineCID.....>
=====
====
Endpoint:  alice                               Unavailable  0 of
inf
  InAuth:  alice/alice
  Aor:    alice                               1
*CLI>
```

Our goal is to recreate alice, using ARI.

## New Configuration

### PJSIP

Remove Alice from pjsip.conf:

#### pjsip.conf

```
[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0:5060

[transport-tcp]
type=transport
protocol=tcp
bind=0.0.0.0:5060
```

### Sorcery

Tell the Sorcery data abstraction framework to pull *endpoint*, *aor*, and *auth* objects from the Asterisk Database:

#### sorcery.conf

```
[res_pjsip]
endpoint=astdb,ps_endpoints
auth=astdb,ps_auths
aor=astdb,ps_aors
```

### Asterisk CLI

Now, if we ask Asterisk for the PJSIP endpoints, it will tell us none are defined:

### Asterisk CLI

```
*CLI> pjsip show endpoints
No objects found.
```

## Pushing Configuration

First, let's push in Alice's authentication:

```
$ curl -X PUT -H "Content-Type: application/json" -u asterisk:secret -d '{"fields": [
{ "attribute": "auth_type", "value": "userpass"}, {"attribute": "username", "value":
"alice"}, {"attribute": "password", "value": "secret" } ] }'
https://localhost:8088/ari/asterisk/config/dynamic/res_pjsip/auth/alice

[{"attribute": "md5_cred", "value": ""}, {"attribute": "realm", "value": ""}, {"attribute": "au
th_type", "value": "userpass"}, {"attribute": "password", "value": "secret"}, {"attribute": "n
once_lifetime", "value": "32"}, {"attribute": "username", "value": "alice"}]
```

We can note a few things from this:

1. We supply the non-default values that make up Alice's authentication in the JSON body of the request. The body specifies the "fields" to update, which is a list of attributes to modify on the object we're creating.
2. We don't have to provide default values for the object. This includes the "type" attribute - ARI is smart enough to figure out what that is from the request URI, where we specify that we are creating an "auth" object.
3. When we've created the object successfully, ARI returns back all the attributes that make up that object as a list of attribute/value pairs - even the attributes we didn't specify.

Next, we can push in Alice's AoRs:

```
$ curl -X PUT -H "Content-Type: application/json" -u asterisk:secret -d '{"fields": [
{ "attribute": "support_path", "value": "yes"}, {"attribute": "remove_existing",
"value": "yes"}, {"attribute": "max_contacts", "value": "1" } ] }'
https://localhost:8088/ari/asterisk/config/dynamic/res_pjsip/aor/alice

[{"attribute": "support_path", "value": "true"}, {"attribute": "default_expiration", "value"
:"3600"}, {"attribute": "qualify_timeout", "value": "3.000000"}, {"attribute": "mailboxes", "
value": ""}, {"attribute": "minimum_expiration", "value": "60"}, {"attribute": "outbound_prox
y", "value": ""}, {"attribute": "maximum_expiration", "value": "7200"}, {"attribute": "qualify
_frequency", "value": "0"}, {"attribute": "authenticate_qualify", "value": "false"}, {"attrib
ute": "contact", "value": ""}, {"attribute": "max_contacts", "value": "1"}, {"attribute": "remo
ve_existing", "value": "true"}]
```

Finally, we can push in Alice's endpoint:

```

$ curl -X PUT -H "Content-Type: application/json" -u asterisk:secret -d '{"fields": [
{ "attribute": "from_user", "value": "alice" }, { "attribute": "allow", "value":
"!all,g722,ulaw,alaw"}, { "attribute": "ice_support", "value": "yes"}, {"attribute":
"force_rport", "value": "yes"}, {"attribute": "rewrite_contact", "value": "yes"},
{"attribute": "rtp_symmetric", "value": "yes"}, {"attribute": "context", "value":
"default" }, {"attribute": "auth", "value": "alice" }, {"attribute": "aors", "value":
"alice" } ] }'
https://localhost:8088/ari/asterisk/config/dynamic/res_pjsip/endpoint/alice

[{"attribute": "timers_sess_expires", "value": "1800"}, {"attribute": "device_state_busy_at", "value": "0"}, {"attribute": "dtls_cipher", "value": ""}, {"attribute": "from_domain", "value": ""}, {"attribute": "dtls_rekey", "value": "0"}, {"attribute": "dtls_fingerprint", "value": "SHA-256"}, {"attribute": "direct_media_method", "value": "invite"}, {"attribute": "send_rpid", "value": "false"}, {"attribute": "pickup_group", "value": ""}, {"attribute": "sdp_session", "value": "Asterisk"}, {"attribute": "dtls_verify", "value": "No"}, {"attribute": "message_context", "value": ""}, {"attribute": "mailboxes", "value": ""}, {"attribute": "named_pickup_group", "value": ""}, {"attribute": "record_on_feature", "value": "automixmon"}, {"attribute": "dtls_private_key", "value": ""}, {"attribute": "named_call_group", "value": ""}, {"attribute": "t38_udptl_maxdatagram", "value": "0"}, {"attribute": "media_encryption_optimistic", "value": "false"}, {"attribute": "aors", "value": "alice"}, {"attribute": "rpid_immediate", "value": "false"}, {"attribute": "outbound_proxy", "value": ""}, {"attribute": "identify_by", "value": "username"}, {"attribute": "inband_progress", "value": "false"}, {"attribute": "rtp_symmetric", "value": "true"}, {"attribute": "transport", "value": ""}, {"attribute": "t38_udptl_ec", "value": "none"}, {"attribute": "fax_detect", "value": "false"}, {"attribute": "t38_udptl_nat", "value": "false"}, {"attribute": "allow_transfer", "value": "true"}, {"attribute": "tos_video", "value": "0"}, {"attribute": "srtp_tag_32", "value": "false"}, {"attribute": "timers_min_session", "value": "90"}, {"attribute": "call_group", "value": ""}, {"attribute": "sub_min_expiry", "value": "0"}, {"attribute": "100rel", "value": "yes"}, {"attribute": "direct_media", "value": "true"}, {"attribute": "g726_non_standard", "value": "false"}, {"attribute": "dtmf_mode", "value": "rfc4733"}, {"attribute": "dtls_cert_file", "value": ""}, {"attribute": "media_encryption", "value": "no"}, {"attribute": "media_use_received_transport", "value": "false"}, {"attribute": "direct_media_glare_mitigation", "value": "none"}, {"attribute": "trust_id_inbound", "value": "false"}, {"attribute": "force_avp", "value": "false"}, {"attribute": "record_off_feature", "value": "automixmon"}, {"attribute": "send_diversion", "value": "true"}, {"attribute": "language", "value": ""}, {"attribute": "mwi_from_user", "value": ""}, {"attribute": "rtp_ipv6", "value": "false"}, {"attribute": "ice_support", "value": "true"}, {"attribute": "callerid", "value": "<unknown>"}, {"attribute": "aggregate_mwi", "value": "true"}, {"attribute": "one_touch_recording", "value": "false"}, {"attribute": "moh_passthrough", "value": "false"}, {"attribute": "cos_video", "value": "0"}, {"attribute": "accountcode", "value": ""}, {"attribute": "allow", "value": "(g722|ulaw|alaw)"}, {"attribute": "rewrite_contact", "value": "true"}, {"attribute": "t38_udptl_ipv6", "value": "false"}, {"attribute": "tone_zone", "value": ""}, {"attribute": "user_eq_phone", "value": "false"}, {"attribute": "allow_subscribe", "value": "true"}, {"attribute": "rtp_engine", "value": "asterisk"}, {"attribute": "auth", "value": "alice"}, {"attribute": "from_user", "value": "alice"}, {"attribute": "disable_direct_media_on_nat", "value": "false"}, {"attribute": "set_var", "value": ""}, {"attribute": "use_ptime", "value": "false"}, {"attribute": "outbound_auth", "value": ""}, {"attribute": "media_address", "value": ""}, {"attribute": "tos_audio", "value": "0"}, {"attribute": "dtls_ca_path", "value": ""}, {"attribute": "dtls_setup", "value": "active"}, {"attribute": "force_rport", "value": "true"}, {"attribute": "connected_line_method", "value": "invite"}, {"attribute": "callerid_tag", "value": ""}, {"attribute": "timers", "value": "yes"}, {"attribute": "sdp_owner", "value": "-"}, {"attribute": "trust_id_outbound", "value": "false"}, {"attribute": "use_avpf", "value": "false"}, {"attribute": "context", "value": "default"}, {"attribute": "moh_suggest", "value": "default"}, {"attribute": "send_pai", "value": "false"}, {"attribute": "t38_udptl", "value": "false"}, {"attribute": "dtls_ca_file", "value": ""}, {"attribute": "callerid_privacy", "value": "allowed_no_tscreened"}, {"attribute": "cos_audio", "value": "0"}]

```

We can now verify that Alice's endpoint exists:

## Asterisk CLI

```
*CLI> pjsip show endpoints
Endpoint: <Endpoint/CID.....> <State.....>
<Channels.>
  I/OAuth:
<AuthId/UserName.....>
  Aor: <Aor.....> <MaxContact>
  Contact: <Aor/ContactUri.....> <Status.....>
<RTT(ms)..>
  Transport: <TransportId.....> <Type> <cos> <tos>
<BindAddress.....>
  Identify:
<Identify/Endpoint.....>
  Match: <ip/cidr.....>
  Channel: <ChannelId.....> <State.....>
<Time(sec)>
  Exten: <DialedExten.....> CLCID: <ConnectedLineCID.....>
=====
====
Endpoint:  alice/unknown                               Invalid      0 of
inf
  InAuth:  alice/alice
  Aor:    alice                                         1
```



### Order Matters!

While ARI itself won't care about the order you create objects in, PJSIP will. A PJSIP endpoint is used to look-up the endpoint's authentication and AoRs. Asterisk and ARI will let you create the endpoint first, referencing an authentication and AoR object that don't yet exist. If an inbound request arrives for that endpoint, the request will be rejected because the endpoint won't be able to find the authentication or store the Contact on a REGISTER request! By creating things in the order that we did, we can guarantee that everything will be in place when the endpoint is instantiated.

We can also verify that Alice exists in the AstDB:

## Asterisk CLI

```
*CLI> database show
/ps_aors/aor/alice           :
{"qualify_frequency":"0","maximum_expiration":"7200","minimum_expiration":"60","qualify_timeout":"3.000000","support_path":"true","default_expiration":"3600","mailboxes":"","authenticate_qualify":"false","outbound_proxy":"","max_contacts":"1","remove_existing":"true"}
/ps_auths/auth/alice        :
{"realm":"","md5_cred":"","nonce_lifetime":"32","auth_type":"userpass","password":"secret","username":"alice"}
/ps_endpoints/endpoint/alice :
{"send_diversion":"true","device_state_busy_at":"0","direct_media_method":"invite","sdp_owner":"-","pickup_group":"","timers_sess_expires":"1800","message_context":"","accountcode":"","dtls_fingerprint":"SHA-256","rpid_immediate":"false","force_avp":"false","aors":"alice","trust_id_inbound":"false","ice_support":"true","fax_detect":"false","outbound_proxy":"","t38_udptl_maxdatagram":"0","direct_media_glare_mitigation":"none","dtls_rekey":"0","context":"default","media_encryption_optimistic":"false","named_pickup_group":"","from_domain":"","mailboxes":"","sdp_session":"Asterisk","cos_video":"0","identify_by":"username","t38_udptl":"false","send_rpid":"false","rtp_engine":"asterisk","t38_udptl_ec":"none","dtls_verify":"No","aggregate_mwi":"true","moh_suggest":"default","media_encryption":"no","callerid":"<unknown>","named_call_group":"","record_on_feature":"automixmon","dtls_setup":"active","inband_progress":"false","timers_min_se":"90","tos_video":"0","rtp_symmetric":"true","rtp_ipv6":"false","transport":"","t38_udptl_nat":"false","connected_line_method":"invite","allow_transfer":"true","allow_subscribe":"true","srtp_tag_32":"false","g726_non_standard":"false","100rel":"yes","use_avpf":"false","call_group":"","moh_passthrough":"false","user_eq_phone":"false","allow":"(g722|ulaw|alaw)","sub_min_expiry":"0","force_rport":"true","direct_media":"true","dtmf_mode":"rfc4733","media_use_received_transport":"false","record_off_feature":"automixmon","language":"","mwi_from_user":"","one_touch_recording":"false","rewrite_contact":"true","cos_audio":"0","t38_udptl_ipv6":"false","trust_id_outbound":"false","tone_zone":"","auth":"alice","from_user":"alice","disable_direct_media_on_nat":"false","tos_audio":"0","use_ptime":"false","media_address":"","timers":"yes","send_pai":"false","callerid_privacy":"allowed_not_screened"}
3 results found.
*CLI>
```

## Deleting Configuration

If we no longer want Alice to have an endpoint, we can remove it and its related objects using the DELETE operation:

```
$ curl -X DELETE -u asterisk:secret
https://localhost:8088/ari/asterisk/config/dynamic/res_pjsip/endpoint/alice
$ curl -X DELETE -u asterisk:secret
https://localhost:8088/ari/asterisk/config/dynamic/res_pjsip/aor/alice
$ curl -X DELETE -u asterisk:secret
https://localhost:8088/ari/asterisk/config/dynamic/res_pjsip/auth/alice
```

And we can confirm that Alice no longer exists:

## Asterisk CLI

```
*CLI> pjsip show endpoints
No objects found.
*CLI> database show
0 results found.
*CLI>
```

**Order Matters!**

Note that we remove Alice in reverse order of how her endpoint was created - we first remove the endpoint, then its related objects. Once the endpoint is removed, further requests will no longer be serviced, which allows us to safely remove the auth and aor objects.

## Back-end Database and Realtime Connectivity



Under Construction



Top-level page for configuration pages about ODBC modules, native database connectors and modules used for realtime.



## cURL



This page is under construction and may be incomplete or missing information in some areas. If you have questions, please wait until this notice is removed before asking, since it is possible your question will be answered by the time this page is completed.

Asterisk's ability to retrieve and store data to realtime backends is most commonly associated with relational databases. One of the lesser-known realtime backends available in Asterisk is `cURL`. Using this realtime backend makes Asterisk use HTTP GET and POST requests in order to retrieve data from and store data to an HTTP server.

- [Justification](#)
- [Dependencies and Installation](#)
  - [Troubleshooting](#)
- [Configuration](#)
- [Operations](#)
  - [multi](#)
  - [store](#)
  - [update](#)
  - [destroy](#)
  - [static](#)
  - [require](#)
- [Other Information](#)

### Justification

If Asterisk is capable of using a relational database as a store for realtime data, then what is the need for using HTTP? There are several potential reasons:

- Your setup hinges on a web service such as Django or something else, and you would prefer that Asterisk go through this service instead of skirting it to get directly at the data.
- You are forced to use a database that Asterisk does not have a native backend for and whose ODBC support is subpar.
- A relational database carries too much overhead

### Dependencies and Installation

Asterisk's realtime `cURL` backend is provided by the module `res_config_curl.so`. In order to build this module, you will first need to have the `libcurl` development library installed on your machine. If you wish to install the library from source, you can find it [here](#). If you would rather use your Linux distribution's package management, then you should be able to download the development libraries that way instead.

If you use a distribution with aptitude-based packaging (Debian, Ubuntu, Mint, et al), then use this command to install:

```
apt-get install libcurl4-openssl-dev
```

If you use a distribution with yum-based packaging (CentOS, RHEL, Fedora, et al), then use this command to install:

```
yum -y install libcurl-devel
```

Both of the above commands assume that you have permission to install the packages. You may need to prepend the command with "sudo" in order to be able to install the packages.

Once you have the `libcurl` development libraries installed, you need to run Asterisk's configure script in order for Asterisk to detect the installed library:

```
$ ./configure
```

In addition to the `libcurl` development library, `res_config_curl.so` relies on two other modules within Asterisk: `res_curl.so` and `func_curl.so`. `res_curl.so` initializes the `cURL` library within Asterisk. `func_curl.so` provides dialplan functions (`CURL` and `CURLOPT`) that are used directly by `res_config_curl.so`.

After running the configure script, run

```
$ make menuselect
```

to select which modules to build. Ensure that you can select `res_curl` and `res_config_curl` from the "Resource Modules" menu and that you can select `func_curl` from the "Dialplan Functions" menu. Once you have ensured that these have been selected, save your changes ('x' key if using curses-based `menuselect` or select the "Save & Exit" option if using newt-based or gtk-based `menuselect`). After, you just need to run

```
$ make && make install
```

in order to build Asterisk and install it on the system. You may need to prepend "sudo" to the "make install" command if there are permission problems when attempting to install. Once you have installed Asterisk, you can test that `res_config_curl.so` has been installed properly by starting Asterisk:

```
$ asterisk -c
```

Once Asterisk has started, type the following on the CLI:

```
*CLI> module show like res_config_curl
Module           Description           Use Count  Status
res_config_curl.so  Realtime Curl configuration  0          Running
1 modules loaded
```

The output when you run the command should look like what is shown above. If it does, then Asterisk is capable of using cURL for realtime.

## Troubleshooting

If you encounter problems along the way, here are some tips to help you get back on track.

- If the required modules in Asterisk are unselectable when you run `make menuselect`, then Asterisk did not detect the `libcurl` development library on your machine. If you installed the `libcurl` development library in a nonstandard place, then when running Asterisk's configure script, specify `--with-curl=/path/to/library` so that Asterisk can know where to look.
- If you built the required Asterisk modules but the `res_config_curl.so` module is not properly loaded, then check your `modules.conf` file to ensure that the necessary modules are being loaded. If you are noloading any of the required modules, then `res_config_curl.so` will not be able to load. If you are loading modules individually, be sure to list `res_curl.so` and `func_curl.so` before `res_config_curl.so` in your configuration.

## Configuration

Unlike other realtime backends, Asterisk does not have a specific configuration file for the realtime cURL backend. Instead, Asterisk gets the information it needs by reading the `extconfig.conf` file that it typically uses for general static and dynamic realtime configuration. The name of the realtime engine that Asterisk uses for cURL is called "curl" in `extconfig.conf`. Here is a sample:

```
[settings]
voicemail = curl,http://myserver.com:8000/voicemail
sippeers = curl,http://myserver.com:8000/sippeers
queues = curl,http://myserver.com:8000/my_queues
```

The basic syntax when using cURL is:

```
realtime_data = curl,<HTTP URL>
```

There are no hard-and-fast rules on what URL you place here. In the above sample, each of the various realtime stores correspond to resources on the same HTTP server. However, it would be perfectly valid to specify completely different servers for different realtime stores. Notice also that there is no requirement for the name of the realtime store to appear in the HTTP URL. In the above example the "queues" realtime store maps to the resource "my\_queues" on the HTTP server.

## Operations

The way Asterisk performs operations on your data is to send HTTP requests to different resources on your HTTP server. For instance, let's say that, based on your `extconfig.conf` file, you have mapped the "queues" realtime store to `http://myserver.com:8000/queues`. Asterisk will append whatever realtime operation it wishes to perform as a resource onto the end of the URL that you have provided. If Asterisk wanted to perform the "single" realtime operation, then Asterisk would send an HTTP request to `http://myserver.com:8000/queues/single`.

If your server is able to provide a response, then your server should return that response as the body of a 200-class HTTP response. If the request is unservable, then an appropriate HTTP error code should be sent.

The operations, as well as what is expected in response, are defined below.

For the first five examples, we will be using external MWI as the sample realtime store that Asterisk will be interacting with. The realtime MWI store stores the following data for each object

- `id`: The name of the mailbox for which MWI is being provided
- `msgs_new`: The number of new messages the mailbox currently has
- `msgs_old`: The number of old messages the mailbox currently has

We will operate with the assumption that the following two objects exist in the realtime store:

- -
  - `id`: "Dazed"
  - `msgs_new`: 5
  - `msgs_old`: 4
- -

- id: "Confused"
- msgs\_new: 6
- msgs\_old: 8

Our `extconfig.conf` file looks like this:

```
[settings]
mailboxes => curl,http://myserver.com:8000/mwi
```

single

The "single" resource is used for Asterisk to retrieve a single object from realtime.

Asterisk sends an HTTP POST request, using the body to indicate what data it wants. Here is an example of such a request:

```
POST /mwi/single HTTP/1.1
User-Agent: asterisk-libcurl-agent/1.0
Host: localhost:8000
Accept: */*
Content-Length: 8
Content-Type: application/x-www-form-urlencoded

id=Dazed
```

In this case, the request from Asterisk wants a single object whose id is "Dazed". Given the data we have stored, we would respond like so:

```
HTTP/1.1 200 OK
Date: Sat, 15 Mar 2014 18:23:21 GMT
Content-Length: 30
Content-Type: text/html

msgs_new=5&msgs_old=4&id=Dazed
```

The parameters describing the requested mailbox are returned on a single line in the HTTP response body. The order that the parameters are listed in is irrelevant.

If a "single" query from Asterisk matches more than one entity, you may choose to either respond with an HTTP error or simply return one of the matching records.

### **multi**

The "multi" resource is used to retrieve multiple objects from the realtime store.

Asterisk sends an HTTP POST request, using the body to indicate what data it wants. Here is an example of such a request:

```
POST /mwi/multi HTTP/1.1
User-Agent: asterisk-libcurl-agent/1.0
Host: localhost:8000
Accept: */*
Content-Length: 13
Content-Type: application/x-www-form-urlencoded

id%20LIKE=%25
```

The "multi" resource is one where Asterisk shows a weakness when not dealing with a relational database as its realtime backend. In this case, Asterisk has requested multiple rows with "id LIKE=%". What this means is that Asterisk wants to retrieve every object from the particular realtime store with an id equal to anything. Other queries Asterisk may send may be more like "foo LIKE=%bar%". In this case, Asterisk would be requesting all objects with a foo parameter that has "bar" as part of its value (so something with foo=barbara would match the query).

For this particular request, we would respond with the following:

```
HTTP/1.1 200 OK
Date: Sat, 15 Mar 2014 18:40:58 GMT
Content-Length: 65
Content-Type: text/html

msgs_new=5&msgs_old=4&id=Dazed
msgs_new=6&msgs_old=8&id=Confused
```

Each returned object is on its own line of the response.

### **store**

The "store" resource is used to save an object into the realtime store.

Asterisk sends an HTTP POST request, using the body to indicate what new object to store. Here is an example of such a request:

```
POST /mwi/store HTTP/1.1
User-Agent: asterisk-libcurl-agent/1.0
Host: localhost:8000
Accept: */*
Content-Length: 30
Content-Type: application/x-www-form-urlencoded

id=Shocked&msgs_old=5&msgs_new=7
```

In this case, Asterisk is attempting to store a new object with id "Shocked", 5 old messages and 7 new messages. Our realtime backend should reply with the number of objects stored.

```
HTTP/1.1 200 OK
Date: Sat, 15 Mar 2014 18:46:54 GMT
Content-Length: 1
Content-Type: text/html

1
```

Since we have stored one new object, we return "1" as our response.

If attempting to store an item that already exists in the database, you may either return an HTTP error or overwrite the old object with the new, depending on your policy.

### **update**

The "update" resource is used to change the values of parameters of objects in the realtime store.

Asterisk sends an HTTP POST request, using URL parameters to indicate what objects to update and using the body to indicate what values within those objects to update. Here is an example of such a request:

```
POST /mwi/update?id=Dazed HTTP/1.1
User-Agent: asterisk-libcurl-agent/1.0
Host: localhost:8000
Accept: */*
Content-Length: 24
Content-Type: application/x-www-form-urlencoded

msgs_old=25&msgs_new=300
```

In this case, the URL parameter "id=Dazed" tells us that Asterisk wants us to update all objects whose id is "Dazed". For any objects that match the criteria, we should update the number of old messages to 25 and the number of new messages to 300.

Our response indicates how many objects we updated. In this case, since we have updated one object, we respond with "1".

```
HTTP/1.1 200 OK
Date: Sat, 15 Mar 2014 18:52:26 GMT
Content-Length: 1
Content-Type: text/html

1
```

If there are no items that match the criteria, you may either respond with a "0" response body or return an HTTP error.

### **destroy**

The "destroy" resource is used to delete objects in the realtime store.

Asterisk sends an HTTP POST request, using the body to indicate what object to delete. Here is an example of such a request:

```
POST /mwi/destroy HTTP/1.1
User-Agent: asterisk-libcurl-agent/1.0
Host: localhost:8000
Accept: */*
Content-Length: 9
Content-Type: application/x-www-form-urlencoded

id=Dazed
```

In this case, Asterisk has requested that we delete the object with the id of "Dazed".

The body of our response indicates the number of items we deleted. Since we have deleted one object, we put "1" in our response body:

```
HTTP/1.1 200 OK
Date: Sat, 15 Mar 2014 18:57:23 GMT
Content-Length: 1
Content-Type: text/html

1
```

If asked to delete an object that does not exist, you may either respond with a "0" body or with an HTTP error.

## **static**

The "static" resource is used for static realtime requests.

Static realtime is a different realm from the more common dynamic realtime. Whereas dynamic realtime is restricted to certain configuration types that are designed to be used this way, static realtime uses a generic construct that can be substituted for any configuration file in Asterisk. The downside to static realtime is that Asterisk only ever interacts with the static realtime backend when the module that uses the configuration is reloaded. Internally, the Asterisk module thinks that it is reading its configuration from a configuration file, but under the hood, the configuration is actually retrieved from a realtime backend.

Static realtime "objects" are all the same, no matter what configuration file the static realtime store is standing in for. Object has been placed in quotation marks in that previous sentence because each static realtime object does not represent an entire configuration object, but rather represents a line in a configuration file. Here are the parameters for each static realtime object:

- id: A unique numerical id for the static realtime object
- filename: The configuration file that this static realtime object belongs to
- cat\_metric: Numerical id for a configuration category. Used by Asterisk to order categories for evaluation.
- category: Name of the configuration category
- var\_metric: Numerical id for a variable within a category. Used by Asterisk to order variables for evaluation.
- var\_name: Parameter name
- var\_val: Parameter value
- commented: If non-zero, indicates this object should be ignored

For our example, we will have the following objects stored in our static realtime store:

- -
  - id: 0
  - cat\_metric: 0
  - var\_metric: 0
  - filename: pjsip.conf
  - category: alice
  - var\_name: type
  - var\_val: endpoint
  - commented: 0
- -
  - id: 1
  - cat\_metric: 0
  - var\_metric: 1
  - filename: pjsip.conf
  - category: alice
  - var\_name: allow
  - var\_val: ulaw
  - commented: 0
- -
  - id: 2
  - cat\_metric: 0
  - var\_metric: 2
  - filename: pjsip.conf
  - category: alice
  - var\_name: context
  - var\_val: fabulous
  - commented: 0

This schema is identical to the `pjsip.conf` configuration file:

```
[alice]
type=endpoint
allow=ulaw
context=fabulous
```

Asterisk uses an HTTP GET to request static realtime data, using a URL parameter to indicate which filename it cares about. Here is an example of such a request:

```
GET /astconfig/static?file=pjsip.conf HTTP/1.1
User-Agent: asterisk-libcurl-agent/1.0
Host: localhost:8000
Accept: */*
```

In this case, Asterisk wants all static realtime objects whose filename is "pjsip.conf". Note that the HTTP request calls the parameter "file", whereas the actual name of the parameter returned from the realtime store is called "filename".

Our response contains all matching static realtime objects:

```
HTTP/1.1 200 OK
Date: Sat, 15 Mar 2014 19:13:41 GMT
Content-Length: 328
Content-Type: text/html

category=alice&commented=0&var_metric=0&var_name=type&var_val=endpoint&id=0&filename=pjsip.conf&cat_metric=0
category=alice&commented=0&var_metric=1&var_name=allow&var_val=ulaw&id=1&filename=pjsip.conf&cat_metric=0
category=alice&commented=0&var_metric=2&var_name=context&var_val=fabulous&id=2&filename=pjsip.conf&cat_metric=0
```

Unlike other realtime responses, the static realtime response needs to present the data in a particular order:

- First order: by descending `cat_metric`
- Second order: by ascending `var_metric`
- Third: lexicographically by category name
- Fourth: lexicographically by variable name

Note that Asterisk only pays attention to the "cat\_metric", "var\_metric", "category", "var\_name", and "var\_value" you return here, but you are free to return the entire object if you want. Note that Asterisk will not pay attention to the "commented" field, so be sure not to return any objects that have a non-zero "commented" value.

In summary, static realtime is cumbersome, confusing, and not worth it. Stay clear unless you just really need to use it.

If your realtime store does not provide objects for the specified file, then you may either return an empty body or an HTTP error.

## require

The "require" resource is used by Asterisk to test that a particular parameter for a realtime object is of a type it expects.

Asterisk sends an HTTP POST with body parameters describing what type it expects for a specific parameter. Here is an example of such a request:

```
POST /queue_members/require HTTP/1.1
User-Agent: asterisk-libcurl-agent/1.0
Host: localhost:8000
Accept: */*
Content-Length: 42
Content-Type: application/x-www-form-urlencoded

paused=integer1%3A1&uniqueid=uinteger2%3A5
```

Decoded, the body is "paused=integer1:1&uniqueid=uinteger2:5". The types that Asterisk can ask for are the following:

- An integer type can be indicated by any of the following
  - integer1
  - integer2
  - integer3
  - integer4
  - integer8
- An unsigned integer type can be indicated by any of the following
  - uinteger1
  - uinteger2
  - uinteger3
  - uinteger4
  - uinteger8
- char: A string
- date: A date
- datetime: A datetime
- float: a floating point number

It is undocumented what the meaning of the number after each of the "integer" and "uinteger" types means. If I'm guessing, it's the number of bytes allocated for the type.

The number after the colon for each parameter represents the minimum width, in digits for integer types and characters for char types, for each parameter.

In the example above, Asterisk is requiring that queue members' "paused" parameter be an integer type that can hold at least 1 digit and their "uniqueid" parameter be an unsigned integer type that can hold at least 5 digits.

Note that the purpose of Asterisk requesting the "require" resource is because Asterisk is going to attempt to **send** data of the type indicated to the realtime store. When receiving such a request, it is completely up to how you are storing your data to determine how to respond. If you are using a schema-less store for your data, then trying to test for width and type for each parameter is pointless, so you may as well just return successfully. If you are using something that has a schema you can check, then you should be sure that your realtime store can accommodate the data Asterisk will send. If your schema cannot accommodate the data, then this is an ideal time to modify the data schema if it is possible.

Respond with a "0" body to indicate success or a "-1" body to indicate failure. Here is an example response:

```
HTTP/1.1 200 OK
Date: Sat, 15 Mar 2014 18:57:23 GMT
Content-Length: 1
Content-Type: text/html

0
```

## Other Information

If you are interested in looking more in-depth into Asterisk's cURL realtime backend, you can find a reference example of an HTTP server in `contrib/scripts/dbsep.cgi` written by Tilghman Leshner, who also wrote Asterisk's realtime cURL support. This script works by converting the HTTP request from Asterisk into a SQL query for a relational database. The configuration for the database used in the `dbsep.cgi` script is detailed in the `configs/dbsep.conf.sample` file in the Asterisk source.

## ODBC



Under Construction



Top-level page for everything about configuring ODBC connectivity, res\_odbc, func\_odbc, etc



## Configuring res\_odbc

### Overview

The **res\_odbc** module for Asterisk can provide Asterisk with connectivity to various database backends through ODBC (a database abstraction layer). Asterisk features such as [Asterisk Realtime Architecture](#), [Call Detail Records](#), [Channel Event Logging](#), can connect to a database through res\_odbc.

More details on specific options within configuration are provided in the [sample configuration file](#) included with Asterisk source.

We'll provide a brief guide here on how to get the res\_odbc.so module configured to connect to an existing ODBC installation.

## Recompile Asterisk to build required modules

You'll need to rebuild Asterisk with the needed modules.

Other pages on the wiki describe that process:

[Building and Installing Asterisk](#)

[Using Menuselect to Select Asterisk Options](#)

When using menuselect, verify that the **func\_odbc** (you'll probably be using that one) and **res\_odbc** (required) modules will be built. Then, build Asterisk and make sure those modules were built and exist in **/usr/lib/asterisk/modules** (or whatever directory you use).

### Configure res\_odbc.conf to connect to your ODBC installation

Find the configuration file, which should typically be located at `/etc/asterisk/res_odbc.conf` and provide a basic configuration such as:

```
[asterisk]
enabled => yes
dsn => your-configured-dsn-name
username => your-database-username
password => insecurepassword
pre-connect => yes
```

Then start up Asterisk and assuming res\_odbc loads properly on the CLI you can use `odbc show` to verify a DSN is configured and shows up:

```
rnewton-office-lab*CLI> odbc show
ODBC DSN Settings
-----
Name:    asterisk
DSN:    your-configured-dsn-name
Last connection attempt: 1969-12-31 18:00:00
```

To verify the connection works you should use `func_odbc` or something similar to query the data source from Asterisk.

### Troubleshooting

If you don't have the **odbc** command at the CLI, check that

- The `res_odbc.so` module exists and has proper permissions in `/usr/lib/asterisk/modules/`
- Your `modules.conf` to make sure the module isn't `noloaded` or being prevented from loading somehow
- Debug during Asterisk startup to look for messages regarding `res_odbc.conf` (see `logger.conf` to get things setup)

If you the **odbc show** output shows "Connected: No" then you'll want to try connecting to your ODBC installation from other methods to verify it is working. The Linux tool **isql** is good for that.

## Getting Asterisk Connected to MySQL via ODBC

### Connect Asterisk to a MySQL back-end through ODBC

This is a short tutorial on how to quickly setup Asterisk to use MySQL, the ODBC MySQL connector and ODBC. We'll use CentOS 6 as the OS in this tutorial. However, the same essential steps apply for any popular Linux distro.

### Installing and Configuring MySQL

There are three basic steps to install and configure MySQL for Asterisk.

- Install MySQL server package and start the DB service.
- Secure the installation if appropriate.
- Configure a user and database for Asterisk in MySQL

### On This Page

- Connect Asterisk to a MySQL back-end through ODBC
  - Installing and Configuring MySQL
    - Install MySQL server package and start the DB service
    - Secure the installation if appropriate
    - Configure a user and database for Asterisk in MySQL
  - Install ODBC and the MySQL ODBC connector
    - Install the latest unixODBC and GNU Libtool Dynamic Module Loader packages
    - Install the latest MySQL ODBC connector
  - Configure ODBC and the MySQL ODBC connector
    - Configure odbcinst.ini for ODBC
    - Configure the MySQL ODBC connector
  - Test the ODBC Data Source Name connection
  - Configuring Asterisk to Use the New ODBC and MySQL Install
    - Getting the right Asterisk modules
    - Configuring Asterisk's ODBC connection

### Install MySQL server package and start the DB service

```
# sudo yum install mysql-server
# sudo service mysqld start
```

### Secure the installation if appropriate

If you intend to push this install into production or practice as if you were then you will want to use the `mysql_secure_installation` script to apply some basic security.

```
# sudo /usr/bin/mysql_secure_installation
```

### Configure a user and database for Asterisk in MySQL

If you want to use a GUI to manage your database then now is the time to set that GUI up and use it to create your asterisk user. Otherwise we will provide basic instructions for user setup below.

We'll have you login into the mysql command console, create a user, create a database and then assign appropriate permissions for the asterisk user. First, login using the root password you set earlier.

```
# mysql -u root -p
```

Now verify you are at the MySQL command prompt. It should look like "mysql>". Then enter the following commands:

```
# CREATE USER 'asterisk'@'%' IDENTIFIED BY 'replace_with_strong_password';
# CREATE DATABASE asterisk;
# GRANT ALL PRIVILEGES ON asterisk.* TO 'asterisk'@'%';
# exit
```

After each of the CREATE and GRANT commands you should see output indicating that the Query was OK including many rows were affected. If you want, you can test out the new permissions by logging in as your user to the asterisk database and then logout again.

```
# mysql -u asterisk -p asterisk
# exit
```

## Install ODBC and the MySQL ODBC connector

Be sure you have followed the previous sections as we presume you already have MySQL installed on your CentOS server along with a database and user for Asterisk configured. The database name should be 'asterisk' and the username should be 'asterisk'.

### Install the latest unixODBC and GNU Libtool Dynamic Module Loader packages

The development packages are necessary as well, since later Asterisk will need to use them when building ODBC related modules.

```
# sudo yum install unixODBC unixODBC-devel libtool-ltdl libtool-ltdl-devel
```

### Install the latest MySQL ODBC connector

```
# sudo yum install mysql-connector-odbc
```

## Configure ODBC and the MySQL ODBC connector

### Configure odbcinst.ini for ODBC

With recent UnixODBC versions the configuration should already be done for you in the /etc/odbcinst.ini file.

Verify that you have the following configuration:

```
# Driver from the mysql-connector-odbc package
# Setup from the unixODBC package
[MySQL]
Description      = ODBC for MySQL
Driver           = /usr/lib/libmyodbc5.so
Setup            = /usr/lib/libodbcmyS.so
Driver64         = /usr/lib64/libmyodbc5.so
Setup64          = /usr/lib64/libodbcmyS.so
FileUsage        = 1
Pooling          = Yes
CPOutput         = 120
```



We configure pooling for the connection as that is probably desired by most users. Consult the [unixODBC documentation](#) to learn more about the function of ODBC connection pooling.

There may also be configuration for PostgreSQL which you can comment out if you are not planning to setup PostgreSQL as well. Comments begin the line with a hash (#) symbol.

You can also call `odbcinst` to query the driver, verifying that the configuration is found.

```
# odbcinst -q -d
```

The output should read simply "[MySQL]"

### Configure the MySQL ODBC connector

Now we'll configure the /etc/odbc.ini file to create a DSN (Data Source Name) for Asterisk. The file may be empty, so you'll have to copy-paste from this example or write this from scratch.

Add the following to /etc/odbc.ini

```
[asterisk-connector]
Description = MySQL connection to 'asterisk' database
Driver = MySQL
Database = asterisk
Server = localhost
Port = 3306
Socket = /var/lib/mysql/mysql.sock
```



You may want to verify that `mysql.sock` is actually in the location specific here. It will differ on some systems depending on your configuration.

### Test the ODBC Data Source Name connection

Now is a good time to test your database by connecting to it and performing a query. The `unixODBC` package provides `isql`; a command line utility that allows you to connect to the Data Source, send SQL commands to it and receive results back. The syntax used is:

```
isql -v dsn_name db_username db_password
```

So, for our purposes you would enter:

```
# isql -v asterisk-connector asterisk replace_with_strong_password
```



It is important to use the `-v` flag so that if `isql` runs into a problem you will be alerted of any diagnostics or errors available.

At this point you should get an SQL prompt. Run the following command:

```
SQL> select 1
```

You should see some simple results if the query is successful. Then you can exit.

```
SQL> select 1
+-----+
| 1     |
+-----+
| 1     |
+-----+
SQLRowCount returns 1
1 rows fetched

SQL> quit
```

### Configuring Asterisk to Use the New ODBC and MySQL Install

Now you have a MySQL database, ODBC and an ODBC MySQL connector installed and basically configured. The next step is to recompile Asterisk so that the ODBC modules which required the previously mentioned items can now be built. Once those modules exist, then you can configure the proper configuration files in Asterisk depending on what information you want to write to or read from MySQL.

#### Getting the right Asterisk modules

If you already had Asterisk installed from source and the modules you need are already selected by default in `menuselect` - then the recompilation process could be as simple as navigating to the Asterisk source and running a few commands.

```
# cd ~/asterisk-source/
# ./configure
# make && make install
```

Otherwise you should follow the typical Asterisk installation process to make sure modules such as `res_odbc`, `res_config_odbc`, `cdr_odbc`, `cdr_adaptive_odbc` and `func_odbc` have their dependencies fulfilled and that they will be built.

See [Building and Installing Asterisk](#) and [Using Menuselect to Select Asterisk Options](#).

#### Configuring Asterisk's ODBC connection

The basic configuration for an Asterisk ODBC connection is handled in `res_odbc.conf`. You should check out the [Configuring res\\_odbc](#) page and follow it using the DSN and database username and password you setup earlier.

After you have the connection set up in Asterisk you are ready to then configure your database tables with the proper schema depending on what exactly you want to do with them. Asterisk comes with some helpful tools to do this, such as `Alembic`. See the [Managing Realtime Databases with Alembic](#) section to get started with `Alembic` if you are working towards an Asterisk Realtime setup.

## SQLite Tables

```
/*
 * res_config_sqlite - SQLite 2 support for Asterisk
 *
 * This module can be used as a static/RealTime configuration module, and a CDR
 * handler. See the Doxygen documentation for a detailed description of the
 * module, and the configs/ directory for the sample configuration file.
 */

/*
 * Tables for res_config_sqlite.so.
 */

/*
 * RealTime static table.
 */
CREATE TABLE ast_config (
  id INTEGER,
  cat_metric INT(11) NOT NULL DEFAULT 0,
  var_metric INT(11) NOT NULL DEFAULT 0,
  commented TINYINT(1) NOT NULL DEFAULT 0,
  filename VARCHAR(128) NOT NULL DEFAULT '',
  category VARCHAR(128) NOT NULL DEFAULT 'default',
  var_name VARCHAR(128) NOT NULL DEFAULT '',
  var_val TEXT NOT NULL DEFAULT '',
  PRIMARY KEY (id)
);

CREATE INDEX ast_config_idx_cat_metric ON ast_config(cat_metric);
CREATE INDEX ast_config_idx_var_metric ON ast_config(var_metric);
CREATE INDEX ast_config_idx_filename_commented ON ast_config(filename, commented);

/*
 * CDR table (this table is automatically created if non existent).
 */
CREATE TABLE ast_cdr (
  id INTEGER,
  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 '',
  start DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
  answer DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
  end DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
  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 '',
  uniqueid VARCHAR(32) NOT NULL DEFAULT '',
  userfield VARCHAR(255) NOT NULL DEFAULT '',
  PRIMARY KEY (id)
);

/*
 * SIP RealTime table.
 */
CREATE TABLE ast_sip (
  id INTEGER,
  commented TINYINT(1) NOT NULL DEFAULT 0,
  name VARCHAR(80) NOT NULL DEFAULT '',
  host VARCHAR(31) NOT NULL DEFAULT '',
  nat VARCHAR(5) NOT NULL DEFAULT 'no',
  type VARCHAR(6) NOT NULL DEFAULT 'friend',
  accountcode VARCHAR(20) DEFAULT NULL,
  amaflags VARCHAR(13) DEFAULT NULL,
  callgroup VARCHAR(10) DEFAULT NULL,
  callerid VARCHAR(80) DEFAULT NULL,
  cancelforward CHAR(3) DEFAULT 'yes',
  directmedia CHAR(3) DEFAULT 'yes',
  context VARCHAR(80) DEFAULT NULL,
  defaulttip VARCHAR(15) DEFAULT NULL,
  dtmfmode VARCHAR(7) DEFAULT NULL,
  fromuser VARCHAR(80) DEFAULT NULL,
  fromdomain VARCHAR(80) DEFAULT NULL,
  insecure VARCHAR(4) DEFAULT NULL,
  language CHAR(2) DEFAULT NULL,
  mailbox VARCHAR(50) DEFAULT NULL,
  md5secret VARCHAR(80) DEFAULT NULL,
  deny VARCHAR(95) DEFAULT NULL,
  permit VARCHAR(95) DEFAULT NULL,
  mask VARCHAR(95) DEFAULT NULL,
```

```

musiconhold VARCHAR(100) DEFAULT NULL,
pickupgroup VARCHAR(10) DEFAULT NULL,
qualify CHAR(3) DEFAULT NULL,
regexten VARCHAR(80) DEFAULT NULL,
restrictoid CHAR(3) DEFAULT NULL,
rtptimeout CHAR(3) DEFAULT NULL,
rtpholdtimeout CHAR(3) DEFAULT NULL,
secret VARCHAR(80) DEFAULT NULL,
setvar VARCHAR(100) DEFAULT NULL,
disallow VARCHAR(100) DEFAULT 'all',
allow VARCHAR(100) DEFAULT 'g729,ilbc,gsm,ulaw,alaw',
fullcontact VARCHAR(80) NOT NULL DEFAULT '',
ipaddr VARCHAR(15) NOT NULL DEFAULT '',
port INT(11) NOT NULL DEFAULT 0,
regserver VARCHAR(100) DEFAULT NULL,
regseconds INT(11) NOT NULL DEFAULT 0,
username VARCHAR(80) NOT NULL DEFAULT '',
PRIMARY KEY (id)
UNIQUE (name)
);

CREATE INDEX ast_sip_idx_commented ON ast_sip(commented);

/*
 * Dialplan RealTime table.
 */
CREATE TABLE ast_exten (
id INTEGER,
commented TINYINT(1) NOT NULL DEFAULT 0,
context VARCHAR(80) NOT NULL DEFAULT '',
exten VARCHAR(40) NOT NULL DEFAULT '',
priority INT(11) NOT NULL DEFAULT 0,
app VARCHAR(128) NOT NULL DEFAULT '',
appdata VARCHAR(128) NOT NULL DEFAULT '',
PRIMARY KEY (id)
);

```

```
CREATE INDEX ast_exten_idx_commented ON ast_exten(commented);  
CREATE INDEX ast_exten_idx_context_exten_priority ON ast_exten(context, exten, priority);
```

# Storing Voicemail in PostgreSQL via ODBC

## How to get ODBC storage with PostgreSQL working with Voicemail

\*Install PostgreSQL, PostgreSQL-devel, unixODBC, and unixODBC-devel, and PostgreSQL-ODBC. Make sure PostgreSQL is running and listening on a TCP socket.

\*Log into your server as root, and then type:

```
[root@localhost ~]# su - postgres
```

This will log you into the system as the "postgres" user, so that you can create a new role and database within the PostgreSQL database system. At the new prompt, type:

```
$ createuser -s -D -R -l -P -e asterisk
Enter password for new role:
Enter it again:
```

Obviously you should enter a password when prompted. This creates the database role (or user).

Next we need to create the asterisk database. Type:

```
$ createdb -O asterisk -e asterisk
```

This creates the database and sets the owner of the database to the asterisk role.

Next, make sure that you are using md5 authentication for the database user. The line in my `/var/lib/pgsql/data/pg_hba.conf` looks like:

```
# "local" is for Unix domain socket connections only
local asterisk asterisk md5
local all all ident sameuser
# IPv4 local connections:
host all all 127.0.0.1/32 md5
```

As soon as you're done editing that file, log out as the postgres user.

- Make sure you have the PostgreSQL odbc driver setup in `/etc/odbcinst.ini`. Mine looks like:

```
[PostgreSQL]
Description = ODBC for PostgreSQL
Driver      = /usr/lib/libodbcpsql.so
Setup      = /usr/lib/libodbcpsqlS.so
FileUsage   = 1
```

You can confirm that unixODBC is seeing the driver by typing:

```
[jsmith2@localhost tmp]$ odbcinst -q -d
[PostgreSQL]
```

- Set up a DSN in `/etc/odbc.ini`, pointing at the PostgreSQL database and driver. Mine looks like:



```
[testing]
Description          = ODBC Testing
Driver               = PostgreSQL
Trace                = No
TraceFile            = sql.log
Database             = asterisk
Servername           = 127.0.0.1
Username             = asterisk
Password             = supersecret
Port                 = 5432
ReadOnly              = No
RowVersioning        = No
ShowSystemTables     = No
ShowOidColumn        = No
FakeOidIndex         = No
ConnSettings         =
```

You can confirm that unixODBC sees your DSN by typing:

```
[jsmith2@localhost tmp]$ odbcinst -q -s
[testing]
```

- Test your database connectivity through ODBC. If this doesn't work, something is wrong with your ODBC setup.

```
[jsmith2@localhost tmp]$ echo "select 1" | isql -v testing
+-----+
| Connected! |
|          |
| sql-statement |
| help [tablename] |
| quit |
+-----+
SQL> +-----+
| ?column? |
+-----+
| 1 |
+-----+
SQLRowCount returns 1
1 rows fetched
```

If your ODBC connectivity to PostgreSQL isn't working, you'll see an error message instead, like this:

```
[jsmith2@localhost tmp]$ echo "select 1" | isql -v testing
[S1000][unixODBC]Could not connect to the server;
Could not connect to remote socket.
[ISQL]ERROR: Could not SQLConnect
bash: echo: write error: Broken pipe
```

- Compile Asterisk with support for ODBC voicemail. Go to your Asterisk source directory and run `make menuselect`. Under "Voicemail Build Options", enable "ODBC\_STORAGE". See doc/README.odbcstorage for more information

Recompile Asterisk and install the new version.

- Once you've recompiled and re-installed Asterisk, check to make sure res\_odbc.so has been compiled.

```
localhost*CLI> show modules like res_odbc.so
Module          Description          Use Count
res_odbc.so     ODBC Resource          0
1 modules loaded
```

- Now it's time to get Asterisk configured. First, we need to tell Asterisk about our ODBC setup. Open /etc/asterisk/res\_odbc.conf and add the following:

```
[postgres]
enabled => yes
dsn => testing
pre-connect => yes
```

- At the Asterisk CLI, unload and then load the `res_odbc.so` module. (You could restart Asterisk as well, but this way makes it easier to tell what's happening.) Notice how it says it's connected to "postgres", which is our ODBC connection as defined in `res_odbc.conf`, which points to the "testing" DSN in ODBC.

```
localhost*CLI> unload res_odbc.so
Jan  2 21:19:36 WARNING[8130]: res_odbc.c:498 odbc_obj_disconnect: res_odbc: disconnected 0 from postgres [testing]
Jan  2 21:19:36 NOTICE[8130]: res_odbc.c:589 unload_module: res_odbc unloaded.
localhost*CLI> load res_odbc.so
Loaded /usr/lib/asterisk/modules/res_odbc.so => (ODBC Resource)
== Parsing '/etc/asterisk/res_odbc.conf': Found
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:266 load_odbc_config: Adding ENV var: INFORMIXSERVER=my_special_database
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:266 load_odbc_config: Adding ENV var: INFORMIXDIR=/opt/informix
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:295 load_odbc_config: registered database handle 'postgres' dsn->[testing]
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:555 odbc_obj_connect: Connecting postgres
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:570 odbc_obj_connect: res_odbc: Connected to postgres [testing]
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:600 load_module: res_odbc loaded.
```

You can also check the status of your ODBC connection at any time from the Asterisk CLI:

```
localhost*CLI> odbc show
Name: postgres
DSN: testing
Connected: yes
```

- Now we can setup our voicemail table in PostgreSQL. Log into PostgreSQL and type (or copy and paste) the following:

```

--
-- First, let's create our large object type, called "lo"
--
CREATE FUNCTION loin (cstring) RETURNS lo AS 'oidin' LANGUAGE internal IMMUTABLE STRICT;
CREATE FUNCTION loout (lo) RETURNS cstring AS 'oidout' LANGUAGE internal IMMUTABLE STRICT;
CREATE FUNCTION lorecv (internal) RETURNS lo AS 'oidrecv' LANGUAGE internal IMMUTABLE STRICT;
CREATE FUNCTION losend (lo) RETURNS bytea AS 'oidrecv' LANGUAGE internal IMMUTABLE STRICT;

CREATE TYPE lo ( INPUT = loin, OUTPUT = loout, RECEIVE = lorecv, SEND = losend, INTERNALLENGTH = 4, PASSEDBYVALUE );
CREATE CAST (lo AS oid) WITHOUT FUNCTION AS IMPLICIT;
CREATE CAST (oid AS lo) WITHOUT FUNCTION AS IMPLICIT;

--
-- If we're not already using plpgsql, then let's use it!
--
CREATE TRUSTED LANGUAGE plpgsql;

--
-- Next, let's create a trigger to cleanup the large object table
-- whenever we update or delete a row from the voicemail messages table
--
CREATE FUNCTION vm_lo_cleanup() RETURNS "trigger"
AS $$
declare
    msgcount INTEGER;
begin
    -- raise notice 'Starting lo_cleanup function for large object with oid %',old.recording;
    -- If it is an update action but the BLOB (lo) field was not changed, dont do anything
    if (TG_OP = 'UPDATE') then
        if ((old.recording = new.recording) or (old.recording is NULL)) then
            raise notice 'Not cleaning up the large object table, as recording has not changed';
            return new;
        end if;
    end if;
    if (old.recording IS NOT NULL) then
        SELECT INTO msgcount COUNT(*) AS COUNT FROM voicemessages WHERE recording = old.recording;
        if (msgcount > 0) then
            raise notice 'Not deleting record from the large object table, as object is still referenced';
            return new;
        else
            perform lo_unlink(old.recording);
            if found then
                raise notice 'Cleaning up the large object table';
                return new;
            else
                raise exception 'Failed to cleanup the large object table';
                return old;
            end if;
        end if;
    else
        raise notice 'No need to cleanup the large object table, no recording on old row';
        return new;
    end if;
end$$
LANGUAGE plpgsql;

--
-- Now, let's create our voicemessages table
-- This is what holds the voicemail from Asterisk
--
CREATE TABLE voicemessages
(
    uniqueid serial PRIMARY KEY,
    msgnum int4,
    dir varchar(80),
    context varchar(80),
    macrocontext varchar(80),
    callerid varchar(40),
    origtime varchar(40),
    duration varchar(20),
    flag varchar(8),
    mailboxuser varchar(80),
    mailboxcontext varchar(80),
    recording lo,
    label varchar(30),
    "read" bool DEFAULT false
);

--
-- Let's not forget to make the voicemessages table use the trigger
--
CREATE TRIGGER vm_cleanup AFTER DELETE OR UPDATE ON voicemessages FOR EACH ROW EXECUTE PROCEDURE vm_lo_cleanup();

```

- Just as a sanity check, make sure you check the voicemail messages table via the isql utility.

```
[jsmith2@localhost ODBC]$ echo "SELECT uniqueid, msgnum, dir, duration FROM voicemailmessages WHERE msgnum = 1" | isql testing
+-----+
| Connected! |
+-----+
| sql-statement |
| help [tablename] |
| quit |
+-----+
SQL>
+-----+
--+
| uniqueid | msgnum | dir | duration |
+-----+
--+
+-----+
--+
SQLRowCount returns 0
```

- Now we can finally configure voicemail in Asterisk to use our database. Open /etc/asterisk/voicemail.conf, and look in the [general] section. I've changed the format to gsm (as I can't seem to get WAV or wav working), and specify both the odbc connection and database table to use.

```
[general]
; Default formats for writing Voicemail
;format=g723sf|wav49|wav
format=gsm
odbcstorage=postgres
odbcstorage=voicemailmessages
```

You'll also want to create a new voicemail context called "odbctest" to do some testing, and create a sample mailbox inside that context. Add the following to the very bottom of voicemail.conf:

```
[odbctest]
101 => 5555,Example Mailbox
```

- Once you've updated voicemail.conf, let's make the changes take effect:

```
localhost*CLI> unload app_voicemail.so
== Unregistered application 'VoiceMail'
== Unregistered application 'VoiceMailMain'
== Unregistered application 'MailboxExists'
== Unregistered application 'VMAuthenticate'
localhost*CLI> load app_voicemail.so
Loaded /usr/lib/asterisk/modules/app_voicemail.so => (Comedian Mail (Voicemail System))
== Registered application 'VoiceMail'
== Registered application 'VoiceMailMain'
== Registered application 'MailboxExists'
== Registered application 'VMAuthenticate'
== Parsing '/etc/asterisk/voicemail.conf': Found
```

You can check to make sure your new mailbox exists by typing:

```
localhost*CLI> show voicemail users for odbctest
Context Mbox User Zone NewMsg
odbctest 101 Example Mailbox 0
```

- Now, let's add a new context called "odbc" to extensions.conf. We'll use these extensions to do some testing:

```
[odbc]
exten => 100,1,Voicemail(101@odbctest)
exten => 200,1,VoicemailMain(101@odbctest)
```

- Next, we need to point a phone at the odbc context. In my case, I've got a SIP phone called "linksys" that is registering to Asterisk, so I'm setting its context to the [odbc] context we created in the previous step. The relevant section of my sip.conf file looks like:



```
[jsmith2@localhost tmp]$ play /tmp/odcb-16599.gsm

Input Filename : /tmp/odcb-16599.gsm
Sample Size    : 8-bits
Sample Encoding: gsm
Channels       : 1
Sample Rate    : 8000

Time: 00:06.22 [00:00.00] of 00:00.00 ( 0.0%) Output Buffer: 298.36K

Done.
```

- Last but not least, we can pull the voicemail message back out of the database by dialing extension 200 and entering "5555" at the password prompt. You should see something like this on the Asterisk CLI:

```
localhost*CLI>
-- Executing VoiceMailMain("SIP/linksys-10228cac", "101@odbctest") in new stack
-- Playing 'vm-password' (language 'en')
-- Playing 'vm-youhave' (language 'en')
-- Playing 'digits/1' (language 'en')
-- Playing 'vm-INBOX' (language 'en')
-- Playing 'vm-message' (language 'en')
-- Playing 'vm-onefor' (language 'en')
-- Playing 'vm-INBOX' (language 'en')
-- Playing 'vm-messages' (language 'en')
-- Playing 'vm-opts' (language 'en')
-- Playing 'vm-first' (language 'en')
-- Playing 'vm-message' (language 'en')
== Parsing '/var/spool/asterisk/voicemail/odbctest/101/INBOX/msg0000.txt': Found
-- Playing 'vm-received' (language 'en')
-- Playing 'digits/at' (language 'en')
-- Playing 'digits/10' (language 'en')
-- Playing 'digits/16' (language 'en')
-- Playing 'digits/p-m' (language 'en')
-- Playing '/var/spool/asterisk/voicemail/odbctest/101/INBOX/msg0000' (language 'en')
-- Playing 'vm-advopts' (language 'en')
-- Playing 'vm-repeat' (language 'en')
-- Playing 'vm-delete' (language 'en')
-- Playing 'vm-toforward' (language 'en')
-- Playing 'vm-savemessage' (language 'en')
-- Playing 'vm-helpexit' (language 'en')
-- Playing 'vm-goodbye' (language 'en')
```

That's it!

Jared Smith  
2 Jan 2006  
(updated 11 Mar 2007)

## Distributed Device State

In the Key Concepts section [States and Presence](#) we discussed Asterisk [Device State](#), [Extension State](#) and [Hints](#).

The child pages here discuss the configuration of modules that allow device state to be distributed between multiple systems or instances of Asterisk.

### In This Section

- [Corosync](#)
- [Distributed Device State with AIS](#)
- [Distributed Device State with XMPP PubSub](#)

# Corosync

## Corosync

Corosync is an open source group messaging system typically used in clusters, cloud computing, and other high availability environments.

The project, at its core, provides four C api features:

- A closed process group communication model with virtual synchrony guarantees for creating replicated state machines.
- A simple availability manager that restarts the application process when it has failed.
- A configuration and statistics in-memory database that provide the ability to set, retrieve, and receive change notifications of information.
- A quorum system that notifies applications when quorum is achieved or lost.

## Corosync and Asterisk

Using Corosync together with res\_corosync allows events to be shared amongst a local cluster of Asterisk servers. Specifically, the types of events that may be shared include:

- Device state
- Message Waiting Indication, or MWI (to allow voicemail to live on a server that is different from where the phones are registered)

## Setup and Configuration

Corosync

- Installation Debian / Ubuntu

```
apt-get install corosync corosync-dev
```

Red Hat / Fedora

```
yum install corosync corosynclib corosynclib-devel
```

- Authkey To create an authentication key for secure communications between your nodes you need to do this on, what will be, the active node.

```
corosync-keygen
```

This creates a key in /etc/corosync/authkey.

```
asterisk_active:~# scp /etc/corosync/authkey asterisk_standby:
```

Now, on the standby node, you'll need to stick the authkey in it's new home and fix it's permissions / ownership.

```
asterisk_standby:~# mv ~/authkey /etc/corosync/authkey
asterisk_standby:~# chown root:root /etc/corosync/authkey
asterisk_standby:~# chmod 400 /etc/corosync/authkey
```

- /etc/corosync/corosync.conf The interface section under the totem block defines the communication path(s) to the other Corosync processes running on nodes within the cluster. These can be either IPv4 or IPv6 ip addresses but can not be mixed and matched within an interface. Adjustments can be made to the cluster settings based on your needs and installation environment.
  - IPv4 Active Node Example

```
totem {
    version: 2
    token: 160
    token_retransmits_before_loss_const: 3
    join: 30
    consensus: 300
    vsftype: none
    max_messages: 20
    threads: 0
    nodeid: 1
    rrp_mode: none
    interface {
        ringnumber: 0
        bindnetaddr: 192.168.1.0
        mcastaddr: 226.94.1.1
        mcastport: 5405
    }
}
```

Standby Node Example



```

totem {
    version: 2
    token: 160
    token_retransmits_before_loss_const: 3
    join: 30
    consensus: 300
    vsftype: none
    max_messages: 20
    threads: 0
    nodeid: 2
    rrp_mode: none
    interface {
        ringnumber: 0
        bindnetaddr: 192.168.1.0
        mcastaddr: 226.94.1.1
        mcastport: 5405
    }
}

```

- Start Corosync

```
service corosync start
```

#### Asterisk

- Installation In your Asterisk source directory:

```
./configure
make
make install
```

- /etc/asterisk/res\_corosync.conf

```

;
; Sample configuration file for res_corosync.
;
; This module allows events to be shared amongst a local cluster of
; Asterisk servers.  Specifically, the types of events that may be
; shared include:
;
; - Device State (for shared presence information)
;
; - Message Waiting Indication, or MWI (to allow Voicemail to live on
;   a server that is different from where the phones are registered)
;
; For more information about Corosync, see: http://www.corosync.org/
;

[general]
;
; Publish Message Waiting Indication (MWI) events from this server to the
; cluster.
publish_event = mwi
;
; Subscribe to MWI events from the cluster.
subscribe_event = mwi
;
; Publish Device State (presence) events from this server to the cluster.
publish_event = device_state
;
; Subscribe to Device State (presence) events from the cluster.
subscribe_event = device_state
;

```

In the general section of the res\_corosync.conf file we are specifying which events we'd like to publish and subscribe to (at the moment this is either device\_state or mwi).

- Verifying Installation If everything is setup correctly, you should see this output after executing a 'corosync show members' on the Asterisk CLI.

```
*CLI> corosync show members

=====
=== Cluster members =====
=====
===
=== Node 1
=== --> Group: asterisk
=== --> Address 1: <host #1 ip goes here>
===
=====
```

After starting Corosync and Asterisk on your second node, the 'corosync show members' output should look something like this:

```
*CLI> corosync show members

=====
=== Cluster members =====
=====
===
=== Node 1
=== --> Group: asterisk
=== --> Address 1: <host #1 ip goes here>
=== Node 2
=== --> Group: asterisk
=== --> Address 1: <host #2 ip goes here>
===
=====
```

# Distributed Device State with AIS

- 1. Introduction
- 2. OpenAIS Installation
  - Description
  - Install Dependencies
  - Download
  - Compile and Install
- 3. OpenAIS Configuration
- 4. Running OpenAIS
- 5. Installing Asterisk
- 6. Configuring Asterisk
- 7. Basic Testing of Asterisk with OpenAIS
- 8. Testing Distributed Device State

## 1. Introduction

Various changes have been made related to "event handling" in Asterisk. One of the most important things included in these changes is the ability to share certain events between servers. The two types of events that can currently be shared between servers are:

1. **MWI - Message Waiting Indication** - This gives you a high performance option for letting servers in a cluster be aware of changes in the state of a mailbox. Instead of having each server have to poll an ODBC database, this lets the server that actually made the change to the mailbox generate an event which will get distributed to the other servers that have subscribed to this information.
2. **Device State** - This lets servers in a local cluster inform each other about changes in the state of a device on that particular server. When the state of a device changes on any server, the overall state of that device across the cluster will get recalculated. So, any subscriptions to the state of a device, such as hints in the dialplan or an application like Queue() which reads device state, will then reflect the state of a device across a cluster.

## 2. OpenAIS Installation

### Description

The current solution for providing distributed events with Asterisk is done by using the AIS (Application Interface Specification), which provides an API for a distributed event service. While this API is standardized, this code has been developed exclusively against the open source implementation of AIS called OpenAIS.

For more information about OpenAIS, visit their web site <http://www.openais.org/>.

### Install Dependencies

- Ubuntu
  - libnss3-dev
- Fedora
  - nss-devel

### Download

Download the latest versions of Corosync and OpenAIS from <http://www.corosync.org/> and <http://www.openais.org/>.

### Compile and Install

```
$ tar xvzf corosync-1.2.8.tar.gz
$ cd corosync-1.2.8
$ ./configure
$ make
$ sudo make install
```

```
$ tar xvzf openais-1.1.4.tar.gz
$ cd openais-1.1.4
$ ./configure
$ make
$ sudo make install
```

## 3. OpenAIS Configuration

Basic OpenAIS configuration to get this working is actually pretty easy. Start by copying in a sample configuration file for Corosync and OpenAIS.

```
$ sudo mkdir -p /etc/ais
$ cd openais-1.1.4
$ sudo cp conf/openais.conf.sample /etc/ais/openais.conf
```

```
$ sudo mkdir -p /etc/corosync
$ cd corosync-1.2.8
$ sudo cp conf/corosync.conf.sample /etc/corosync/corosync.conf
```

Now, edit `openais.conf` using the editor of your choice.

```
$ ${EDITOR:-vim} /etc/ais/openais.conf
```

The only section that you should need to change is the `totem` - interface section.

### `/etc/ais/openais.conf`

```
totem {
    ...
    interface {
        ringnumber: 0
        bindnetaddr: 10.24.22.144
        mcastaddr: 226.94.1.1
        mcastport: 5405
    }
}
```

The default `mcastaddr` and `mcastport` is probably fine. You need to change the `bindnetaddr` to match the address of the network interface that this node will use to communicate with other nodes in the cluster.

Now, edit `/etc/corosync/corosync.conf`, as well. The same change will need to be made to the `totem`-interface section in that file.

## 4. Running OpenAIS

While testing, I recommend starting the `aisexec` application in the foreground so that you can see debug messages that verify that the nodes have discovered each other and joined the cluster.

```
$ sudo aisexec -f
```

For example, here is some sample output from the first server after starting `aisexec` on the second server:

```
Nov 13 06:55:30 corosync [CLM ] CLM CONFIGURATION CHANGE
Nov 13 06:55:30 corosync [CLM ] New Configuration:
Nov 13 06:55:30 corosync [CLM ]         r(0) ip(10.24.22.144)
Nov 13 06:55:30 corosync [CLM ]         r(0) ip(10.24.22.242)
Nov 13 06:55:30 corosync [CLM ] Members Left:
Nov 13 06:55:30 corosync [CLM ] Members Joined:
Nov 13 06:55:30 corosync [CLM ]         r(0) ip(10.24.22.242)
Nov 13 06:55:30 corosync [TOTEM] A processor joined or left the membership and a new membership was formed.
Nov 13 06:55:30 corosync [MAIN] Completed service synchronization, ready to provide service.
```

## 5. Installing Asterisk

Install Asterisk as usual. Just make sure that you run the configure script after OpenAIS gets installed. That way, it will find the AIS header files and will let you build the `res_ais` module. Check `menuselect` to make sure that `res_ais` is going to get compiled and installed.

```
$ cd asterisk-source
$ ./configure

$ make menuselect
---> Resource Modules
```

If you have existing configuration on the system being used for testing, just be sure to install the addition configuration file needed for `res_ais`.

```
$ sudo cp configs/ais.conf.sample /etc/asterisk/ais.conf
```

## 6. Configuring Asterisk

First, ensure that you have a unique "entity ID" set for each server.

```
*CLI> core show settings
...
Entity ID:                01:23:45:67:89:ab
```

The code will attempt to generate a unique entity ID for you by reading MAC addresses off of a network interface. However, you can also set it manually in the [options] section of asterisk.conf.

```
$ sudo ${EDITOR:-vim} /etc/asterisk/asterisk.conf
```

### asterisk.conf

```
[options]

entity_id=01:23:45:67:89:ab
```

Edit the Asterisk ais.conf to enable distributed events. For example, if you would like to enable distributed device state, you should add the following section to the file:

```
$ sudo ${EDITOR:-vim} /etc/asterisk/ais.conf
```

### /etc/asterisk/ais.conf

```
[device_state]
type=event_channel
publish_event=device_state
subscribe_event=device_state
```

For more information on the contents and available options in this configuration file, please see the sample configuration file:

```
$ cd asterisk-source
$ less configs/ais.conf.sample
```

## 7. Basic Testing of Asterisk with OpenAIS

If you have OpenAIS successfully installed and running, as well as Asterisk with OpenAIS support successfully installed, configured, and running, then you are ready to test out some of the AIS functionality in Asterisk.

The first thing to test is to verify that all of the nodes that you think should be in your cluster are actually there. There is an Asterisk CLI command which will list the current cluster members using the AIS Cluster Membership Service (CLM).

```
*CLI> ais clm show members

=====
=== Cluster Members =====
=====
===
=== -----
=== Node Name: 10.24.22.144
=== ==> ID: 0x9016180a
=== ==> Address: 10.24.22.144
=== ==> Member: Yes
=== -----
===
=== -----
=== Node Name: 10.24.22.242
=== ==> ID: 0xf216180a
=== ==> Address: 10.24.22.242
=== ==> Member: Yes
=== -----
===
=====
```



If you're having trouble getting the nodes of the cluster to see each other, make sure you do not have firewall rules that are blocking the multicast traffic that is used to communicate amongst the nodes.

The next thing to do is to verify that you have successfully configured some event channels in the Asterisk `ais.conf` file. This command is related to the event service (EVT), so like the previous command, uses the syntax: `ais <service name> <command>`.

```
*CLI> ais evt show event channels

=====
=== Event Channels =====
=====
===
=== -----
=== Event Channel Name: device_state
=== ==> Publishing Event Type: device_state
=== ==> Subscribing to Event Type: device_state
=== -----
===
=====
```

## 8. Testing Distributed Device State

The easiest way to test distributed device state is to use the `DEVICE_STATE()` dialplan function. For example, you could have the following piece of dialplan on every server:

```
/etc/asterisk/extensions.conf

[devstate_test]

exten => 1234, hint, Custom:mystate
```

Now, you can test that the cluster-wide state of "Custom:mystate" is what you would expect after going to the CLI of each server and adjusting the state.

```
server1*CLI> dialplan set global DEVICE_STATE(Custom:mystate) NOT_INUSE
...

server2*CLI> dialplan set global DEVICE_STATE(Custom:mystate) INUSE
...
```

Various combinations of setting and checking the state on different servers can be used to verify that it works as expected. Also, you can see the status of the hint on each server, as well, to see how extension state would reflect the state change with distributed device state:

```
server2*CLI> core show hints
-- Registered Asterisk Dial Plan Hints --
      1234@devstate_test      : Custom:mystate      State:InUse      Watchers  0
```

One other helpful thing here during testing and debugging is to enable debug logging. To do so, enable debug on the console in `/etc/asterisk/logger.conf`. Also, enable debug at the Asterisk CLI.

```
*CLI> core set debug 1
```

When you have this debug enabled, you will see output during the processing of every device state change. The important thing to look for is where the known state of the device for each server is added together to determine the overall state.

# Distributed Device State with XMPP PubSub

- 1. Introduction
- 2. Tigase Installation
  - Description
  - Download
  - Install
  - 2.1. Tigase Configuration
    - Generating the keystore file
    - Configuring init.properties
  - 2.2. Running Tigase
  - 2.3. Adding Buddies to Tigase
- 3. Installing Asterisk
  - 3.1. Configuring Asterisk
- 4. Basic Testing of Asterisk with XMPP PubSub
- 5. Testing Distributed Device State
- 6. Notes On Large Installations

## 1. Introduction

This document describes installing and utilizing XMPP PubSub events to distribute device state and message waiting indication (MWI) events between servers. The difference between this method and OpenAIS (see [Distributed Device State with AIS](#)) is that OpenAIS can only be used in low latency networks; meaning only on the LAN, and not across the internet.

If you plan on distributing device state or MWI across the internet, then you will require the use of XMPP PubSub events.

## 2. Tigase Installation

### **Description**

Currently the only server supported for XMPP PubSub events is the Tigase open source XMPP/Jabber environment. This is the server that the various Asterisk servers will connect to in order to distribute the events. The Tigase server can even be clustered in order to provide high availability for your device state; however, that is beyond the scope of this document.

For more information about Tigase, visit their web site <http://www.tigase.org/>.

### **Download**

To download the Tigase environment, get the latest version at <http://www.tigase.org/content/tigase-downloads>. Some distributions have Tigase packaged, as well.

### **Install**

The Tigase server requires a working Java environment, including both a JRE (Java Runtime Environment) and a JDK (Java Development Kit), currently at least version 1.6.

For more information about how to install Tigase, see the web site <http://www.tigase.org/content/quick-start>.

### 2.1. Tigase Configuration

While installing Tigase, be sure you enable the PubSub module. Without it, the PubSub events won't be accepted by the server, and your device state will not be distributed.

There are a couple of things you need to configure in Tigase before you start it in order for Asterisk to connect. The first thing we need to do is generate the self-signed certificate. To do this we use the keytool application. More information can be found here <http://www.tigase.org/content/server-certificate>.

#### **Generating the keystore file**

Generally, we need to run the following commands to generate a new keystore file.

```
# cd /opt/Tigase-4.3.1-bl858/certs
```

Be sure to change the 'yourdomain' to your domain.

```
# keytool -genkey -alias yourdomain -keystore rsa-keystore -keyalg RSA -sigalg MD5withRSA
```

The keytool application will then ask you for a password. Use the password 'keystore' as this is the default password that Tigase will use to load the keystore file.

You then need to specify your domain as the first value to be entered in the security certificate.

```
What is your first and last name?
[Unknown]: asterisk.mydomain.tld
What is the name of your organizational unit?
[Unknown]:
What is the name of your organization?
[Unknown]:
What is the name of your City or Locality?
[Unknown]:
What is the name of your State or Province?
[Unknown]:
What is the two-letter country code for this unit?
[Unknown]:
Is CN=asterisk.mydomain.tld, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
[no]: yes
```

You will then be asked for another password, in which case you must just press enter for the same password as Tigase will not work without them being the same.

```
Enter key password for <mykey>
(RETURN if same as keystore password):
```

### Configuring `init.properties`

The next step is to configure the `init.properties` file which is used by Tigase to generate the `tigase.xml` file. Whenever you change the `init.properties` file because sure to remove the current `tigase.xml` file so that it will be regenerated at start up.

```
# cd /opt/Tigase-4.3.1-b1858/etc
```

Then edit the `init.properties` file and add the following:

```
config-type=--gen-config-def
--admins=admin@asterisk.mydomain.tld
--virt-hosts=asterisk.mydomain.tld
--debug=server
--user-db=derby
--user-db-uri=jdbc:derby:/opt/Tigase-4.3.1-b1858
--comp-name-1=pubsub
--comp-class-1=tigase.pubsub.PubSubComponent
```

Be sure to change the domain in the `--admin` and `--virt-hosts` options. The most important lines are `--comp-name-1` and `--comp-class-1` which tell Tigase to load the PubSub module.

## 2.2. Running Tigase

You can then start the Tigase server with the `tigase.sh` script.

```
# cd /opt/Tigase-4.3.1-b1858
# ./scripts/tigase.sh start etc/tigase.conf
```

## 2.3. Adding Buddies to Tigase

At this time, Asterisk is not able to automatically register your peers for you, so you'll need to use an external application to do the initial registration.

Pidgin is an excellent multi-protocol instant messenger application which supports XMPP. It runs on Linux, Windows, and OSX, and is open source. You can get Pidgin from <http://www.pidgin.im>

Then add the two buddies we'll use in Asterisk with Pidgin by connecting to the Tigase server. For more information about how to register new buddies, see the Pidgin documentation.

Once the initial registration is done and loaded into Tigase, you no longer need to worry about using Pidgin. Asterisk will then be able to load the peers into memory at start up.

The example peers we've used in the following documentation for our two nodes are:

```
server1@asterisk.mydomain.tld/astvoip1
server2@asterisk.mydomain.tld/astvoip2
```

## 3. Installing Asterisk

Install Asterisk as usual. However, you'll need to make sure you have the `res_jabber` module compiled, which requires the `iksemel` development library. Additionally, be sure you have the OpenSSL development library installed so you can connect securely to the Tigase server.

Make sure you check `menuselect` that `res_jabber` is selected so that it will compile.



```
# cd asterisk-source
# ./configure

# make menuselect
----> Resource Modules
```

If you don't have jabber.conf in your existing configuration, because sure to copy the sample configuration file there.

```
# cd configs
# cp jabber.conf.sample /etc/asterisk/jabber.conf
```

### 3.1. Configuring Asterisk

We then need to configure our servers to communicate with the Tigase server. We need to modify the jabber.conf file on the servers. The configurations below are for a 2 server setup, but could be expanded for additional servers easily.

The key note here is to note that the pubsub\_node option needs to start with pubsub, so for example, pubsub.asterisk.mydomain.tld. Without the 'pubsub' your Asterisk system will not be able to distribute events.

Additionally, you will need to specify each of the servers you need to connect to using the 'buddy' option.

\*Asterisk Server 1

#### jabber.conf on server1

```
[general]
debug=no                ;;Turn on debugging by default.
;autoprunce=yes        ;;Auto remove users from buddy list. Depending on
your
for a test)            ;;setup (ie, using your personal Gtalk account
'no'.                  ;;you might lose your contacts list. Default is
autoregister=yes       ;;Auto register users from buddy list.
;collection_nodes=yes  ;;Enable support for XEP-0248 for use with
                        ;;distributed device state. Default is 'no'.
;pubsub_autocreate=yes ;;Whether or not the PubSub server supports/is
using                  ;;auto-create for nodes. If it is, we have to
                        ;;explicitly pre-create nodes before publishing
them.                  ;;Default is 'no'.

[asterisk]
type=client
serverhost=asterisk.mydomain.tld
pubsub_node=pubsub.asterisk.mydomain.tld
username=server1@asterisk.mydomain.tld/astvoip1
secret=welcome
distribute_events=yes
status=available
usetls=no
usesasl=yes
buddy=server2@asterisk.mydomain.tld/astvoip2
```

Asterisk Server 2

## jabber.conf on server2

```
[general]
debug=yes      ;;Turn on debugging by default.
;autoprune=yes  ;;Auto remove users from buddy list. Depending on your
                ;;setup (ie, using your personal Gtalk account for a test)
                ;;you might lose your contacts list. Default is 'no'.
autoregister=yes  ;;Auto register users from buddy list.
;collection_nodes=yes  ;;Enable support for XEP-0248 for use with
                ;;distributed device state. Default is 'no'.
;pubsub_autocreate=yes  ;;Whether or not the PubSub server supports/is using
                ;;auto-create for nodes. If it is, we have to
                ;;explicitly pre-create nodes before publishing them.
                ;;Default is 'no'.

[asterisk]
type=client
serverhost=asterisk.mydomain.tld
pubsub_node=pubsub.asterisk.mydomain.tld
username=server2@asterisk.mydomain.tld/astvoip2
secret=welcome
distribute_events=yes
status=available
usetls=no
usesasl=yes
buddy=server1@asterisk.mydomain.tld/astvoip1
```

## 4. Basic Testing of Asterisk with XMPP PubSub

Once you have Asterisk installed with XMPP PubSub, it is time to test it out.

We need to start up our first server and make sure we get connected to the XMPP server. We can verify this with an Asterisk console command to determine if we're connected.

On Asterisk 1 we can run 'jabber show connected' to verify we're connected to the XMPP server.

```
*CLI> jabber show connected
Jabber Users and their status:
  User: server1@asterisk.mydomain.tld/astvoip1    - Connected
----
Number of users: 1
```

The command above has given us output which verifies we've connected our first server.

We can then check the state of our buddies with the 'jabber show buddies' CLI command.

```
*CLI> jabber show buddies
Jabber buddy lists
Client: server1@asterisk.mydomain.tld/astvoip1
Buddy: server2@asterisk.mydomain.tld
Resource: None
Buddy: server2@asterisk.mydomain.tld/astvoip2
Resource: None
```

The output above tells us we're not connected to any buddies, and thus we're not distributing state to anyone (or getting it from anyone). That makes sense since we haven't yet started our other server.

Now, let's start the other server and verify the servers are able to establish a connection between each other.

On Asterisk 2, again we run the 'jabber show connected' command to make sure we've connected successfully to the XMPP server.

```
*CLI> jabber show connected
Jabber Users and their status:
  User: server2@asterisk.mydomain.tld/astvoip2    - Connected
----
Number of users: 1
```

And now we can check the status of our buddies.

```
*CLI> jabber show buddies
Jabber buddy lists
Client: server2@scooter/astvoip2
Buddy: server1@asterisk.mydomain.tld
Resource: astvoip1
  node: http://www.asterisk.org/xmpp/client/caps
  version: asterisk-xmpp
  Jingle capable: yes
Status: 1
Priority: 0
Buddy: server1@asterisk.mydomain.tld/astvoip1
Resource: None
```

Excellent! So we're connected to the buddy on Asterisk 1, and we could run the same command on Asterisk 1 to verify the buddy on Asterisk 2 is seen.

## 5. Testing Distributed Device State

The easiest way to test distributed device state is to use the `DEVICE_STATE()` dialplan function. For example, you could have the following piece of dialplan on every server:

```
[devstate_test]

exten => 1234, hint, Custom:mystate

exten => set_inuse, 1, Set(DEVICE_STATE(Custom:mystate)=INUSE)
exten => set_not_inuse, 1, Set(DEVICE_STATE(Custom:mystate)=NOT_INUSE)

exten => check, 1, NoOp(Custom:mystate is ${DEVICE_STATE(Custom:mystate)})
```

Now, you can test that the cluster-wide state of "Custom:mystate" is what you would expect after going to the CLI of each server and adjusting the state.

```
server1*CLI> console dial set_inuse@devstate_test
...

server2*CLI> console dial check@devstate_test
-- Executing [check@devstate_test:1] NoOp("OSS/dsp", "Custom:mystate is INUSE") in new stack
```

Various combinations of setting and checking the state on different servers can be used to verify that it works as expected. Also, you can see the status of the hint on each server, as well, to see how extension state would reflect the state change with distributed device state:

```
server2*CLI> core show hints
-- Registered Asterisk Dial Plan Hints ==
      1234@devstate_test      : Custom:mystate      State:InUse      Watchers  0
```

One other helpful thing here during testing and debugging is to enable debug logging. To do so, enable debug on the console in `/etc/asterisk/logger.conf`. Also, enable debug at the Asterisk CLI.

```
*CLI> core set debug 1
```

When you have this debug enabled, you will see output during the processing of every device state change. The important thing to look for is where the known state of the device for each server is added together to determine the overall state.

## 6. Notes On Large Installations

On larger installations where you want a fully meshed network of buddies (i.e. all servers have all the buddies of the remote servers), you may want some method of distributing those buddies automatically so you don't need to modify all servers (N+1) every time you add a new server to the cluster.

The problem there is that you're confined by what's allowed in XEP-0060, and unfortunately that means modifying affiliations by individual JID (as opposed to the various subscription access models, which are more flexible).

See here for details <http://xmpp.org/extensions/xep-0060.html#owner-affiliations>

One method for making this slightly easier is to utilize the `#exec` functionality in configuration files, and dynamically generate the buddies via script that pulls the information from a database, or to `#include` a file which is automatically generated on all the servers when you add a new node to the cluster.

Unfortunately this still requires a reload of `res_jabber.so` on all the servers, but this could also be solved through the use of the Asterisk Manager Interface (AMI).

So while this is not the ideal situation, it is programmatically solvable with existing technologies and features found in Asterisk today.

# Simple Network Management Protocol (SNMP) Support

## Asterisk SNMP Support

Rudimentary support for SNMP access to Asterisk is available. To build this, one needs to have Net-SNMP development headers and libraries on the build system, including any libraries Net-SNMP depends on.

Note that on some (many?) Linux-distributions the dependency list in the net-snmp-devel list is not complete, and additional packages will need to be installed. This is usually seen as configure failing to detect net-snmp-devel as the configure script does a sanity check of the net-snmp build environment, based on the output of 'net-snmp-config --agent-libs'.

To see what your distribution requires, run:

```
'net-snmp-config --agent-libs'.
```

You will receive a response similar to the following:

```
-L/usr/lib -lnetsnmpmibs -lnetsnmpagent -lnetsnmphelpers -lnetsnmp -ldl  
-lrpm -lrpmio -lpopt -lz -lcrypto -lm -lsensors -L/usr/lib/lib -lwrap  
-Wl,-E -Wl,-rpath,/usr/lib/perl5/5.8.8/i386-linux-thread-multi/CORE  
-L/usr/local/lib  
/usr/lib/perl5/5.8.8/i386-linux-thread-multi/auto/DynaLoader/DynaLoader.a  
-L/usr/lib/perl5/5.8.8/i386-linux-thread-multi/CORE -lperl -lresolv -lnsl  
-ldl -lm -lcrypt -lutil -lpthread -lc
```

The packages required may include the following:

- bzip2-devel
- lm\_sensors-devel
- newt-devel

SNMP support comes in two varieties – as a sub-agent to a running SNMP daemon using the AgentX protocol, or as a full standalone agent. If you wish to run a full standalone agent, Asterisk must run as root in order to bind to port 161.

Configuring access when running as a full agent is something that is left as an exercise to the reader.

To enable access to the Asterisk SNMP subagent from a master SNMP daemon, one will need to enable AgentX support, and also make sure that Asterisk will be able to access the Unix domain socket. One way of doing this is to add the following to /etc/snmp/snmpd.conf:

```
# Enable AgentX support  
master agentx  
  
# Set permissions on AgentX socket and containing  
# directory such that process in group 'asterisk'  
# will be able to connect  
agentXPerms 0660 0550 nobody asterisk
```

This assumes that you run Asterisk under group 'asterisk' (and does not care what user you run as).

# Asterisk MIB Definitions

```
ASTERISK-MIB DEFINITIONS ::= BEGIN

IMPORTS
    OBJECT-TYPE, MODULE-IDENTITY, Integer32, Counter32, TimeTicks,
    Unsigned32, Gauge32
    FROM SNMPv2-SMI

    TEXTUAL-CONVENTION, DisplayString, TruthValue
    FROM SNMPv2-TC

    digium
    FROM DIGIUM-MIB;

asterisk MODULE-IDENTITY
    LAST-UPDATED "200806202025Z"
    ORGANIZATION "Digium, Inc."
    CONTACT-INFO
        "Mark A. Spencer
        Postal: Digium, Inc.
            445 Jan Davis Drive
            Huntsville, AL 35806
            USA
        Tel: +1 256 428 6000
        Email: markster@digium.com

        Thorsten Lockert
        Postal: Voop AS
            Boehmergaten 42
            NO-5057 Bergen
            Norway
        Tel: +47 5598 7200
        Email: tholo@voop.no"
    DESCRIPTION
        "Asterisk is an Open Source PBX. This MIB defined
        objects for managing Asterisk instances."
    REVISION "200806202025Z"
    DESCRIPTION
        "smilint police --
        Add missing imports; fix initial capitalization
        of enumeration elements; add missing range
        restrictions for Integer32 indices, correct
        spelling of astChanCidANI in its definition.
        Addresses bug 12905. - jeffg@opennms.org"
    REVISION "200708211450Z"
    DESCRIPTION
        "Add total and current call counter statistics."
    REVISION "200603061840Z"
    DESCRIPTION
        "Change audio codec identification from 3kAudio to
        Audio3k to conform better with specification.

        Expand on contact information."
    REVISION "200602041900Z"
    DESCRIPTION
        "Initial published revision."
    ::= { digium 1 }

asteriskVersion OBJECT IDENTIFIER ::= { asterisk 1 }
asteriskConfiguration OBJECT IDENTIFIER ::= { asterisk 2 }
asteriskModules OBJECT IDENTIFIER ::= { asterisk 3 }
asteriskIndications OBJECT IDENTIFIER ::= { asterisk 4 }
asteriskChannels OBJECT IDENTIFIER ::= { asterisk 5 }

-- asteriskVersion

astVersionString OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Text version string of the version of Asterisk that
        the SNMP Agent was compiled to run against."
    ::= { asteriskVersion 1 }

astVersionTag OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "SubVersion revision of the version of Asterisk that
        the SNMP Agent was compiled to run against -- this is
        typically 0 for release-versions of Asterisk."
    ::= { asteriskVersion 2 }

-- asteriskConfiguration
```

```

astConfigUpTime OBJECT-TYPE
SYNTAX TimeTicks
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Time ticks since Asterisk was started."
 ::= { asteriskConfiguration 1 }

astConfigReloadTime OBJECT-TYPE
SYNTAX TimeTicks
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Time ticks since Asterisk was last reloaded."
 ::= { asteriskConfiguration 2 }

astConfigPid OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The process id of the running Asterisk process."
 ::= { asteriskConfiguration 3 }

astConfigSocket OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The control socket for giving Asterisk commands."
 ::= { asteriskConfiguration 4 }

astConfigCallsActive OBJECT-TYPE
SYNTAX Gauge32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of calls currently active on the Asterisk PBX."
 ::= { asteriskConfiguration 5 }

astConfigCallsProcessed OBJECT-TYPE
SYNTAX Counter32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The total number of calls processed through the Asterisk PBX since last
restart."
 ::= { asteriskConfiguration 6 }

-- asteriskModules

astNumModules OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Number of modules currently loaded into Asterisk."
 ::= { asteriskModules 1 }

-- asteriskIndications

astNumIndications OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Number of indications currently defined in Asterisk."
 ::= { asteriskIndications 1 }

astCurrentIndication OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Default indication zone to use."
 ::= { asteriskIndications 2 }

astIndicationsTable OBJECT-TYPE
SYNTAX SEQUENCE OF AstIndicationsEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Table with all the indication zones currently know to
the running Asterisk instance."
 ::= { asteriskIndications 3 }

astIndicationsEntry OBJECT-TYPE
SYNTAX AstIndicationsEntry

```

```

MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
 "Information about a single indication zone."
INDEX { astIndIndex }
 ::= { astIndicationsTable 1 }

AstIndicationsEntry ::= SEQUENCE {
 astIndIndex Integer32,
 astIndCountry DisplayString,
 astIndAlias DisplayString,
 astIndDescription DisplayString
 }

astIndIndex OBJECT-TYPE
SYNTAX Integer32 (1 .. 2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "Numerical index into the table of indication zones."
 ::= { astIndicationsEntry 1 }

astIndCountry OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "Country for which the indication zone is valid,
 typically this is the ISO 2-letter code of the country."
 ::= { astIndicationsEntry 2 }

astIndAlias OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 ""
 ::= { astIndicationsEntry 3 }

astIndDescription OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "Description of the indication zone, usually the full
 name of the country it is valid for."
 ::= { astIndicationsEntry 4 }

-- asteriskChannels

astNumChannels OBJECT-TYPE
SYNTAX Gauge32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "Current number of active channels."
 ::= { asteriskChannels 1 }

astChanTable OBJECT-TYPE
SYNTAX SEQUENCE OF AstChanEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
 "Table with details of the currently active channels
 in the Asterisk instance."
 ::= { asteriskChannels 2 }

astChanEntry OBJECT-TYPE
SYNTAX AstChanEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
 "Details of a single channel."
INDEX { astChanIndex }
 ::= { astChanTable 1 }

AstChanEntry ::= SEQUENCE {
 astChanIndex Integer32,
 astChanName DisplayString,
 astChanLanguage DisplayString,
 astChanType DisplayString,
 astChanMusicClass DisplayString,
 astChanBridge DisplayString,
 astChanMasq DisplayString,
 astChanMasqr DisplayString,
 astChanWhenHangup TimeTicks,
 astChanApp DisplayString,
 astChanData DisplayString,
 astChanContext DisplayString,

```

```

astChanMacroContext DisplayString,
astChanMacroExten DisplayString,
astChanMacroPri Integer32,
astChanExten DisplayString,
astChanPri Integer32,
astChanAccountCode DisplayString,
astChanForwardTo DisplayString,
astChanUniqueId DisplayString,
astChanCallGroup Unsigned32,
astChanPickupGroup Unsigned32,
astChanState INTEGER,
astChanMuted TruthValue,
astChanRings Integer32,
astChanCidDNID DisplayString,
astChanCidNum DisplayString,
astChanCidName DisplayString,
astChanCidANI DisplayString,
astChanCidRDNIS DisplayString,
astChanCidPresentation DisplayString,
astChanCidANI2 Integer32,
astChanCidTON Integer32,
astChanCidTNS Integer32,
astChanAMAFlags INTEGER,
astChanADSI INTEGER,
astChanToneZone DisplayString,
astChanHangupCause INTEGER,
astChanVariables DisplayString,
astChanFlags BITS,
astChanTransferCap INTEGER
}

astChanIndex OBJECT-TYPE
SYNTAX Integer32 (1 .. 2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Index into the channel table."
 ::= { astChanEntry 1 }

astChanName OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Name of the current channel."
 ::= { astChanEntry 2 }

astChanLanguage OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Which language the current channel is configured to
use -- used mainly for prompts."
 ::= { astChanEntry 3 }

astChanType OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Underlying technology for the current channel."
 ::= { astChanEntry 4 }

astChanMusicClass OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Music class to be used for Music on Hold for this
channel."
 ::= { astChanEntry 5 }

astChanBridge OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Which channel this channel is currently bridged (in a
conversation) with."
 ::= { astChanEntry 6 }

astChanMasq OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Channel masquerading for us."
 ::= { astChanEntry 7 }

```



```

astChanMasqr OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Channel we are masquerading for."
 ::= { astChanEntry 8 }

astChanWhenHangup OBJECT-TYPE
SYNTAX TimeTicks
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"How long until this channel will be hung up."
 ::= { astChanEntry 9 }

astChanApp OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Current application for the channel."
 ::= { astChanEntry 10 }

astChanData OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Arguments passed to the current application."
 ::= { astChanEntry 11 }

astChanContext OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Current extension context."
 ::= { astChanEntry 12 }

astChanMacroContext OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Current macro context."
 ::= { astChanEntry 13 }

astChanMacroExten OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Current macro extension."
 ::= { astChanEntry 14 }

astChanMacroPri OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Current macro priority."
 ::= { astChanEntry 15 }

astChanExten OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Current extension."
 ::= { astChanEntry 16 }

astChanPri OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Current priority."
 ::= { astChanEntry 17 }

astChanAccountCode OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Account Code for billing."
 ::= { astChanEntry 18 }

```

```

astChanForwardTo OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Where to forward to if asked to dial on this
    interface."
 ::= { astChanEntry 19 }

astChanUniqueId OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Unique Channel Identifier."
 ::= { astChanEntry 20 }

astChanCallGroup OBJECT-TYPE
SYNTAX Unsigned32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Call Group."
 ::= { astChanEntry 21 }

astChanPickupGroup OBJECT-TYPE
SYNTAX Unsigned32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Pickup Group."
 ::= { astChanEntry 22 }

astChanState OBJECT-TYPE
SYNTAX INTEGER {
    stateDown(0),
    stateReserved(1),
    stateOffHook(2),
    stateDialing(3),
    stateRing(4),
    stateRinging(5),
    stateUp(6),
    stateBusy(7),
    stateDialingOffHook(8),
    statePreRing(9)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Channel state."
 ::= { astChanEntry 23 }

astChanMuted OBJECT-TYPE
SYNTAX TruthValue
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Transmission of voice data has been muted."
 ::= { astChanEntry 24 }

astChanRings OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Number of rings so far."
 ::= { astChanEntry 25 }

astChanCidDNID OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Dialled Number ID."
 ::= { astChanEntry 26 }

astChanCidNum OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Caller Number."
 ::= { astChanEntry 27 }

astChanCidName OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION

```

```

"Caller Name."
::= { astChanEntry 28 }

astChanCidANI OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"ANI"
::= { astChanEntry 29 }

astChanCidRDNIS OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Redirected Dialed Number Service."
::= { astChanEntry 30 }

astChanCidPresentation OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Number Presentation/Screening."
::= { astChanEntry 31 }

astChanCidANI2 OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"ANI 2 (info digit)."
::= { astChanEntry 32 }

astChanCidTON OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Type of Number."
::= { astChanEntry 33 }

astChanCidTNS OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Transit Network Select."
::= { astChanEntry 34 }

astChanAMAFlags OBJECT-TYPE
SYNTAX INTEGER {
    default(0),
    omit(1),
    billing(2),
    documentation(3)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"AMA Flags."
::= { astChanEntry 35 }

astChanADSI OBJECT-TYPE
SYNTAX INTEGER {
    unknown(0),
    available(1),
    unavailable(2),
    offHookOnly(3)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Whether or not ADSI is detected on CPE."
::= { astChanEntry 36 }

astChanToneZone OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Indication zone to use for channel."
::= { astChanEntry 37 }

astChanHangupCause OBJECT-TYPE
SYNTAX INTEGER {
    notDefined(0),
    unregistered(3),

```

```

normal(16),
busy(17),
noAnswer(19),
congestion(34),
failure(38),
noSuchDriver(66)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"why is the channel hung up."
 ::= { astChanEntry 38 }

astChanVariables OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Channel Variables defined for this channel."
 ::= { astChanEntry 39 }

astChanFlags OBJECT-TYPE
SYNTAX BITS {
wantsJitter(0),
deferDTMF(1),
writeInterrupt(2),
blocking(3),
zombie(4),
exception(5),
musicOnHold(6),
spying(7),
nativeBridge(8),
autoIncrementingLoop(9)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Flags set on this channel."
 ::= { astChanEntry 40 }

astChanTransferCap OBJECT-TYPE
SYNTAX INTEGER {
speech(0),
digital(8),
restrictedDigital(9),
audio3k(16),
digitalWithTones(17),
video(24)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Transfer Capabilities for this channel."
 ::= { astChanEntry 41 }

astNumChanTypes OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Number of channel types (technologies) supported."
 ::= { asteriskChannels 3 }

astChanTypeTable OBJECT-TYPE
SYNTAX SEQUENCE OF AstChanTypeEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Table with details of the supported channel types."
 ::= { asteriskChannels 4 }

astChanTypeEntry OBJECT-TYPE
SYNTAX AstChanTypeEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"Information about a technology we support, including
how many channels are currently using this technology."
INDEX { astChanTypeIndex }
 ::= { astChanTypeTable 1 }

AstChanTypeEntry ::= SEQUENCE {
astChanTypeIndex Integer32,
astChanTypeName DisplayString,
astChanTypeDesc DisplayString,
astChanTypeDeviceState Integer32,
astChanTypeIndications Integer32,
astChanTypeTransfer Integer32,
astChanTypeChannels Gauge32

```

```

}

astChanTypeIndex OBJECT-TYPE
SYNTAX Integer32 (1 .. 2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Index into the table of channel types."
 ::= { astChanTypeEntry 1 }

astChanTypeName OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Unique name of the technology we are describing."
 ::= { astChanTypeEntry 2 }

astChanTypeDesc OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Description of the channel type (technology).".
 ::= { astChanTypeEntry 3 }

astChanTypeDeviceState OBJECT-TYPE
SYNTAX TruthValue
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Whether the current technology can hold device states."
 ::= { astChanTypeEntry 4 }

astChanTypeIndications OBJECT-TYPE
SYNTAX TruthValue
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Whether the current technology supports progress indication."
 ::= { astChanTypeEntry 5 }

astChanTypeTransfer OBJECT-TYPE
SYNTAX TruthValue
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Whether the current technology supports transfers, where
    Asterisk can get out from inbetween two bridged channels."
 ::= { astChanTypeEntry 6 }

astChanTypeChannels OBJECT-TYPE
SYNTAX Gauge32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Number of active channels using the current technology."
 ::= { astChanTypeEntry 7 }

astChanScalars OBJECT IDENTIFIER ::= { asteriskChannels 5 }

astNumChanBridge OBJECT-TYPE
SYNTAX Gauge32
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "Number of channels currently in a bridged state."

```

```
 ::= { astChanScalars 1 }  
END
```

## Digium MIB Definitions

```
DIGIUM-MIB DEFINITIONS ::= BEGIN

IMPORTS
    enterprises, MODULE-IDENTITY
        FROM SNMPv2-SMI;

digium MODULE-IDENTITY
    LAST-UPDATED "200806202000Z"
    ORGANIZATION "Digium, Inc."
    CONTACT-INFO
        "Mark Spencer
         Email: markster@digium.com"
    DESCRIPTION
        "The Digium private-enterprise MIB"
    REVISION "200806202000Z"
    DESCRIPTION
        "Corrected imports and missing revision for last update.
         Addresses bug 12905. - jeffg@opennms.org"
    REVISION "200602041900Z"
    DESCRIPTION
        "Initial revision."
    ::= { enterprises 22736 }

END
```

# Speech Recognition API

## The Asterisk Speech Recognition API

The generic speech recognition engine is implemented in the `res_speech.so` module. This module connects through the API to speech recognition software, that is not included in the module.

To use the API, you must load the `res_speech.so` module before any connectors. For your convenience, there is a preload line commented out in the `modules.conf` sample file.

### Dialplan Applications:

The dialplan API is based around a single speech utilities application file, which exports many applications to be used for speech recognition. These include an application to prepare for speech recognition, activate a grammar, and play back a sound file while waiting for the person to speak. Using a combination of these applications you can easily make a dialplan use speech recognition without worrying about what speech recognition engine is being used.

#### SpeechCreate(Engine Name)

This application creates information to be used by all the other applications. It must be called before doing any speech recognition activities such as activating a grammar. It takes the engine name to use as the argument, if not specified the default engine will be used.

If an error occurs are you are not able to create an object, the variable `ERROR` will be set to 1. You can then exit your speech recognition specific context and play back an error message, or resort to a DTMF based IVR.

#### SpeechLoadGrammar(Grammar Name|Path)

Loads grammar locally on a channel. Note that the grammar is only available as long as the channel exists, and you must call `SpeechUnloadGrammar` before all is done or you may cause a memory leak. First argument is the grammar name that it will be loaded as and second argument is the path to the grammar.

#### SpeechUnloadGrammar(Grammar Name)

Unloads a locally loaded grammar and frees any memory used by it. The only argument is the name of the grammar to unload.

#### SpeechActivateGrammar(Grammar Name)

This activates the specified grammar to be recognized by the engine. A grammar tells the speech recognition engine what to recognize, and how to portray it back to you in the dialplan. The grammar name is the only argument to this application.

#### SpeechStart()

Tell the speech recognition engine that it should start trying to get results from audio being fed to it. This has no arguments.

#### SpeechBackground(Sound File|Timeout)

This application plays a sound file and waits for the person to speak. Once they start speaking playback of the file stops, and silence is heard. Once they stop talking the processing sound is played to indicate the speech recognition engine is working. Note it is possible to have more than one result. The first argument is the sound file and the second is the timeout. Note the timeout will only start once the sound file has stopped playing.

#### SpeechDeactivateGrammar(Grammar Name)

This deactivates the specified grammar so that it is no longer recognized. The only argument is the grammar name to deactivate.

#### SpeechProcessingSound(Sound File)

This changes the processing sound that `SpeechBackground` plays back when the speech recognition engine is processing and working to get results. It takes the sound file as the only argument.

#### SpeechDestroy()

This destroys the information used by all the other speech recognition applications. If you call this application but end up wanting to recognize more speech, you must call `SpeechCreate` again before calling any other application. It takes no arguments.

### Getting Result Information:

The speech recognition utilities module exports several dialplan functions that you can use to examine results.

- `${SPEECH(status)}` - Returns 1 if `SpeechCreate` has been called. This uses the same check that applications do to see if a speech object is setup. If it returns 0 then you know you can not use other speech applications.
- `${SPEECH(spoke)}` - Returns 1 if the speaker spoke something, or 0 if they were silent.
- `${SPEECH(results)}` - Returns the number of results that are available.
- `${SPEECH_SCORE(result number)}` - Returns the score of a result.
- `${SPEECH_TEXT(result number)}` - Returns the recognized text of a result.
- `${SPEECH_GRAMMAR(result number)}` - Returns the matched grammar of the result.
- `SPEECH_ENGINE(name)=value` - Sets a speech engine specific attribute.

### Dialplan Flow:



1. Create a speech recognition object using `SpeechCreate()`
2. Activate your grammars using `SpeechActivateGrammar(Grammar Name)`
3. Call `SpeechStart()` to indicate you are going to do speech recognition immediately
4. Play back your audio and wait for recognition using `SpeechBackground(Sound File|Timeout)`
5. Check the results and do things based on them
6. Deactivate your grammars using `SpeechDeactivateGrammar(Grammar Name)`
7. Destroy your speech recognition object using `SpeechDestroy()`

#### Dialplan Examples:

This is pretty cheeky in that it does not confirmation of results. As well the way the grammar is written it returns the person's extension instead of their name so we can just do a Goto based on the result text.

#### Grammar: company-directory.gram

```
#ABNF 1.0;
language en-US;
mode voice;
tag-format <lumenvox/1.0>;
root $company_directory;

$josh = ((Joshua | Josh) [Colp]):"6066";
$mark = (Mark [Spencer] | Markster):"4569";
$kevin = (Kevin [Fleming]):"2567";

$company_directory = ($josh | $mark | $kevin) { $ = $$ };
```

#### Dialplan logic

#### extensions.conf

```
[dial-by-name]
exten => s,1,SpeechCreate()
exten => s,2,SpeechActivateGrammar(company-directory)
exten => s,3,SpeechStart()
exten => s,4,SpeechBackground(who-would-you-like-to-dial)
exten => s,5,SpeechDeactivateGrammar(company-directory)
exten => s,6,Goto(internal-extensions-${SPEECH_TEXT(0)})
```

#### Useful Dialplan Tidbits

A simple macro that can be used for confirm of a result. Requires some sound files. ARG1 is equal to the file to play back after "I heard..." is played.

## extensions.conf

```
[macro-speech-confirm]
exten => s,1,SpeechActivateGrammar(yes_no)
exten => s,2,Set(OLDTEXT0=${SPEECH_TEXT(0)})
exten => s,3,Playback(heard)
exten => s,4,Playback(${ARG1})
exten => s,5,SpeechStart()
exten => s,6,SpeechBackground(correct)
exten => s,7,Set(CONFIRM=${SPEECH_TEXT(0)})
exten => s,8,GotoIf("${SPEECH_TEXT(0)}" = "1"?9:10)
exten => s,9,Set(CONFIRM=yes)
exten => s,10,Set(CONFIRMED=${OLDTEXT0})
exten => s,11,SpeechDeactivateGrammar(yes_no)
```

## The Asterisk Speech Recognition C API

The module `res_speech.so` exports a C based API that any developer can use to speech recognize enable their application. The API gives greater control, but requires the developer to do more on their end in comparison to the dialplan speech utilities.

For all API calls that return an integer value, a non-zero value indicates an error has occurred.

### Creating a speech structure

```
struct ast_speech *ast_speech_new(char *engine_name, int format)

struct ast_speech *speech = ast_speech_new(NULL, AST_FORMAT_SLINEAR);
```

This will create a new speech structure that will be returned to you. The speech recognition engine name is optional and if NULL the default one will be used. As well for now format should always be `AST_FORMAT_SLINEAR`.

### Activating a grammar

```
int ast_speech_grammar_activate(struct ast_speech *speech, char *grammar_name)

res = ast_speech_grammar_activate(speech, "yes_no");
```

This activates the specified grammar on the speech structure passed to it.

### Start recognizing audio

```
void ast_speech_start(struct ast_speech *speech)

ast_speech_start(speech);
```

This essentially tells the speech recognition engine that you will be feeding audio to it from then on. It MUST be called every time before you start feeding audio to the speech structure.

### Send audio to be recognized

```
int ast_speech_write(struct ast_speech *speech, void *data, int len)

res = ast_speech_write(speech, fr->data, fr->datalen);
```

This writes audio to the speech structure that will then be recognized. It must be written signed linear only at this time. In the future other formats may be supported.

### Checking for results

The way the generic speech recognition API is written is that the speech structure will undergo state changes to indicate progress of recognition. The states are outlined below:

- `AST_SPEECH_STATE_NOT_READY` - The speech structure is not ready to accept audio
- `AST_SPEECH_STATE_READY` - You may write audio to the speech structure
- `AST_SPEECH_STATE_WAIT` - No more audio should be written, and results will be available soon.
- `AST_SPEECH_STATE_DONE` - Results are available and the speech structure can only be used again by calling `ast_speech_start`

It is up to you to monitor these states. Current state is available via a variable on the speech structure. (`state`)

### Knowing when to stop playback

If you are playing back a sound file to the user and you want to know when to stop play back because the individual started talking use the following.

```
ast_test_flag(speech, AST_SPEECH_QUIET) /* This will return a positive value when the
person has started talking. */
```

### Getting results

```
struct ast_speech_result *ast_speech_results_get(struct ast_speech *speech)

struct ast_speech_result *results = ast_speech_results_get(speech);
```

This will return a linked list of result structures. A result structure looks like the following:

```
struct ast_speech_result {
    char *text;                /*!< Recognized text */
    int score;                 /*!< Result score */
    char *grammar;            /*!< Matched grammar */
    struct ast_speech_result *next; /*!< List information */
};
```

### Freeing a set of results

```
int ast_speech_results_free(struct ast_speech_result *result)

res = ast_speech_results_free(results);
```

This will free all results on a linked list. Results MAY NOT be used as the memory will have been freed.

### Deactivating a grammar

```
int ast_speech_grammar_deactivate(struct ast_speech *speech, char *grammar_name)

res = ast_speech_grammar_deactivate(speech, "yes_no");
```

This deactivates the specified grammar on the speech structure.

### Destroying a speech structure

```
int ast_speech_destroy(struct ast_speech *speech)

res = ast_speech_destroy(speech);
```

This will free all associated memory with the speech structure and destroy it with the speech recognition engine.

### Loading a grammar on a speech structure

```
int ast_speech_grammar_load(struct ast_speech *speech, char *grammar_name, char *grammar)
res = ast_speech_grammar_load(speech, "builtin:yes_no", "yes_no");
```

#### **Unloading a grammar on a speech structure**

If you load a grammar on a speech structure it is preferred that you unload it as well, or you may cause a memory leak. Don't say I didn't warn you.

```
int ast_speech_grammar_unload(struct ast_speech *speech, char *grammar_name)
res = ast_speech_grammar_unload(speech, "yes_no");
```

This unloads the specified grammar from the speech structure.

# Utilizing the StatsD Dialplan Application

## Utilizing the StatsD Dialplan Application

This page will document how to use the StatsD dialplan application. Statistics will be logged anytime that a call is made to an extension, within the dialplan, that uses the StatsD application.

### Overview

StatsD is a daemon used to aggregate statistics by using different metrics and then summarizing these statistics in a way that can be useful to the users. One of StatsD's most useful capabilities is its ability to use a graphing back-end to display the statistics graphically. StatsD makes this very simple by accepting statistics through a short, one-line command and then grouping and arranging the statistics for you.

This StatsD application is a dialplan application that is used to send statistics automatically whenever a call is made to an extension that employs the application. The user must provide the arguments to the application in the dialplan, but after that, the application will send statistics to StatsD without requiring the user to perform anymore actions whenever a call comes through that extension.

### Setup

To send statistics to a StatsD server, you first need to have a StatsD server able to accept the metrics you are sending. This does not have to be the same machine that contains your asterisk instance.

Installing StatsD on a machine will make the machine a StatsD server. A backend can then be installed on the same machine as a way of viewing the statistics that StatsD receives. StatsD already has the capability built in to support a backend. Most backends are graphs that you can view through a browser and allow you to track statistics that you have sent in real time.

After a StatsD server has been set up, all that is needed is to provide the IP address of the server and the port to *statsd.conf*. You will now have a StatsD server that can accept statistics and your dialplan application will send statistics directly to this server. If a backend is configured on the StatsD server, then the backend will automatically interact with StatsD when statistics are sent to the server.

If you wish to set up your own StatsD server, then you can download StatsD from here: <https://github.com/etsy/statsd>.

A list of available backends to be used is available here: <https://github.com/etsy/statsd/wiki/Backends>.

### Requirements

Only a few requirements are needed for working with the StatsD application.

- You need a statsd capable backend to receive statistics.
- Build Asterisk with the *res\_statsd* module. This provides the StatsD dialplan application.
- Configure and enable StatsD support in Asterisk's *statsd.conf*

### Configuration

To send statistics from the dialplan application to a StatsD server, the only options that you need from *statsd.conf* are *enabled* and *server*.

- *enabled*- whether or not StatsD support is enabled in Asterisk. To use StatsD, this must be set to yes.
- *server*- the address of the StatsD server. A port is not required, and if one is not provided, will be used 8125 as the default port.

#### *statsd.conf*

```
[general]
enabled = yes      ; When set to yes, statsd support is enabled
server = 127.0.0.1 ; server[:port] of statsd server to use.
                  ; If not specified, the port is 8125
;prefix =         ; Prefix to prepend to all metrics
;add_newline = no ; Append a newline to every event. This is
                  ; useful if you want to run a fake statsd
                  ; server using netcat (nc -lu 8125)
```

If you wish to add a port, such as 8126, to the server address, then you would add it at the end of the address with a colon, like so: *127.0.0.1:8126*.

#### *extension.conf*

```
[default]

exten => 100,1,NoOp( )
  same => n,StatsD(g,confBridgeUsers,+1,1)
  same => n,Set(CHANNEL(hangup_handler_push)=default,200,1);
  same => n,ConfBridge(1000)
  same => n,StatsD(g,confBridgeUsers,-1,1)
  same => n,Hangup( )

exten => 200,1,NoOp( )
  same => n,StatsD(g,confBridgeUsers,-1,1)
  same => n,Hangup( )
```

## Example

The image below is an example of how calling into the dialplan provided above would send statistics to a StatsD server.

- The number of members in the conference initially is at 0. When someone calls into the conference, the gauge *confBridgeUsers* is incremented by 1, and the graph below shows the count of *confBridgeUsers* to be 1.
- When another person calls in and joins the conference, the count is incremented to 2, and the graph displays that two people are in the conference.
- When one person hangs up or is kicked from the conference, the count will decrement, showing that only one person remains in the conference.
- When the final person hangs up, the count of *confBridgeUsers* is decremented again, and the value of *confBridgeUsers* is again 0.



Note: this graph is not a part of StatsD, but is rather a backend that StatsD can be configured to use. This configuration would occur on the StatsD server.

# Deployment

## Basic PBX Functionality

In this section, we're going to guide you through the basic setup of a very primitive PBX. After you finish, you'll have a basic PBX with two phones that can dial each other. In later modules, we'll go into more detail on each of these steps, but in the meantime, this will give you a basic system on which you can learn and experiment.



# The Most Basic PBX

## Requirements and Assumptions

While it won't be anything to brag about, this basic PBX that you will build from Asterisk will help you learn the fundamentals of configuring Asterisk.

For this exercise, we're going to assume that you have the following:

**A machine, virtual or real, with Asterisk already installed.**

Got here without installing Asterisk? Head back to the [Installation Asterisk](#) section. Be sure to install the SIP Channel Driver module that you want to use for SIP connectivity. This tutorial will cover using `chan_sip` and `res_pjsip/chan_pjsip`.

**Two or more phones which speak the SIP voice-over-IP protocol.**

There are a wide variety of SIP phones available in many different shapes and sizes, and if your budget doesn't allow for you to buy phones, feel free to use a free soft phone. [Softphones](#) are simply computer programs which run on your computer and emulate a real phone, and communicate with other devices across your network, just like a real voice-over-IP phone would. If you do use a soft phone, remember to watch out for software firewalls blocking traffic from or to the system hosting the softphone.

# Creating SIP Accounts

In order for our phones to communicate with each other, we need to configure an account for each phone in the channel driver which corresponds to the protocol they'll be using. Since the phones are using the SIP protocol, we actually have two options for a SIP channel driver, the configuration file would be **sip.conf** for `chan_sip`, or **pjsip.conf** for `chan_pjsip/res_pjsip` (`res_pjsip` actually provides the configuration). You may already know that `chan_pjsip` is only available in Asterisk 12 or later. These files reside in the Asterisk configuration directory, which is typically `/etc/asterisk`. We'll include separate instructions for each channel driver below, so you have the option of using either.



For the sake of the examples in this section, you should only configure and use one SIP channel driver at a time. Advanced configuration is needed to use two different SIP channel drivers simultaneously.

## Use of templates

In the below examples of channel driver configuration we will use templates. Using templates can make the configuration seem a bit more confusing at first, but in the long run it will simplify managing larger groups of extensions. Templates are used to define sections of options or settings that can then be inherited by a child section. [Read about templates here](#). The short story is that parents are defined with a bang in parentheses after their section name, and children specify a parent in the parentheses after their section name.

### On this page

- [Use of templates](#)
- [Configuring `chan\_sip`](#)
- [Configuring `chan\_pjsip`](#)
- [Loading the configuration](#)

## Configuring `chan_sip`

Open **sip.conf** with your favorite text editor, and spend a minute or two looking at the sample file. (Don't let it overwhelm you — the sample **sip.conf** has a **lot** of data in it, and can be overwhelming at first glance.) Notice that there are a couple of sections at the top of the configuration, such as `[general]` and `[authentication]`, which control the overall functionality of the channel driver. Below those sections, there are sections which correspond to SIP accounts on the system.

Now copy the sample `sip.conf` to something like `sip.conf.sample` and create a new blank `sip.conf` to work with.

Create the following sections of configuration in the `sip.conf` file. Let's name your phones **Alice** and **Bob**, so that we can easily differentiate between them.

```
[general]
transport=udp

[friends_internal]()
type=friend
host=dynamic
context=from-internal
disallow=all
allow=ulaw

[demo-alice](friends_internal)
secret=verysecretpassword ; put a strong, unique password here instead

[demo-bob](friends_internal)
secret=othersecretpassword ; put a strong, unique password here instead
```

You can read what each of the options do in the `sip.conf` sample file. However, let's take a look at why we are using each option here.

In the general section:

- `transport=udp`: sets the general transport used by all `chan_sip` accounts defined in configuration if they don't have their own transport defined. We want to use UDP

In the template for `friends_internal`:

- `type=friend`: we are using the friend type to make things easy. See the `sip.conf.sample` for more explanation on the section types available in `sip.conf`
- `host=dynamic`: our phones will register to Asterisk. Otherwise we would define the IP address of the phone here.
- `context=from-internal`: When Asterisk receives a call from this phone, it'll look for the dialed extension number inside this context within dialplan (`/etc/asterisk/extensions.conf` typically)
- `disallow=all`; Don't allow any codecs to be used except what is set in 'allow'
- `allow=ulaw`; Only allow the ulaw codec to be used.

In the sections for `demo-alice` and `demo-bob`:

- `secret=verysecretpassword`: This is the authentication password the phone needs to use when authenticating against Asterisk.

Note that in `chan_sip` configuration, the authentication username for each SIP account is the section name itself. That is "demo-alice" is the name you'll

have your phone authenticate against when registering. This will be covered more in the next wiki section on registering phones to Asterisk.

## Configuring chan\_pjsip

Take a minute to look over the pjsip.conf sample file if you haven't already. Then backup your pjsip.conf file and create a new blank one.

Add the following configuration:

```
[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0

;Templates for the necessary config sections

[endpoint_internal]()
type=endpoint
context=from-internal
disallow=all
allow=ulaw

[auth_userpass]()
type=auth
auth_type=userpass

[aor_dynamic]()
type=aor
max_contacts=1

;Definitions for our phones, using the templates above

[demo-alice](endpoint_internal)
auth=demo-alice
aors=demo-alice
[demo-alice](auth_userpass)
password=unsecurepassword ; put a strong, unique password here instead
username=demo-alice
[demo-alice](aor_dynamic)

[demo-bob](endpoint_internal)
auth=demo-bob
aors=demo-bob
[demo-bob](auth_userpass)
password=unsecurepassword ; put a strong, unique password here instead
username=demo-bob
[demo-bob](aor_dynamic)
```

This configuration is roughly comparable to the chan\_sip configuration in its end result. The pjsip configuration is a little more complex as the channel driver's architecture allows for more flexibility in configuration, so things tend to be more modular and broken out.

Each configuration section has a type.

In the **transport** type section we configure the following:

- `type=transport`; This defines a transport that can be used by other configuration objects/sections, such as endpoints
- `protocol=udp`; We want to use the UDP protocol
- `bind=0.0.0.0`; We want to communicate over the most appropriate available interfaces.

In the **endpoint** template, `endpoint_internal`:

- `type=endpoint`; This defines an endpoint; a primary configuration section that you could think of as a profile of settings for a SIP connection
- `context=from-internal`; When Asterisk receives a call from this phone, it'll look for the dialed extension number inside this context within dialplan (/etc/asterisk/extensions.conf typically)
- `disallow=all`; Don't allow any codecs to be used except what is set in 'allow'
- `allow=ulaw`; Only allow the ulaw codec to be used.

In the `auth` template, `auth_userpass`:

- `type=auth`; This defines an auth section which describes credentials used for authentication. Any particular endpoint or registration will typically reference an auth section by name.
- `auth_type=userpass`; This tells Asterisk to use the 'username' and 'password' options set in this auth. In this case we'll be setting those in the children auths that inherit this template.

In the **aor** template, `aor_dynamic`:

- `type=aor`; This defines an aor section which describes location information making up an Address of Record. Endpoints make use of an AOR so that Asterisk knows where to send calls to the endpoint.
- `max_contacts=1`; We want to allow up to a maximum of one registration to this AOR. That is, we are only going to have one phone registering for each AOR.

In the AOR type configuration section (see `type=aor`) we purposefully don't define any contacts, and instead we set "`max_contacts=1`", this allows us to register phones to the AORs configured for each endpoint. This accomplishes similar behavior to "`host=dynamic`" for account entries in the older `chan_sip`

driver.

After the templates, you'll see the definitions of the **demo-alice** and **demo-bob** endpoints. In those, all we do is tie them to their own auth and aor sections. In their **auth** sections we set their unique usernames and passwords. In their aor sections we don't set anything, because the necessary settings were already set in the template.

If you want to explore pjsip configuration more deeply after you finish this PBX tutorial, check out the [Configuring res\\_pjsip](#) wiki section.

## Loading the configuration

We have more configuration to do in the next section, so there is no need to load configuration yet. However if you do want to go ahead and load the configuration; note that for some channel drivers, like chan\_pjsip, certain configuration sections (transport, system) can't be reloaded at runtime. Therefore it is easiest to restart Asterisk completely here since we have made set transport settings.

```
#service asterisk restart  
or  
#asterisk -rx "core restart now"
```

# Registering Phones to Asterisk

The next step is to configure the phones themselves to communicate with Asterisk. The way we have configured the accounts in the SIP channel driver, Asterisk will expect the phones to **register** to it. Registration is simply a mechanism where a phone communicates "Hey, I'm Bob's phone... here's my username and password. Oh, and if you get any calls for me, I'm at this particular IP address."

Configuring your particular phone is obviously beyond the scope of this guide, but here are a list of common settings you're going to want to set in your phone, so that it can communicate with Asterisk:

- **Registrar/Registration Server** - The location of the server which the phone should register to. This should be set to the IP address of your Asterisk system.
- **SIP User Name/Account Name/Address** - The SIP username on the remote system. This should be set to demo-alice on one phone and demo-bob on the other. This username corresponds directly to the section name in sip.conf.
- **SIP Authentication User/Auth User** - On Asterisk-based systems, this will be the same as the SIP user name above.
- **Proxy Server/Outbound Proxy Server** - This is the server with which your phone communicates to make outside calls. This should be set to the IP address of your Asterisk system.

When using chan\_sip you can tell whether or not your phone has registered successfully to Asterisk by checking the output of the **sip show peers** command at the Asterisk CLI. If the **Host** column says **(Unspecified)**, the phone has not yet registered. On the other hand, if the **Host** column contains an IP address and the **Dyn** column contains the letter **D**, you know that the phone has successfully registered.

```
server*CLI> sip show peers
Name/username      Host              Dyn NAT ACL Port      Status
demo-alice         (Unspecified)    D      A  5060      Unmonitored
demo-bob           192.168.5.105    D      A  5060      Unmonitored
2 sip peers [Monitored: 0 online, 0 offline Unmonitored: 2 online, 0 offline]
```

In the example above, you can see that Alice's phone has not registered, but Bob's phone has registered.

For chan\_pjsip you can use **pjsip show endpoints**.



### Debugging SIP Registrations

If you're having troubles getting a phone to register to Asterisk, make sure you watch the Asterisk CLI with the verbosity level set to at least three while you reboot the phone. You'll likely see error messages indicating what the problem is, like in this example:

```
NOTICE[22214]: chan_sip.c:20824 handle_request_register: Registration from '"Alice"&nbsp;
<sip:demo-alice@192.168.5.50>' failed for '192.168.5.103' - Wrong password
```

As you can see, Asterisk has detected that the password entered into the phone doesn't match the secret setting in the [demo-alice] section of sip.conf.

# Creating Dialplan Extensions

The last things we need to do to enable Alice and Bob to call each other is to configure a couple of extensions in the dialplan.



## What is an Extension?

When dealing with Asterisk, the term extension does not represent a physical device such as a phone. An extension is simply a set of actions in the dialplan which may or may not write a physical device. In addition to writing a phone, an extensions might be used for such things auto-attendant menus and conference bridges. In this guide we will be careful to use the words phone or device when referring to the physical phone, and extension when referencing the set of instructions in the Asterisk dialplan.

Let's take a quick look at the dialplan, and then add two extensions.

Open **extensions.conf**, and take a quick look at the file. Near the top of the file, you'll see some general-purpose sections named [general] and [globals]. Any sections in the dialplan beneath those two sections is known as a **context**. The sample **extensions.conf** file has a number of other contexts, with names like [demo] and [default].

We cover the concept of contexts more in [Dialplan](#), but for now you should know that each phone or outside connection in Asterisk points at a single context. If the dialed extension does not exist in the specified context, Asterisk will reject the call. That means it is important to understand that the **context** option in your sip.conf or pjsip.conf configuration is what tells Asterisk to direct the call from the endpoint to the context we build in the next step.

Go to the bottom of your **extensions.conf** file, and add a new context named **[from-internal]** since from-internal is what we configured for the context option in the [Creating SIP Accounts](#) page.

## Naming Your Dialplan Contexts

There's nothing special about the name **from-internal** for this context. It could have been named **strawberry\_milkshake**, and it would have behaved exactly the same way. It is considered best practice, however, to name your contexts for the types of extensions that are contained in that context. Since this context contains extensions that will be dialing from inside the network, we'll call it from-internal.

Underneath that context name, we'll create an extension numbered **6001** which attempts to ring Alice's phone for twenty seconds, and an extension **6002** which attempts to ring Bob's phone for twenty seconds.

```
[from-internal]
exten=>6001,1,Dial(SIP/demo-alice,20)
exten=>6002,1,Dial(SIP/demo-bob,20)
```

After adding that section to **extensions.conf**, go to the Asterisk command-line interface and tell Asterisk to reload the dialplan by typing the command **dialplan reload**. You can verify that Asterisk successfully read the configuration file by typing **dialplan show from-internal** at the CLI.

```
server*CLI> dialplan show from-internal
[ Context 'from-internal' created by 'pbx_config' ]
'6001' =>      1. Dial(SIP/demo-alice,20)           [pbx_config]
'6002' =>      1. Dial(SIP/demo-bob,20)           [pbx_config]

-= 2 extensions (2 priorities) in 1 context. =-
```

Now we're ready to make a test call!



Learn more about dialplan format in the [Contexts, Extensions, and Priorities](#) section.

# Making a Phone Call

At this point, you should be able to pick up Alice's phone and dial extension **6002** to call Bob, and dial **6001** from Bob's phone to call Alice. As you make a few test calls, be sure to watch the Asterisk command-line interface (and ensure that your verbosity is set to a value three or higher) so that you can see the messages coming from Asterisk, which should be similar to the ones below:

```
server*CLI>    -- Executing [6002@from-internal:1] Dial("SIP/demo-alice-00000000", "SIP/demo-bob,20") in new stack
-- Called demo-bob
-- SIP/demo-bob-00000001 is ringing
-- SIP/demo-bob-00000001 answered SIP/demo-alice-00000000
-- Native bridging SIP/demo-alice-00000000 and SIP/demo-bob-00000001
== Spawn extension (from-internal, 6002, 1) exited non-zero on 'SIP/demo-alice-00000000'
```

As you can see, Alice called extension **6002** in the `[from-internal]` context, which in turn used the **Dial** application to call Bob's phone. Bob's phone rang, and then answered the call. Asterisk then bridged the two calls (one call from Alice to Asterisk, and the other from Asterisk to Bob), until Alice hung up the phone.

At this point, you have a very basic PBX. It has two extensions which can dial each other, but that's all. Before we move on, however, let's review a few basic troubleshooting steps that will help you be more successful as you learn about Asterisk.



## Basic PBX Troubleshooting

The most important troubleshooting step is to set your verbosity level to three (or higher), and watch the command-line interface for errors or warnings as calls are placed.

To ensure that your SIP phones are registered, type **sip show peers**(chan\_sip), or **pjsip show endpoints**(chan\_pjsip) at the Asterisk CLI.

To see which context your SIP phones will send calls to, type **sip show users**(chan\_sip) or **pjsip show endpoint <endpoint name>**(chan\_pjsip).

To ensure that you've created the extensions correctly in the `[from-internal]` context in the dialplan, type **dialplan show from-internal**.

To see which extension will be executed when you dial extension **6002**, type **dialplan show 6002@from-internal**.

## Auto-attendant and IVR Menus

In this section, we'll cover the how to build voice menus, often referred to as auto-attendants and IVR menus. IVR stands for *Interactive Voice Response*, and is used to describe a system where a caller navigates through a system by using the touch-tone keys on their phone keypad.

When the caller presses a key on their phone keypad, the phone emits two tones, known as DTMF tones. DTMF stands for *Dual Tone Multi-Frequency*. Asterisk recognizes the DTMF tones and responds accordingly.

Let's dive in and learn how to build IVR menus in the Asterisk dialplan!



## Background and WaitExten Applications

The **Background()** application plays a sound prompt, but listens for DTMF input. Asterisk then tries to find an extension in the current dialplan context that matches the DTMF input. If it finds a matching extension, Asterisk will send the call to that extension.

The Background() application takes the name of the sound prompt as the first parameter just like the Playback() application, so remember not to include the file extension.



### Multiple Prompts

If you have multiple prompts you'd like to play during the Background() application, simply concatenate them together with the ampersand (&) character, like this:

```
exten => 6123,1,Background(prompt1&prompt2&prompt3)
```

One problem you may encounter with the **Background()** application is that you may want Asterisk to wait a few more seconds after playing the sound prompt. In order to do this, you can call the **WaitExten()** application. You'll usually see the **WaitExten()** application called immediately after the **Background()** application. The first parameter to the **WaitExten()** application is the number of seconds to wait for the caller to enter an extension. If you don't supply the first parameter, Asterisk will use the built-in response timeout (which can be modified with the **TIMEOUT()** dialplan function).

```
[auto_attendant]
exten => start,1,Verbose(2,Incoming call from ${CALLERID(all)})
    same => n,Playback(silence/1)
    same => n,Background(prompt1&prompt2&prompt3)
    same => n,WaitExten(10)
    same => n,Goto(timeout-handler,1)

exten => timeout-handler,1)
    same => n,Dial(${GLOBAL(OPERATOR)},30)
    same => n,Voicemail(operator@default,${IF(${DIALSTATUS} = BUSY)?b:u})
    same => n,Hangup()
```

## Goto Application and Priority Labels

Before we create a simple auto-attendant menu, let's cover a couple of other useful dialplan applications. The **Goto()** application allows us to jump from one position in the dialplan to another. The parameters to the **Goto()** application are slightly more complicated than with the other applications we've looked at so far, but don't let that scare you off.

The **Goto()** application can be called with either one, two, or three parameters. If you call the **Goto()** application with a single parameter, Asterisk will jump to the specified priority (or its label) within the current extension. If you specify two parameters, Asterisk will read the first as an extension within the current context to jump to, and the second parameter as the priority (or label) within that extension. If you pass three parameters to the application, Asterisk will assume they are the context, extension, and priority (respectively) to jump to.

```
[StartingContext]
exten => 100,1,Goto(monkeys)
    same => n,NoOp(We skip this)
    same => n(monkeys),Playback(tt-monkeys)
    same => n,Hangup()

exten => 200,1,Goto(start,1) ; play tt-weasels then tt-monkeys

exten => 300,1,Goto(start,monkeys) ; only play tt-monkeys

exten => 400,1,Goto(JumpingContext,start,1) ; play hello-world

exten => start,1,NoOp()
    same => n,Playback(tt-weasels)
    same => n(monkeys),Playback(tt-monkeys)

[JumpingContext]
exten => start,1,NoOp()
    same => n,Playback(hello-world)
    same => n,Hangup()
```

## SayDigits, SayNumber, SayAlpha, and SayPhonetic Applications

While not exactly related to auto-attendant menus, we'll introduce some applications to read back various pieces of information back to the caller. The **SayDigits()** and **SayNumber()** applications read the specified number back to caller. To use the **SayDigits()** and **SayNumber()** application simply pass it the number you'd like it to say as the first parameter.

The **SayDigits()** application reads the specified number one digit at a time. For example, if you called **SayDigits(123)**, Asterisk would read back "one two three". On the other hand, the **SayNumber()** application reads back the number as if it were a whole number. For example, if you called **SayNumber(123)** Asterisk would read back "one hundred twenty three".

The **SayAlpha()** and **SayPhonetic()** applications are used to spell an alphanumeric string back to the caller. The **SayAlpha()** reads the specified string one letter at a time. For example, **SayAlpha(hello)** would read spell the word "hello" one letter at a time. The **SayPhonetic()** spells back a string one letter at a time, using the international phonetic alphabet. For example, **SayPhonetic(hello)** would read back "Hotel Echo Lima Lima Oscar".

We'll use these four applications to read back various data to the caller throughout this guide. In the meantime, please feel free to add some sample extensions to your dialplan to try out these applications. Here are some examples:

```
exten => 6592,1,SayDigits(123)
exten => 6593,1,SayNumber(123)
exten => 6594,1,SayAlpha(hello)
exten => 6595,1,SayPhonetic(hello)
```

## Creating a Simple IVR Menu

Let's go ahead and apply what we've learned about the various dialplan applications by building a very simple auto-attendant menu. It is common practice to create an auto-attendant or IVR menu in a new context, so that it remains independent of the other extensions in the dialplan. Please add the following to your dialplan (the `extensions.conf` file) to create a new **demo-menu** context. In this new context, we'll create a simple menu that prompts you to enter one or two, and then it will read back what you're entered.



### Sample Sound Prompts

Please note that the example below (and many of the other examples in this guide) use sound prompts that are part of the *extra* sounds packages. If you didn't install the extra sounds earlier, now might be a good time to do that.

```
[demo-menu]
exten => s,1,Answer(500)
    same => n(loop),Background(press-1&or&press-2)
    same => n,WaitExten()

exten => 1,1,Playback(you-entered)
    same => n,SayNumber(1)
    same => n,Goto(s,loop)

exten => 2,1,Playback(you-entered)
    same => n,SayNumber(2)
    same => n,Goto(s,loop)
```

Before we can use the demo menu above, we need to add an extension to the `[docs:users]` context to redirect the caller to our menu. Add this line to the `[docs:users]` context in your dialplan:

```
exten => 6598,1,Goto(demo-menu,s,1)
```

Reload your dialplan, and then try dialing extension **6598** to test your auto-attendant menu.

## Handling Special Extensions

We have the basics of an auto-attendant created, but now let's make it a bit more robust. We need to be able to handle special situations, such as when the caller enters an invalid extension, or doesn't enter an extension at all. Asterisk has a set of special extensions for dealing with situations like there. They all are named with a single letter, so we recommend you don't create any other extensions named with a single letter. You can read about all the [Special Dialplan Extensions on the wiki](#).

Let's add a few more lines to our `[docs:demo-menu]` context, to handle invalid entries and timeouts. Modify your `[docs:demo-menu]` context so that it matches the one below:

```
[demo-menu]
exten => s,1,Answer(500)
    same => n(loop),Background(press-1&or&press-2)
    same => n,WaitExten()

exten => 1,1,Playback(you-entered)
    same => n,SayNumber(1)
    same => n,Goto(s,loop)

exten => 2,1,Playback(you-entered)
    same => n,SayNumber(2)
    same => n,Goto(s,loop)

exten => i,1,Playback(option-is-invalid)
    same => n,Goto(s,loop)

exten => t,1,Playback(are-you-still-there)
    same => n,Goto(s,loop)
```

Now dial your auto-attendant menu again (by dialing extension **6598**), and try entering an invalid option (such as **3**) at the auto-attendant menu. If you watch the Asterisk command-line interface while you dial and your verbosity level is three or higher, you should see something similar to the following:

```
-- Executing [6598@users:1] Goto("SIP/demo-alice-00000008", "demo-menu,s,1") in new stack
-- Goto (demo-menu,s,1)
-- Executing [s@demo-menu:1] Answer("SIP/demo-alice-00000008", "500") in new stack
-- Executing [s@demo-menu:2] Background("SIP/demo-alice-00000008", "press-1&or&press-2") in new stack
-- <SIP/demo-alice-00000008> Playing 'press-1.gsm' (language 'en')
-- <SIP/demo-alice-00000008> Playing 'or.gsm' (language 'en')
-- <SIP/demo-alice-00000008> Playing 'press-2.gsm' (language 'en')
-- Invalid extension '3' in context 'demo-menu' on SIP/demo-alice-00000008
-- Executing [i@demo-menu:1] Playback("SIP/demo-alice-00000008", "option-is-invalid") in new stack
-- <SIP/demo-alice-00000008> Playing 'option-is-invalid.gsm' (language 'en')
-- Executing [i@demo-menu:2] Goto("SIP/demo-alice-00000008", "s,loop") in new stack
-- Goto (demo-menu,s,2)
-- Executing [s@demo-menu:2] Background("SIP/demo-alice-00000008", "press-1&or&press-2") in new stack
-- <SIP/demo-alice-00000008> Playing 'press-1.gsm' (language 'en')
-- <SIP/demo-alice-00000008> Playing 'or.gsm' (language 'en')
-- <SIP/demo-alice-00000008> Playing 'press-2.gsm' (language 'en')
```

If you don't enter anything at the auto-attendant menu and instead wait approximately ten seconds, you should hear (and see) Asterisk go to the `t` extension as well.

## Record Application

For creating your own auto-attendant or IVR menus, you're probably going to want to record your own custom prompts. An easy way to do this is with the **Record()** application. The **Record()** application plays a beep, and then begins recording audio until you press the hash key (#) on your keypad. It then saves the audio to the filename specified as the first parameter to the application and continues on to the next priority in the extension. If you hang up the call before pressing the hash key, the audio will not be recorded. For example, the following extension records a sound prompt called **custom-menu** in the **gsm** format in the **en/** sub-directory, and then plays it back to you.

```
exten => 6597,1,Answer(500)
same => n,Record(en/custom-menu.gsm)
same => n,Wait(1)
same => n,Playback(custom-menu)
same => n,Hangup()
```



### Recording Formats

When specifying a file extension when using the **Record()** application, you must choose a file extension which represents one of the supported file formats in Asterisk. For the complete list of file formats supported in your Asterisk installation, type **core show file formats** at the Asterisk command-line interface.

You've now learned the basics of how to create a simple auto-attendant menu. Now let's build a more practical menu for callers to be able to reach Alice or Bob or the dial-by-name directory.

### Procedure 216.1. Building a Practical Auto-Attendant Menu

1. Add an extension 6599 to the [docs:users] context which sends the calls to a new context we'll build called [docs:day-menu]. Your extension should look something like:

```
exten=>6599,1,Goto(day-menu,s,1)
```

2. Add a new context called **[docs:day-menu]**, with the following contents:

```
[day-menu]
exten => s,1,Answer(500)
same => n(loop),Background(custom-menu)
same => n,WaitExten()

exten => 1,1,Goto(users,6001,1)

exten => 2,1,Goto(users,6002,1)

exten => 9,1,Directory(vm-demo,users,fe)
exten => *,1,VoiceMailMain(@vm-demo)

exten => i,1,Playback(option-is-invalid)
same => n,Goto(s,loop)

exten => t,1,Playback(are-you-still-there)
same => n,Goto(s,loop)
```

1. Dial extension **6597** to record your auto-attendant sound prompt. Your sound prompt should say something like "Thank you for calling! Press one for Alice, press two for Bob, or press 9 for a company directory". Press the hash key (#) on your keypad when you're finished recording, and Asterisk will play it back to you. If you don't like it, simply dial extension **6597** again to re-record it.
2. Dial extension **6599** to test your auto-attendant menu.

In just a few lines of code, you've created your own auto-attendant menu. Feel free to experiment with your auto-attendant menu before moving on to the next section.

## Adding Voice Mail to Dialplan Extensions

Adding voicemail to the extensions is quite simple. The [Asterisk voicemail module](#) provides two key applications for dealing with voice mail. The first, named **VoiceMail()**, allows a caller to leave a voice mail message in the specified mailbox. The second, called **VoiceMailMain()**, allows the mailbox owner to retrieve their messages and change their greetings.

## Configuring Voice Mail Boxes

Now that we've covered the two main voice mail applications, let's look at the voicemail configuration. Voice mail options and mailboxes are configured in the **voicemail.conf** configuration file. This file has three major sections:

### The [general] section

Near the top of **voicemail.conf**, you'll find the **[general]** section. This section of the configuration file controls the general aspects of the voicemail system, such as the maximum number of messages per mailbox, the maximum length of a voicemail message, and so forth. Feel free to look at the sample **voicemail.conf** file for more details about the various settings.

### The [zonemessages] section

The **[zonemessages]** section is used to define various timezones around the world. Each mailbox can be assigned to a particular time zone, so that times and dates are announced relative to their local time. The time zones specified in this section also control the way in which times and dates are announced, such as reading the time of day in 24-hour format.

### Voice Mail Contexts

After the **[general]** and **[zonemessages]** sections, any other bracketed section is a voice mail context. Within each context, you can define one or more mailbox. To define a mailbox, we set a mailbox number, a PIN, the mailbox owner's name, the primary email address, a secondary email address, and a list of mailbox options (separated by the pipe character), as shown below:

```
mailbox=>pin,full name,email address,short email address,mailbox options
```

By way of explanation, the short email address is an email address that will receive shorter email notifications suitable for mobile devices such as cell phones and pagers. It will never receive attachments.

To add voice mail capabilities to extensions **6001** and **6002**, add these three lines to the bottom of **voicemail.conf**.

```
[vm-demo]
6001 => 8762,Alice
Jones,alice@example.com,alice2@example.com,attach=no|tz=central|maxmsg=10
6002 => 9271,Bob Smith,bob@example.com,bob2@example.com,attach=yes|tz=eastern
```

Now that we've defined the mailboxes, we can go into the Asterisk CLI and type **voicemail reload** to get Asterisk to reload the **voicemail.conf** file. We can also verify that the new mailboxes have been created by typing **voicemail show users**.

```
server*CLI> voicemail reload
Reloading voicemail configuration...
server*CLI> voicemail show users
Context  Mbox  User                Zone      NewMsg
default  general New User            0
default  1234  Example Mailbox    0
other    1234  Company2 User      0
vm-demo  6001  Alice Jones        central   0
vm-demo  6002  Bob Smith           eastern   0
5 voicemail users configured.
```

Now that we have mailboxes defined, let's add a priority to extensions **6001** and **6002** which will allow callers to leave voice mail in their respective mailboxes. We'll also add an extension **6500** to allow Alice and Bob to check their voicemail messages. Please modify your **[users]** context in **extensions.conf** to look like the following:

```
[users]
exten => 6000,1,Answer(500)
exten => 6000,n,Playback(hello-world)
exten => 6000,n,Hangup()

exten => 6001,1,Dial(SIP/demo-alice,20)
exten => 6001,n,VoiceMail(6001@vm-demo,u)

exten => 6002,1,Dial(SIP/demo-bob,20)
exten => 6002,n,VoiceMail(6002@vm-demo,u)

exten => 6500,1,Answer(500)
exten => 6500,n,VoiceMailMain(@vm-demo)
```



Reload the dialplan by typing **dialplan reload** at the Asterisk CLI. You can then test the voice mail system by dialing from one phone to the other and waiting twenty seconds. You should then be connected to the voicemail system, where you can leave a message. You should also be able to dial extension **6500** to retrieve the voicemail message. When prompted, enter the mailbox number and PIN number of the mailbox.

While in the **VoiceMainMain()** application, you can also record the mailbox owner's name, unavailable greeting, and busy greeting by pressing 0 at the voicemail menu. Please record at least the name greeting for both Alice and Bob before continuing on to the next section.

Go into lots of detail about the voicemail interface? How to move between messages, move between folders, forward messages, etc?

# Deployment In Your Network



Under Construction



Top-level page for pages on dealing with NAT, Firewalling and more in relation to Asterisk.

# Emergency Calling



Under Construction



Top-level page for resources, tutorials or ideas regarding emergency dialing and calling, E-911, etc.

## Important Security Considerations

The pages in this section provide specific warnings about security that are pertinent to Asterisk. Just because you're already familiar with securing your Linux machine, doesn't mean you can skip this section.



PLEASE READ THE FOLLOWING IMPORTANT SECURITY RELATED INFORMATION. IMPROPER CONFIGURATION OF ASTERISK COULD ALLOW UNAUTHORIZED USE OF YOUR FACILITIES, POTENTIALLY INCURRING SUBSTANTIAL CHARGES.

Asterisk security involves both network security (encryption, authentication) as well as dialplan security (authorization - who can access services in your pbx). If you are setting up Asterisk in production use, please make sure you understand the issues involved.

# Network Security

## Network Security

If you install Asterisk and use the "make samples" command to install a demonstration configuration, Asterisk will open a few ports for accepting VoIP calls. Check the channel configuration files for the ports and IP addresses.

If you enable the manager interface in manager.conf, please make sure that you access manager in a safe environment or protect it with SSH or other VPN solutions.

For all TCP/IP connections in Asterisk, you can set ACL lists that will permit or deny network access to Asterisk services. Please check the "permit" and "deny" configuration options in manager.conf and the VoIP channel configurations - i.e. sip.conf and iax.conf.

The IAX2 protocol supports strong RSA key authentication as well as AES encryption of voice and signaling. The SIP channel supports TLS encryption of the signaling, as well as SRTP (encrypted media).

# Dialplan Security

## Dialplan Security

First and foremost remember this:



USE THE EXTENSION CONTEXTS TO ISOLATE OUTGOING OR TOLL SERVICES FROM ANY INCOMING CONNECTIONS.

You should consider that if any channel, incoming line, etc can enter an extension context that it has the capability of accessing any extension within that context.

Therefore, you should NOT allow access to outgoing or toll services in contexts that are accessible (especially without a password) from incoming channels, be they IAX channels, FX or other trunks, or even untrusted stations within you network. In particular, never ever put outgoing toll services in the "default" context. To make things easier, you can include the "default" context within other private contexts by using:

```
include => default
```

in the appropriate section. A well designed PBX might look like this:

```
[longdistance]
exten => _91NXXNXXXXXX,1,Dial(DAHDI/g2/${EXTEN:1})
include => local

[local]
exten => _9NXXNXXX,1,Dial(DAHDI/g2/${EXTEN:1})
include => default

[default]
exten => 6123,Dial(DAHDI/1)
```



DON'T FORGET TO TAKE THE DEMO CONTEXT OUT OF YOUR DEFAULT CONTEXT. There isn't really a security reason, it just will keep people from wanting to play with your Asterisk setup remotely.

## Log Security

### Log Security

Please note that the Asterisk log files, as well as information printed to the Asterisk CLI, may contain sensitive information such as passwords and call history. Keep this in mind when providing access to these resources.

## **Asterisk Security Webinars**

### **Asterisk VoIP Security - Part 1 of 3**

VoIP Fraud: Current Threats From A Law Enforcement Perspective  
Special Agent Michael McAndrews, FBI

### **Asterisk VoIP Security - Part 2 of 3**

VoIP Security Best Practices  
Dan York, Chairman, Best Practices Group, VoIP Security Alliance

### **Asterisk VoIP Security - Part 3 of 3**

Securing Asterisk Systems  
Jared Smith, Training Manager, Digium



# Privacy Configuration

So, you want to avoid talking to pesky telemarketers/charity seekers/poll takers/magazine renewers/etc?

## FTC Don't Call List

The FTC "Don't call" database, this alone will reduce your telemarketing call volume considerably. (see: <https://www.donotcall.gov/default.aspx> ) But, this list won't protect from the Charities, previous business relationships, etc.

## Fighting Autodialers

Zapateller detects if callerid is present, and if not, plays the da-da-da tones that immediately precede messages like, "I'm sorry, the number you have called is no longer in service."

Most humans, even those with unlisted/callerid-blocked numbers, will not immediately slam the handset down on the hook the moment they hear the three tones. But autodialers seem pretty quick to do this.

I just counted 40 hangups in Zapateller over the last year in my CDR's. So, that is possibly 40 different telemarketers/charities that have hopefully slashed my back-waters, out-of-the-way, humble home phone number from their lists.

I highly advise Zapateller for those seeking the nirvana of "privacy".

## Fighting Empty Caller ID

A considerable percentage of the calls you don't want, come from sites that do not provide CallerID.

Null callerid's are a fact of life, and could be a friend with an unlisted number, or some charity looking for a handout. The PrivacyManager application can help here. It will ask the caller to enter a 10-digit phone number. They get 3 tries(configurable), and this is configurable, with control being passed to next priority where you can check the channelvariable PRIVACYMGRSTATUS. If the callerid was valid this variable will have the value SUCCESS, otherwise it will have the value FAILED.

PrivacyManager can't guarantee that the number they supply is any good, tho, as there is no way to find out, short of hanging up and calling them back. But some answers are obviously wrong. For instance, it seems a common practice for telemarketers to use your own number instead of giving you theirs. A simple test can detect this. More advanced tests would be to look for 555 numbers, numbers that count up or down, numbers of all the same digit, etc.

PrivacyManager can be told about a context where you can have patterns that describe valid phone numbers. If none of the patterns match the input, it will be considered a non-valid phonenumber and the user can try again until the retry counter is reached. This helps in resolving the issues stated in the previous paragraph.

My logs show that 39 have hung up in the PrivacyManager script over the last year.

(Note: Demanding all unlisted incoming callers to enter their CID may not always be appropriate for all users. Another option might be to use call screening. See below.)

## Using Welcome Menus for Privacy

Experience has shown that simply presenting incoming callers with a set of options, no matter how simple, will deter them from calling you. In the vast majority of situations, a telemarketer will simply hang up rather than make a choice and press a key.

This will also immediately foil all autodialers that simply belch a message in your ear and hang up.

### Example usage of Zapateller and PrivacyManager

```
[homeline]
exten => s,1,Answer
exten => s,2,SetVar,repeatcount=0
exten => s,3,Zapateller,nocallerid
exten => s,4,PrivacyManager
;; do this if they don't enter a number to Privacy Manager
exten => s,5,GotoIf($[ "${PRIVACYMGRSTATUS}" = "FAILED" ]?s,105)
exten => s,6,GotoIf($[ "${CALLERID(num)}" = "7773334444" & "${CALLERID(name)}" : "Privacy
Manager" ]?callerid-liar,s,1:s,7)
exten => s,7,Dial(SIP/yourphone)
exten => s,105,Background(tt-allbusy)
exten => s,106,Background(tt-somethingwrong)
exten => s,107,Background(tt-monkeysintro)
exten => s,108,Background(tt-monkeys)
exten => s,109,Background(tt-weasels)
exten => s,110,Hangup
```

I suggest using Zapateller at the beginning of the context, before anything else, on incoming calls. This can be followed by the PrivacyManager App.

Make sure, if you do the PrivacyManager app, that you take care of the error condition! or their non-compliance will be rewarded with access to the system. In the above, if they can't enter a 10-digit number in 3 tries, they get the humorous "I'm sorry, but all household members are currently helping other telemarketers...", "something is terribly wrong", "monkeys have carried them away...", various loud monkey screechings, "weasels have...", and a hangup. There are plenty of other paths to my torture scripts, I wanted to have some fun.

In nearly all cases now, the telemarketers/charity-seekers that usually get thru to my main intro, hang up. I guess they can see it's pointless, or the average telemarketer/charity-seeker is instructed not to enter options when encountering such systems. Don't know.

## Making life difficult for telemarketers

I have developed an elaborate script to torture Telemarketers, and entertain friends.

While mostly those that call in and traverse my teletorture scripts are those we know, and are doing so out of curiosity, there have been these others from Jan 1st,2004 thru June 1st, 2004: (the numbers may or may not be correct.)

- 603890zzzz - hung up telemarket options.
- "Integrated Sale" - called a couple times. hung up in telemarket options
- "UNITED STATES GOV" - maybe a military recruiter, trying to lure one of my sons.
- 800349zzzz - hung up in charity intro
- 800349zzzz - hung up in charity choices, intro, about the only one who actually travelled to the bitter bottom of the scripts!
- 216377zzzz - hung up the magazine section
- 626757zzzz = "LIR " (pronounced "Liar"?) hung up in telemarket intro, then choices
- 757821zzzz - hung up in new magazine subscription options.

That averages out to maybe 1 a month. That puts into question whether the ratio of the amount of labor it took to make the scripts versus the benefits of lower call volumes was worth it, but, well, I had fun, so what the heck.

But, that's about it. Not a whole lot. But I haven't had to say "NO" or "GO AWAY" to any of these folks for about a year now ...!

## Using Call Screening

Another option is to use call screening in the Dial command. It has two main privacy modes, one that remembers the CID of the caller, and how the callee wants the call handled, and the other, which does not have a "memory".

Turning on these modes in the dial command results in this sequence of events, when someone calls you at an extension:

The caller calls the Asterisk system, and at some point, selects an option or enters an extension number that would dial your extension.

Before ringing your extension, the caller is asked to supply an introduction. The application asks them: "After the tone, say your name". They are allowed 4 seconds of introduction.

After that, they are told "Hang on, we will attempt to connect you to your party. Depending on your dial options, they will hear ringing indications, or get music on hold. I suggest music on hold.

Your extension is then dialed. When (and if) you pick up, you are told that a caller presenting themselves as their recorded intro is played is calling, and you have options, like being connected, sending them to voicemail, torture, etc.

You make your selection, and the call is handled as you chose.

There are some variations, and these will be explained in due course.

To use these options, set your Dial to something like:

```
exten => 3,3,Dial(DAHDI/5r3&DAHDI/6r3,35,tmpA(beep))
```

or:

```
exten => 3,3,Dial(DAHDI/5r3&DAHDI/6r3,35,tmpP(something)A(beep))
```

or:

```
exten => 3,3,Dial(DAHDI/5r3&DAHDI/6r3,35,tmpA(beep))
```

The **'t'** allows the dialed party to transfer the call using '#'. It's optional.

The **'m'** is for music on hold. I suggest it. Otherwise, the calling party gets to hear all the ringing, and lack thereof. It is generally better to use Music On Hold. Lots of folks hang up after the 3rd or 4th ring, and you might lose the call before you can enter an option!

The **'P'** option alone will database everything using the extension as a default 'tree'. To get multiple extensions sharing the same database, use P(some-shared-key). Also, if the same person has multiple extensions, use P(unique-id) on all their dial commands.

Use little **'p'** for screening. Every incoming call will include a prompt for the callee's choice.

The **A(beep)**, will generate a 'beep' that the callee will hear if they choose to talk to the caller. It's kind of a prompt to let the callee know that he has to say 'hi'. It's not required, but I find it helpful.

When there is no CallerID, **P** and **p** options will always record an intro for the incoming caller. This intro will be stored temporarily in the **/var/lib/asterisk/sounds/priv-callerintros** dir, under the name **NOCALLERID\_extension** channelname and will be erased after the callee decides what to do with the call.

Of course, NOCALLERID is not stored in the database. All those with no CALLERID will be considered "Unknown".

## Call Screening Options

Two other options exist, that act as modifiers to the privacy options 'P' and 'p'. They are 'N' and 'n'. You can enter them as dialing options, but they only affect things if P or p are also in the options.

'N' says, "Only screen the call if no CallerID is present". So, if a callerID were supplied, it will come straight thru to your extension.

'n' says, "Don't save any introductions". Folks will be asked to supply an introduction ("At the tone, say your name") every time they call. Their introductions will be removed after the callee makes a choice on how to handle the call. Whether the P option or the p option is used, the incoming caller will have to supply their intro every time they call.



# Screening Calls with Recorded Introductions

## Philosophical Side Note

The 'P' option stores the CALLERID in the database, along with the callee's choice of actions, as a convenience to the CALLEE, whereas introductions are stored and re-used for the convenience of the CALLER.

## Introductions

Unless instructed to not save introductions (see the 'n' option above), the screening modes will save the recordings of the caller's names in the directory `/var/lib/asterisk/sounds/priv-callerintros`, if they have a CallerID. Just the 10-digit callerid numbers are used as filenames, with a ".gsm" at the end.

Having these recordings around can be very useful, however...

First of all, if a callerid is supplied, and a recorded intro for that number is already present, the caller is spared the inconvenience of having to supply their name, which shortens their call a bit.

Next of all, these intros can be used in voicemail, played over loudspeakers, and perhaps other nifty things. For instance:

```
exten => s,6,Set(PATH=/var/lib/asterisk/sounds/priv-callerintros)
exten => s,7,System(/usr/bin/play ${PATH}/${CALLERID(num)}.gsm&,0)
```

When a call comes in at the house, the above priority gets executed, and the callers intro is played over the phone systems speakers. This gives us a hint who is calling.

(Note: the `.0` option at the end of the System command above, is a local mod I made to the System command. It forces a 0 result code to be returned, whether the play command successfully completed or not. Therefore, I don't have to ensure that the file exists or not. While I've turned this mod into the developers, it hasn't been incorporated yet. You might want to write an AGI or shell script to handle it a little more intelligently)

And one other thing. You can easily supply your callers with an option to listen to, and re-record their introductions. Here's what I did in the home system's `extensions.conf`. (assume that a `Goto(home-introduction,s,1)` exists somewhere in your main menu as an option):

```
[home-introduction]
exten => s,1,Background(intro-options) ;; Script:
;; To hear your Introduction, dial 1.
;; to record a new introduction, dial 2.
;; to return to the main menu, dial 3.
;; to hear what this is all about, dial 4.
exten => 1,1,Playback,priv-callerintros/${CALLERID(num)}
exten => 1,2,Goto(s,1)
exten => 2,1,Goto(home-introduction-record,s,1)
exten => 3,1,Goto(homeline,s,7)
exten => 4,1,Playback(intro-intro) ;; Script:
;; This may seem a little strange, but it really is a neat
;; thing, both for you and for us. I've taped a short introduction
;; for many of the folks who normally call us. Using the Caller ID
;; from each incoming call, the system plays the introduction
;; for that phone number over a speaker, just as the call comes in.
;; This helps the folks
;; here in the house more quickly determine who is calling.
;; and gets the right ones to gravitate to the phone.
;; You can listen to, and record a new intro for your phone number
;; using this menu.
exten => 4,2,Goto(s,1)
exten => t,1,Goto(s,1)
exten => i,1,Background(invalid)
exten => i,2,Goto(s,1)
exten => o,1,Goto(s,1)

[home-introduction-record]
exten => s,1,Background(intro-record-choices) ;; Script:
;; If you want some advice about recording your
```

```
;; introduction, dial 1.
;; otherwise, dial 2, and introduce yourself after
;; the beep.
exten => 1,1,Playback(intro-record)
;; Your introduction should be short and sweet and crisp.
;; Your introduction will be limited to 4 seconds.
;; This is NOT meant to be a voice mail message, so
;; please, don't say anything about why you are calling.
;; After we are done making the recording, your introduction
;; will be saved for playback.
;; If you are the only person that would call from this number,
;; please state your name. Otherwise, state your business
;; or residence name instead. For instance, if you are
;; friend of the family, say, Olie McPherson, and both
;; you and your kids might call here a lot, you might
;; say: "This is the distinguished Olie McPherson Residence!"
;; If you are the only person calling, you might say this:
;; "This is the illustrious Kermit McFrog! Pick up the Phone, someone!!
;; If you are calling from a business, you might pronounce a more sedate introduction,
like,
;; "Fritz from McDonalds calling.", or perhaps the more original introduction:
;; "John, from the Park County Morgue. You stab 'em, we slab 'em!".
;; Just one caution: the kids will hear what you record every time
;; you call. So watch your language!
;; I will begin recording after the tone.
;; When you are done, hit the # key. Gather your thoughts and get
;; ready. Remember, the # key will end the recording, and play back
;; your intro. Good Luck, and Thank you!"
exten => 1,2,Goto(2,1)
exten => 2,1,Background(intro-start)
;; OK, here we go! After the beep, please give your introduction.
exten => 2,2,Background(beep)
exten => 2,3,Record(priv-callerintros/${CALLERID(num)}:gsm,4)
exten => 2,4,Background(priv-callerintros/${CALLERID(num)})
exten => 2,5,Goto(home-introduction,s,1)
exten => t,1,Goto(s,1)
exten => i,1,Background(invalid)
```

```
exten => i,2,Goto(s,1)
exten => o,1,Goto(s,1)
```

In the above, you'd most likely reword the messages to your liking, and maybe do more advanced things with the 'error' conditions (i,o,t priorities), but I hope it conveys the idea.

# Internationalization and Localization



Under Construction



Top-level page for pages discussing Asterisk's capabilities for adapting to various language environments for text, sound prompts and more.

## Asterisk Sounds Packages

Asterisk utilizes a variety of sound prompts that are available in several file formats and languages. Multiple languages and formats can be installed on the same system, and Asterisk will utilize prompts from languages installed, and will automatically pick the least CPU intensive format that is available on the system (based on codecs in use, in addition to the codec and format modules installed and available).

In addition to the prompts available with Asterisk, you can create your own sets of prompts and utilize them as well. This document will tell you how the prompts available for Asterisk are created so that the prompts you create can be as close and consistent in the quality and volume levels as those shipped with Asterisk.

## Getting the Sounds Tools

The sounds tools are available in the publicly accessible repotools repository. You can check these tools out with Subversion via the following command:

```
# svn co http://svn.asterisk.org/svn/repotools
```

The sound tools are available in the subdirectory sound\_tools/ which contains the following directories:

- audiofilter
- makeg722
- scripts

## About the Sounds Tools

The following sections will describe the sound tools in more detail and explain what they are used for in the sounds package creation process.

### ***audiofilter***

The audiofilter application is used to "tune" the sound files in such a way that they sound good when being used while in a compressed format. The values in the scripts for creating the sound files supplied in repotools is essentially a high-pass filter that drops out audio below 100Hz (or so).

(There is an ITU specification that states for 8KHz audio that is being compressed frequencies below a certain threshold should be removed because they make the resulting compressed audio sound worse than it should.)

The audiofilter application is used by the 'converter' script located in the scripts subdirectory of repotools/sound\_tools. The values being passed to the audiofilter application is as follows:

```
audiofilter -n 0.86916 -1.73829 0.86916 -d 1.00000 -1.74152 0.77536
```

The two options -n and -d are 'numerator' and 'denominator'. Per the author, Jean-Marc Valin, "These values are filter coefficients (-n means numerator, -d is denominator) expressed in the z-transform domain. There represent an elliptic filter that I designed with Octave such that 'the result sounds good'."

### ***makeg722***

The makeg722 application is used by the 'converters' script to generate the G.722 sound files that are shipped with Asterisk. It starts with the RAW sound files and then converts them to G.722.

### ***scripts***

The scripts folder is where all the magic happens. These are the scripts that the Asterisk open source team use to build the packaged audio files for the various formats that are distributed with Asterisk.

- chkcore - used to check that the contents of core-sounds-lang.txt are in sync
- chkextra - same as above, but checks the extra sound files
- mkcore - script used to generate the core sounds packages
- mkextra - script used to generate the extra sounds packages
- mkmoh - script used to generate the music on hold packages
- converters - script used to convert the master files to various formats

# Sound Prompt Searching based on Channel Language

## How Asterisk Searches for Sound Prompts Based on Channel Language

Each channel in Asterisk can be assigned a language by the channel driver. The channel's language code is split, piece by piece (separated by underscores), and used to build paths to look for sound prompts. Asterisk then uses the first file that is found.

This means that if we set the language to en\_GB\_female\_BT, for example, Asterisk would search for files in:

.../sounds/en\_GB\_female\_BT

.../sounds/en\_GB\_female

.../sounds/en\_GB

.../sounds/en

.../sounds

This scheme makes it easy to add new sound prompts for various language variants, while falling back to a more general prompt if there is no prompt recorded in the more specific variant.



# Troubleshooting



Under Construction



This page will be reworked to become a top level page for troubleshooting articles, including a page for overall troubleshooting approach and specific sub-pages on troubleshooting certain sub-systems.

If you're able to get the command-line examples above working, feel free to skip this section. Otherwise, let's look at troubleshooting connections to the Asterisk CLI.

The most common problem that people encounter when learning the Asterisk command-line interface is that sometimes they're not able to connect to the Asterisk service running in the background. For example, let's say that Fred starts the Asterisk service, but then isn't able to connect to it with the CLI:

```
[root@server ~]# service asterisk start
Starting asterisk:                                     [ OK ]
[root@server ~]# asterisk -r
Asterisk version, Copyright (C) 1999 - 2010 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
=====
Unable to connect to remote asterisk (does /var/run/asterisk/asterisk.ctl exist?)
```

What does this mean? It most likely means that Asterisk did not remain running between the time that the service was started and the time Fred tried to connect to the CLI (even if it was only a matter of a few seconds.) This could be caused by a variety of things, but the most common is a broken configuration file.

To diagnose Asterisk start-up problems, we'll start Asterisk in a special mode, known as **console** mode. In this mode, Asterisk does not run as a background service or daemon, but instead runs directly in the console. To start Asterisk in console mode, pass the **-c** parameter to the **asterisk** application. In this case, we also want to turn up the verbosity, so we can see any error messages that might indicate why Asterisk is unable to start.

```
[root@server ~]# asterisk -vvvc
Asterisk version, Copyright (C) 1999 - 2010 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=====
== Parsing '/etc/asterisk/asterisk.conf': == Found
== Parsing '/etc/asterisk/extconfig.conf': == Found
== Parsing '/etc/asterisk/logger.conf': == Found
== Parsing '/etc/asterisk/asterisk.conf': == Found
Asterisk Dynamic Loader Starting:
== Parsing '/etc/asterisk/modules.conf': == Found
...
```

Carefully look for any errors or warnings that are printed to the CLI, and you should have enough information to solve whatever problem is keeping Asterisk from starting up.



## Running Asterisk in Console Mode

We don't recommend you use Asterisk in console mode on a production system, but simply use it for debugging, especially when debugging start-up problems. On production systems, run Asterisk as a background service.

# SIP Retransmissions

## What is the problem with SIP retransmits?

Sometimes you get messages in the console like these:

```
retrans_pkt: Hanging up call XX77yy - no reply to our critical packet.  
retrans_pkt: Cancelling retransmit of OPTIONS
```

The SIP protocol is based on requests and replies. Both sides send requests and wait for replies. Some of these requests are important. In a TCP/IP network many things can happen with IP packets. Firewalls, NAT devices, Session Border Controllers and SIP Proxys are in the signalling path and they will affect the call.

### SIP Call setup - INVITE-200 OK - ACK

To set up a SIP call, there's an INVITE transaction. The SIP software that initiates the call sends an INVITE, then wait to get a reply. When a reply arrives, the caller sends an ACK. This is a three-way handshake that is in place since a phone can ring for a very long time and the protocol needs to make sure that all devices are still on line when call setup is done and media starts to flow.

- The first reply we're waiting for is often a "100 trying". This message means that some type of SIP server has received our request and makes sure that we will get a reply. It could be the other endpoint, but it could also be a SIP proxy or SBC that handles the request on our behalf.
- After that, you often see a response in the 18x class, like "180 ringing" or "183 Session Progress". This typically means that our request has reached at least one endpoint and something is alerting the other end that there's a call coming in.
- Finally, the other side answers and we get a positive reply, "200 OK". This is a positive answer. In that message, we get an address that goes directly to the device that answers. Remember, there could be multiple phones ringing. The address is specified by the Contact: header.
- To confirm that we can reach the phone that answered our call, we now send an ACK to the Contact: address. If this ACK doesn't reach the phone, the call fails. If we can't send an ACK, we can't send anything else, not even a proper hangup. Call signaling will simply fail for the rest of the call and there's no point in keeping it alive.
- If we get an error response to our INVITE, like "Busy" or "Rejected", we send the ACK to the same address as we sent the INVITE, to confirm that we got the response.

In order to make sure that the whole call setup sequence works and that we have a call, a SIP client retransmits messages if there's too much delay between request and expected response. We retransmit a number of times while waiting for the first response. We retransmit the answer to an incoming INVITE while waiting for an ACK. If we get multiple answers, we send an ACK to each of them.

If we don't get the ACK or don't get an answer to our INVITE, even after retransmissions, we will hangup the call with the first error message you see above.

### Other SIP requests

Other SIP requests are only based on request - reply. There's no ACK, no three-way handshake. In Asterisk we mark some of these as CRITICAL - they need to go through for the call to work as expected. Some are non-critical, we don't really care what happens with them, the call will go on happily regardless.

### The qualification process - OPTIONS

If you turn on `qualify=` in `sip.conf` for a device, Asterisk will send an OPTIONS request every minute to the device and check if it replies. Each OPTIONS request is retransmitted a number of times (to handle packet loss) and if we get no reply, the device is considered unreachable. From that moment, we will send a new OPTIONS request (with retransmits) every tenth second.

### Why does this happen?

For some reason signalling doesn't work as expected between your Asterisk server and the other device. There could be many reasons why this happens.

- A NAT device in the signalling path. A misconfigured NAT device is in the signalling path and stops SIP messages.
- A firewall that blocks messages or reroutes them wrongly in an attempt to assist in a too clever way.
- A SIP middlebox (SBC) that rewrites contact: headers so that we can't reach the other side with our reply or the ACK.
- A badly configured SIP proxy that forgets to add record-route headers to make sure that signalling works.
- Packet loss. IP and UDP are unreliable transports. If you loose too many packets the retransmits doesn't help and communication is impossible. If this happens with signaling, media would be unusable anyway.

### What can I do?

Turn on SIP debug, try to understand the signalling that happens and see if you're missing the reply to the INVITE or if the ACK gets lost. When you know

what happens, you've taken the first step to track down the problem. See the list above and investigate your network.

For NAT and Firewall problems, there are many documents to help you. Start with reading sip.conf.sample that is part of your Asterisk distribution.

The SIP signalling standard, including retransmissions and timers for these, is well documented in the IETF RFC 3261.

Good luck sorting out your SIP issues!

/Olle E. Johansson

– oej (at) edvina.net, Sweden, 2008-07-22

– <http://www.voip-forum.com>

# Troubleshooting Asterisk Module Loading

## Symptoms

- Specific Asterisk functionality is no longer available or completely non-functioning, but other Asterisk features and modules continue to function.
- Specific Asterisk CLI commands are no longer available.

Example :

```
No such command 'sip show peers'
```

We can presume that something is wrong with **chan\_sip** module since we know it provides the 'sip' commands and sub-commands.

## Problem

Asterisk has started successfully and the module providing the missing functionality either didn't load at all, or it loaded but isn't running.

The reason for the failure to load or run is typically invalid configuration or a failure to parse the configuration for the module.

### On this Page

- [Symptoms](#)
- [Problem](#)
- [Solution](#)
- [Troubleshooting](#)
  - [Check Module Loaded and Running States](#)
  - [Make sure Asterisk is configured to load the module](#)
  - [Check For Module Loading Issues on Asterisk Startup](#)

## Solution

Identify the state of the module. If the module is loaded but not running, or not loaded at all, then resolve file format, configuration syntax issues or unwanted modules.conf configuration for the specific module. Restart Asterisk.

## Troubleshooting

### Check Module Loaded and Running States

From the [Asterisk CLI](#) you can use the 'module show' commands to identify the state of a module.

Previous to Asterisk 12, you could only see if the module is loaded. However it may not actually be running (usable).

```
*CLI> module show like chan_sip.so
Module      Description      Use Count
chan_sip.so Session Initiation Protocol (SIP)  0
1 modules loaded
```

In Asterisk 12 and beyond you can quickly see if a module is loaded and whether it is running or not.

```
*CLI> module show like chan_sip.so
Module      Description      Use Count  Status
chan_sip.so Session Initiation Protocol (SIP)  0          Not Running
1 modules loaded
```

### Make sure Asterisk is configured to load the module

Modules.conf is a [core configuration](#) file that includes parameters affecting module loading and loading order. There are a few items to check.

Verify that **autoload=yes** is enabled if you are intending to load modules from the Asterisk modules directory automatically.

Verify that there is **not** a '**noload**' line for the module that is failing to load. That is, if we had a line as follows:

```
noload => chan_sip.so
```

That would tell Asterisk to not load chan\_sip.so.

If you are not using **autoload**, then be sure you have a **load** line for the module you desire to load.

```
load => chan_sip.so
```

## Check For Module Loading Issues on Asterisk Startup

Now that we know the suspect module should be loading, we can look at some logs that may tell us what is happening.

### Stop Asterisk

Be sure Asterisk is stopped to avoid issues with making the logs confusing.

```
asterisk -rx "core stop now"
```

or

```
service asterisk stop
```

### Enable logging channels

You can read in detail about [Logging facilities](#) on the wiki. In short, for this example, make sure you have the following lines uncommented in your `logger.conf` file.

```
[logfiles]
full => notice,warning,error,debug,verbose
```

### Clear out old logs

You don't want to mistakenly look at an older log where Asterisk was loading appropriately.

Remove the most recent log file, or else move it somewhere you want to keep it.

```
# rm /var/log/asterisk/full
```

### Start Asterisk with appropriate log levels

It is important to start Asterisk with [log levels](#) that will provide us enough information.

```
# asterisk -cvvvvvddd
```

You'll see a lot of information output in the terminal as Asterisk loads.

### Stop Asterisk after it has finished loading

After the output calms down and Asterisk has finished loading, go ahead and stop Asterisk. The logs should have already been recorded.

```
*CLI> core stop now
```

### Search logs for lines related to suspect module

Search the [log file](#) using keywords based on the specific module that appeared to be failing to load or run.

```
/var/log/asterisk# grep -i chan_sip full
[Oct  9 14:54:43] VERBOSE[21809] chan_sip.c: SIP channel loading...
[Oct  9 14:54:43] ERROR[21809] chan_sip.c: Contents of sip.conf are invalid and cannot be parsed

/var/log/asterisk# grep -i sip.conf full
[Oct  9 14:54:43] DEBUG[21809] config.c: Parsing /etc/asterisk/sip.conf
[Oct  9 14:54:43] VERBOSE[21809] config.c: == Parsing '/etc/asterisk/sip.conf': Found
[Oct  9 14:54:43] WARNING[21809] config.c: parse error: No category context for line 1 of /etc/asterisk/sip.conf
[Oct  9 14:54:43] ERROR[21809] chan_sip.c: Contents of sip.conf are invalid and cannot be parsed
[Oct  9 14:54:55] DEBUG[21809] config.c: Parsing /etc/asterisk/sip.conf
[Oct  9 14:54:55] VERBOSE[21809] config.c: == Parsing '/etc/asterisk/sip.conf': Found
[Oct  9 14:54:55] WARNING[21809] config.c: parse error: No category context for line 1 of /etc/asterisk/sip.conf
```

Based on the lines found, you can then use an editor like VIM to view the full log and jump to where the relevant messages are.

```
[Oct 9 14:54:43] VERBOSE[21809] chan_sip.c: SIP channel loading...
[Oct 9 14:54:43] DEBUG[21809] config.c: Parsing /etc/asterisk/sip.conf
[Oct 9 14:54:43] VERBOSE[21809] config.c: == Parsing '/etc/asterisk/sip.conf': Found
[Oct 9 14:54:43] WARNING[21809] config.c: parse error: No category context for line 1 of /etc/asterisk/sip.conf
[Oct 9 14:54:43] ERROR[21809] chan_sip.c: Contents of sip.conf are invalid and cannot be parsed
```

In this case, not much more is revealed past what we saw with `grep`. You can see that Asterisk tries to load and run `chan_sip`, it fails because the contents of `sip.conf` are invalid and cannot be parsed. The most specific clue is the WARNING:

```
WARNING[21809] config.c: parse error: No category context for line 1 of /etc/asterisk/sip.conf
```

### Edit the related config file to resolve the issue

If we look at line 1 of `sip.conf` we'll spot the root problem.

```
general]
context=public
allowoverlap=no
```

For our example, a square bracket is missing from the context definition! Fix this issue, restart Asterisk and things should work assuming I don't have any other syntax errors.

# Unable to connect to remote Asterisk

## Overview

When first learning Asterisk some users will find they are unable to connect to the Asterisk service.

You may see the below message after running some variation of `asterisk -r`

```
Unable to connect to remote asterisk (does /var/run/asterisk/asterisk.ctl exist?)
```

First, let's break down the message.

- "Unable to connect to remote asterisk"

This means you are attempting to connect to Asterisk with a [remote console](#). That is, you are using "asterisk -r" as opposed to "asterisk -c".

- "does /var/run/asterisk/asterisk.ctl exist?"

This is letting you know specifically what file is potentially missing or unable to be accessed.

Now let's find out what asterisk.ctl is and investigate potential causes of this error.

## The asterisk.ctl file

asterisk.ctl is a UNIX Domain Socket file. It is used to pass commands to an Asterisk process and it is how the Asterisk console ("asterisk r" or any variation) communicates with the back-end Asterisk process.

## The /var/run/asterisk/ directory

This directory is the default Asterisk run directory. Asterisk will create it the first time it is run. This location is defined in the [Asterisk Main Configuration File](#). As is explained in the [Directory and File Structure](#) section; when Asterisk is [running](#), you'll see two files here, **asterisk.ctl** and **asterisk.pid**. These are the control socket and the PID(Process ID) files for Asterisk.

## Potential Causes and Solutions

### Cause 1: asterisk.ctl does exist. Your user does not have write permission for this file.

Solution:

1. To verify, check the permissions of the asterisk.ctl file and also check what user you are currently running as.
2. Switch to the correct user that has access to the /var/run/asterisk/ directory and asterisk.ctl file, or provide your current user with the correct permissions.

### Cause 2: Permissions issue. Asterisk does not have write access to /var/run/asterisk or your user doesn't have write access to asterisk.ctl.

Solution:

1. If /var/run/asterisk does not exist then create it and assign permission to it such that the user that runs Asterisk will have write and read access.
2. If it already exists, simply verify and assign the correct permissions to the directory. You probably want to double-check what user runs Asterisk.

### Cause 3: asterisk.ctl does not exist because Asterisk isn't running. When Asterisk is started it may run briefly and then quickly halt due to an error.

Solution:

You need to find out what error is causing Asterisk to halt and resolve it.

- The quick way is to run Asterisk in root/core console mode and watch for the last messages Asterisk prints out before halting.

```
asterisk -cvvvvv
```

This will start Asterisk in console mode with level 5 verbosity. That should give you plenty to look at.

- Another way is to setup Asterisk [Logging](#) to log what you want to see to a file. You'll need to read up on Asterisk's [Logging Configuration](#)
- Asterisk could halt for a variety of reasons. The last messages you see before Asterisk halts will give you a clue. Then you can Google from there or ask the user community.

**Cause 4: SELinux enabled and not properly configured for Asterisk. SELinux not allowing asterisk.ctl to be created.**

Solution:

Configuring SELinux is outside the scope of this article.

- Consult an experienced SELinux user or SELinux documentation on how to configure it properly.
- Disable SELinux if you don't require it (Not recommended)



# IPv6 Support

## Overview

Since Asterisk 12, IPv6 is supported by the most commonly used components of Asterisk which support IP based communication. This includes the latest SIP channel driver `chan_pjsip` as well as the older `chan_sip`. IPv6 support may be spotty before Asterisk 12. For sufficient IPv6 support it is recommended that you upgrade to Asterisk 13 or greater.

## Configuration

Fortunately configuration is easy and most things will simply work. For channel technologies such as `chan_pjsip`, `chan_sip` or `chan_iax2` the IPv6 support must be configured in the channel's configuration file. (i.e. `pjsip.conf`, `sip.conf` or `iax.conf`). In each configuration file there are typically options referring to a general bind address or specific TCP or UDP bind addresses. Other than configuring those options to bind to IPv6 interfaces there should be few other options needed. The documentation or configuration samples for each driver should make this clear.

Links to specific instructions:

- [Configuring res\\_pjsip for IPv6](#)
- [Configuring chan\\_sip for IPv6](#)
- [Configuring IAX for IPv6](#)
- [Configuring ACLs for IPv6](#)

# PSTN Connectivity

# Advice of Charge

Written by: David Vossel  
Initial version: 04-19-2010  
Email: dvossel@digium.com

This document is designed to give an overview of how to configure and generate Advice of Charge along with a detailed explanation of how each option works.

## Read This First

PLEASE REPORT ANY ISSUES ENCOUNTERED WHILE USING AOC. This feature has had very little community feedback so far. If you are using this feature please share with us any problems you are having and any improvements that could make this feature more useful. Thank you!

## Terminology

**AOC:** Advice of Charge

**AOC-S:** Advice of Charge message sent at the beginning of a call during call setup. This message contains a list of rates associated with the call.

**AOC-D:** Advice of Charge message sent during the call. This message is typically used to update the endpoint with the current call charge.

**AOC-E:** Advice of Charge message sent at the end of a call. This message is used to indicate to the endpoint the final call charge.

**AMI:** Asterisk Manager Interface. This interface is used to generate AOC messages and listen for AOC events.

## AOC in chan\_dahdi

### LibPRI Support:

ETSI, or euroisdn, is the only switchtype that LibPRI currently supports for AOC.

### Enable AOC Pass-through in chan\_dahdi

To enable AOC pass-through between the ISDN and Asterisk use the 'aoc\_enable' config option. This option allows for any combination of AOC-S, AOC-D, and AOC-E to be enabled or disabled.

For example:

```
aoc_enable=s,d,e ; enables pass-through of AOC-S, AOC-D, and AOC-E  
  
aoc_enable=s,d   ; enables pass-through of AOC-S and AOC-D. Rejects  
; AOC-E and AOC-E request messages
```

Since AOC messages are often transported on facility messages, the 'facilityenable' option must be enabled as well to fully support AOC pass-through.

### Handling AOC-E in chan\_dahdi

Whenever a dahdi channel receives an AOC-E message from Asterisk, it stores that message to deliver it at the appropriate time during call termination. This means that if two AOC-E messages are received on the same call, the last one will override the first one and only one AOC-E message will be sent during call termination.

There are some tricky situations involving the final AOC-E message. During a bridged call, if the endpoint receiving the AOC messages terminates the call before the endpoint delivering the AOC does, the final AOC-E message sent by the sending side during termination will never make it to the receiving end because Asterisk will have already torn down that channel.

This is where the chan\_dahdi.conf 'aoce\_delayhangup' option comes into play.

By enabling 'aoce\_delayhangup', anytime a hangup is initiated by the ISDN side of an Asterisk channel, instead of hanging up the channel, the channel sends a unique internal AOC-E termination request to its bridge channel. This indicates it is about to hangup and wishes to receive the final AOC-E message from the bridged channel before completely tearing down. If the bridged channel knows what to do with this AOC-E termination request, it will do whatever is necessary to indicate to its endpoint that the call is being terminated without actually hanging up the Asterisk channel. This allows the final AOC-E message to come in and be sent across the bridge while both channels are still up. If the channel delaying its hangup for the final AOC-E message times out, the call will be torn down just as it normally would. In chan\_dahdi the timeout period is 1/2 the T305 timer which by default is 15 seconds.

'aoce\_delayhangup' currently only works when both bridged channels are dahdi\_channels. If a SIP channel receives an AOC-E termination request, it just responds by immediately hanging up the channel. Using this option when bridged to any channel technology besides SIP or DAHDI will result in the 15 second timeout period before tearing down the call completely.

## Requesting AOC services

AOC can be requested on a call by call basis using the DAHDI dialstring option, A(). The A() option takes in 's', 'd', and 'e' parameters which represent the three types of AOC messages, AOC-S, AOC-D, and AOC-E. By using this option Asterisk will indicate to the endpoint during call setup that it wishes to receive the specified forms of AOC during the call.

Example Usage in extensions.conf

```
exten => 1111,1,Dial(DAHDI/g1/1112/A(s,d,e) ; requests AOC-S, AOC-D, and AOC-E on
; call setup
exten => 1111,1,Dial(DAHDI/g1/1112/A(d,e) ; requests only AOC-D, and AOC-E on
; call setup
```

### AOC in chan\_sip

Asterisk supports a very basic way of sending AOC on a SIP channel to Snom phones using an AOC specification designed by Snom. This support is limited to the sending of AOC-D and AOC-E pass-through messages. No support for AOC-E on call termination is present, so if the Snom endpoint receiving the AOC messages from Asterisk terminates the call, the channel will be torn down before the phone can receive the final AOC-E message.

To enable passthrough of AOC messages via the snom specification, use the 'snom\_aoc\_enabled' option in sip.conf.

### Generate AOC Messages via AMI

Asterisk supports a way to generate AOC messages on a channel via the AMI action AOCMessage. At the moment the AOCMessage action is limited to AOC-D and AOC-E message generation. There are some limitations involved with delivering the final AOC-E message as well. The AOCMessage action has its own detailed parameter documentation so this discussion will focus on higher level use. When generating AOC messages on a Dahdi channel first make sure the appropriate chan\_dahdi.conf options are enabled. Without enabling 'aoc\_enable' correctly for pass-through the AOC messages will never make it out the pri. The same goes with SIP, the 'snom\_aoc\_enabled' option must be configured before messages can successfully be set to the endpoint.

### AOC-D Message Generation

AOC-D message generation can happen anytime throughout the call. This message type is very straight forward.

Example: AOCMessage action generating AOC-D currency message with Success response.

```
Action: AOCMessage
Channel: DAHDI/i1/1111-1
MsgType: d
ChargeType: Currency
CurrencyAmount: 16
CurrencyName: USD
CurrencyMultiplier: OneThousandth
AOCBillingId: Normal
ActionID: 1234

Response: Success
ActionID: 1234
Message: AOC Message successfully queued on channel
```

### AOC-E Message Generation

AOC-E messages are sent during call termination and represent the final charge total for the call. Since Asterisk call termination results in the channel being destroyed, it is currently not possible for the AOCMessage AMI action to be used to send the final AOC-E message on call hangup. There is however a work around for this issue that can be used for Dahdi channels. By default chan\_dahdi saves any AOC-E message it receives from Asterisk during a call and waits to deliver that message during call termination. If multiple AOC-E messages are received from Asterisk on the same Dahdi channel, only the last message received is stored for delivery. This means that each new AOC-E message received on the channel overrides the previous one. Knowing this the final AOC-E message can be continually updated on a Dahdi channel until call termination occurs allowing the last update to be sent on hangup. This method is only as accurate as the intervals in which it is updated, but allows some form of AOC-E to be generated.

Example: AOCMessage action generating AOC-E unit message with Success response.

```
Action: AOCMessage
Channel: DAHDI/i1/1111-1
MsgType: e
ChargeType: Unit
UnitAmount(0): 111
UnitType(0): 6
UnitAmount(1): 222
UnitType(1): 5
UnitAmount(2): 333
UnitType(3): 4
UnitAmount(4): 444
AOCBillingId: Normal
ActionID: 1234

Response: Success
ActionID: 1234
Message: AOC Message successfully queued on channel
```

## Caller ID in India

India finds itself in a unique situation (hopefully). It has several telephone line providers, and they are not all using the same CID signaling; and the CID signalling is not like other countries.

In order to help those in India quickly find to the CID signaling system that their carrier uses (or range of them), and get the configs right with a minimal amount of experimentation, this file is provided. Not all carriers are covered, and not all mentioned below are complete. Those with updates to this table should post the new information on bug [6683](#) of the asterisk bug tracker.

---

**Provider:** Bharti (is this BSNL?)

**Config:**

```
cidstart=polarity_in
cidsignalling=dtmf
```

**Results:** ? (this should work), but needs to be tested?

**Tested by:** ?

---

**Provider:** VSNL

**Config:**

```
null
```

**Results:** ?

**Tested by:** ?

---

**Provider:** BSNL

**Config:**

```
cid_start=ring
cid_signalling=dtmf
```

**Results:** ?

**Tested by:** (abhi)

---

**Provider:** MTNL, old BSNL

**Config:**

```
cidsignalling = v23
cidstart=ring
```

**Results:** works

**Tested by:** (enterux)

---

**Provider:** MTNL (Delhi)

**Config:**

```
cidsignalling = v23
cidstart = ring
```

or:

```
cidsignalling = dtmf
cidstart = polarity_IN
```

or:

```
cidsignalling = dtmf  
cidstart = polarity
```

**Results:** fails

**Tested by:** brealer

---

**Provider:** TATA

**Config:**

```
cidsignalling = dtmf  
cidstart=polarity_IN
```

**Results:** works

**Tested by:** brealer

---

Asterisk still doesn't work with some of the CID scenarios in India. If you are in India, and not able to make CID work with any of the permutations of cidsignalling and cidstart, it could be that this particular situation is not covered by Asterisk. A good course of action would be to get in touch with the provider, and find out from them exactly how their CID signalling works. Describe this to us, and perhaps someone will be able to extend the code to cover their signaling.

# Signaling System Number 7

## Where to get LibSS7?

Currently, all released branches of Asterisk including trunk use the released versions of libss7.

The latest compatible libss7 code can be obtained from the [1.0 SVN branch](#)

As of 7/2013, the libss7 trunk (<http://svn.asterisk.org/svn/libss7/trunk>) is currently only usable with this Asterisk branch ([http://svn.asterisk.org/svn/asterisk/teamm/rmudgett/ss7\\_27\\_knk](http://svn.asterisk.org/svn/asterisk/teamm/rmudgett/ss7_27_knk)) based off of Asterisk trunk.

### Tested Switches:

- Siemens EWSD - (ITU style) MTP2 and MTP3 comes up, ISUP inbound and outbound calls work as well.
- DTI DXC 4K - (ANSI style) 56kbps link, MTP2 and MTP3 come up, ISUP inbound and outbound calls work as well.
- Huawei M800 - (ITU style) MTP2 and MTP3 comes up, ISUP National, International inbound and outbound calls work as well, CallerID presentation&screening work.

and MORE~!

### Thanks:

Mark Spencer, for writing Asterisk and libpri and being such a great friend and boss.

Luciano Ramos, for donating a link in getting the first "real" ITU switch working.

Collin Rose and John Lodden, John for introducing me to Collin, and Collin for the first "real" ANSI link and for holding my hand through the remaining changes that had to be done for ANSI switches.

### To Use:

In order to use libss7, you must get at least the following versions of DAHDI and Asterisk:

DAHDI: 2.0.x

libss7: trunk (currently, there **only** is a trunk release).

Asterisk: 1.6.x

You must then do a `make; make install` in each of the directories that you installed in the given order (DAHDI first, libss7 second, and Asterisk last).



In order to check out the code, you must have the subversion client installed. This is how to check them out from the public subversion server.

These are the commands you would type to install them:

```
`svn co http://svn.digium.com/svn/dahdi/linux/trunk dahdi-trunk`  
`cd dahdi-trunk`  
`make; make install`  
  
`svn co http://svn.digium.com/svn/dahdi/tools/trunk dahdi-tools`  
`cd dahdi-tools`  
`./configure; make; make install`  
  
`svn co http://svn.digium.com/svn/libss7/trunk libss7-trunk`  
`cd libss7-trunk`  
`make; make install`  
  
`svn co http://svn.digium.com/svn/asterisk/trunk asterisk-trunk`  
`cd asterisk-trunk`  
`./configure; make; make install;`
```

This should build DAHDI, libss7, and Asterisk with SS7 support.

In the past, there was a special asterisk-ss7 branch to use which contained the SS7 code. That code has been merged back into the trunk version of Asterisk, and the old asterisk-ss7 branch has been deprecated and removed. If you are still using the asterisk-ss7 branch, it will not work against the current version of libss7, and you should switch to asterisk-trunk instead.

### Configuration:

In `/etc/dahdi/system.conf`, your signalling channel(s) should be a "dchan" and your bearers should be set as "bchan".

The sample `chan_dahdi.conf` contains sample configuration for setting up an E1 link.



In brief, here is a simple ss7 linkset setup:

```
signalling = ss7
ss7type = itu ; or ansi if you are using an ANSI link

linkset = 1 ; Pick a number for your linkset identifier in chan_dahdi.conf

pointcode = 28 ; The decimal form of your point code. If you are using an
; ANSI linkset, you can use the xxx-xxx-xxx notation for
; specifying your link
adjpointcode = 2 ; The point code of the switch adjacent to your linkset

defaultdpc = 3 ; The point code of the switch you want to send your ISUP
; traffic to. A lot of the time, this is the same as your
; adjpointcode.

; Now we configure our Bearer channels (CICs)

cicbeginswith = 1 ; Number to start counting the CICs from. So if DAHDI/1 to
; DAHDI/15 are CICs 1-15, you would set this to 1 before you
; declare channel=1-15

channel=1-15 ; Use DAHDI/1-15 and assign them to CICs 1-15

cicbeginswith = 17 ; Now for DAHDI/17 to DAHDI/31, they are CICs 17-31 so we initialize
; cicbeginswith to 17 before we declare those channels

channel = 17-31 ; This assigns CICs 17-31 to channels 17-31

sigchan = 16 ; This is where you declare which DAHDI channel is your signalling
; channel. In our case it is DAHDI/16. You can add redundant
; signalling channels by adding additional sigchan= lines.

; If we want an alternate redundant signalling channel add this

sigchan = 48 ; This would put two signalling channels in our linkset, one at
; DAHDI/16 and one at DAHDI/48 which both would be used to send/receive
; ISUP traffic.

; End of chan_dahdi.conf
```

This is how a basic linkset is setup. For more detailed chan\_dahdi.conf SS7 config information as well as other options available for that file, see the default chan\_dahdi.conf that comes with the samples in asterisk. If you would like, you can do a `make samples` in your asterisk-trunk directory and it will install a sample chan\_dahdi.conf for you that contains more information about SS7 setup.

For more information, you can ask questions of the community on the asterisk-ss7 or asterisk-dev mailing lists.

# Secure Calling

Top-level page for articles about securing VoIP calls using encryption.

## Secure Calling Specifics

Asterisk supports a channel-agnostic method for handling secure call requirements. Since there is no single meaning of what constitutes a "secure call," Asterisk allows the administrator the control to define "secure" for themselves via the dialplan and channel-specific configuration files.

### Channel-specific configuration

Currently the IAX2 and SIP channels support the call security features in Asterisk. Both channel-specific configuration files (`iax2.conf` and `sip.conf`) support the `encryption=yes` setting. For IAX2, this setting causes Asterisk to offer encryption when placing or receiving a call. To force encryption with IAX2, the `forceencrypt=yes` option is required. Due to limitations of SDP, `encryption=yes` in `sip.conf` results in a call with only a secure media offer, therefore `forceencrypt=yes` would be redundant in `sip.conf`.

If a peer is defined as requiring encryption but the endpoint does not support it, the call will fail with a HANGUPCAUSE of 58 (bearer capability does not exist).

### Security-based dialplan branching

Each channel that supports secure signaling or media can implement a CHANNEL read callback function that specifies whether or not that channel meets the specified criteria. Currently, `chan_iax2` and `chan_sip` implement these callbacks. Channels that do not support secure media or signaling will return an empty string when queried. For example, to only allow an inbound call that has both secure signaling and media, see the following example.

```
exten => 123,1,GotoIf("${CHANNEL(secure_signaling)}" = "1"]?:fail)
exten => 123,n,GotoIf("${CHANNEL(secure_media)}" = "1"]?:fail)
exten => 123,n,Dial(SIP/123)
exten => 123,n,Hangup
exten => 123,n(fail),Playback(vm-goodbye)
exten => 123,n,Hangup
```

### Forcing bridged channels to be secure

Administrators can force outbound channels that are to be bridged to a calling channel to conform to secure media and signaling policies. For example, to first make a call attempt that has both secure signaling and media, but gracefully fall back to non-secure signaling and media see the following example:

```
exten => 123,1,NoOp(We got a call)
exten => 123,n,Set(CHANNEL(secure_bridge_signaling)=1)
exten => 123,n,Set(CHANNEL(secure_bridge_media)=1)
exten => 123,n,Dial(SIP/somebody)
exten => 123,n,NoOp(HANGUPCAUSE=${HANGUPCAUSE})
exten => 123,n,GotoIf("${HANGUPCAUSE}"="58"]?encrypt_fail)
exten => 123,n,Hangup ; notify user that retrying via insecure channel (user-provided
prompt)
exten => 123,n(encrypt_fail),Playback(secure-call-fail-retry)
exten => 123,n,Set(CHANNEL(secure_bridge_signaling)=0)
exten => 123,n,Set(CHANNEL(secure_bridge_media)=0)
exten => 123,n,Dial(SIP/somebody)
exten => 123,n,Hangup
```

# Secure Calling Tutorial

- [Overview](#)
- [Part 1 \(TLS\)](#)
  - [Keys](#)
  - [Asterisk chan\\_pjsip configuration](#)
  - [Asterisk chan\\_sip configuration](#)
  - [Configuring a TLS-enabled SIP client to talk to Asterisk](#)
  - [Problems with server verification](#)
- [Part 2 \(SRTP\)](#)

## Overview



This tutorial makes use of SRTP and TLS. SRTP support was added in Asterisk 1.8, TLS was added in 1.6.

So you'd like to make some secure calls.

Here's how to do it, using [Blink](#), a SIP soft client for Mac OS X, Windows, and Linux. You can find some [brief instructions for installing Blink on Ubuntu](#) on the wiki.

These instructions assume that you're running as the root user (sudo su -).

## Part 1 (TLS)

[Transport Layer Security \(TLS\)](#) provides encryption for call signaling. It's a practical way to prevent people who aren't Asterisk from knowing who you're calling. Setting up TLS between Asterisk and a SIP client involves creating key files, modifying Asterisk's SIP configuration to enable TLS, creating a SIP peer that's capable of TLS, and modifying the SIP client to connect to Asterisk over TLS.

### Keys

First, let's make a place for our keys.

```
mkdir /etc/asterisk/keys
```

Next, use the "ast\_tls\_cert" script in the "contrib/scripts" Asterisk source directory to make a self-signed certificate authority and an Asterisk certificate.

```
./ast_tls_cert -C pbx.mycompany.com -O "My Super Company" -d /etc/asterisk/keys
```

- The "-C" option is used to define our host - DNS name or our IP address.
- The "-O" option defines our organizational name.
- The "-d" option is the output directory of the keys.

1. You'll be asked to enter a pass phrase for /etc/asterisk/keys/ca.key, put in something that you'll remember for later.
2. This will create the /etc/asterisk/keys/ca.crt file.
3. You'll be asked to enter the pass phrase again, and then the /etc/asterisk/keys/asterisk.key file will be created.
4. The /etc/asterisk/keys/asterisk.crt file will be automatically generated.
5. You'll be asked to enter the pass phrase a third time, and the /etc/asterisk/keys/asterisk.pem will be created, a combination of the asterisk.key and asterisk.crt files.

Next, we generate a client certificate for our SIP device.

```
./ast_tls_cert -m client -c /etc/asterisk/keys/ca.crt -k /etc/asterisk/keys/ca.key -C phone1.mycompany.com -O "My Super Company" -d /etc/asterisk/keys -o malcolm
```

- The "-m client" option tells the script that we want a client certificate, not a server certificate.
- The "-c /etc/asterisk/keys/ca.crt" option specifies which Certificate Authority (ourselves) that we're using.
- The "-k /etc/asterisk/keys/ca.key" provides the key for the above-defined Certificate Authority.
- The "-C" option, since we're defining a client this time, is used to define the hostname or IP address of our SIP phone.
- The "-O" option defines our organizational name.
- The "-d" option is the output directory of the keys."
- The "-o" option is the name of the key we're outputting.

1. You'll be asked to enter the pass phrase from before to unlock /etc/asterisk/keys/ca.key.

Now, let's check the keys directory to see if all of the files we've built are there. You should have:

```
asterisk.crt
asterisk.csr
asterisk.key
asterisk.pem
malcolm.crt
malcolm.csr
malcolm.key
malcolm.pem
ca.cfg
ca.crt
ca.key
tmp.cfg
```

Next, copy the malcolm.pem and ca.crt files to the computer running the Blink soft client.



#### .p12 Client Certificates

If your client requires a .p12 certificate file instead, you can generate that using openssl like:

```
# openssl pkcs12 -export -out MySuperClientCert.p12 -inkey ca.key -in ca.crt
-certfile asterisk.crt
```

## Asterisk chan\_pjsip configuration

Now, let's configure Asterisk's PJSIP channel driver to use TLS.

In the **pjsip.conf** configuration file, you'll need to enable a TLS-capable transport. An example of one would resemble:

```
[transport-tls]
type=transport
protocol=tls
bind=0.0.0.0:5061
cert_file=/etc/asterisk/keys/asterisk.crt
priv_key_file=/etc/asterisk/keys/asterisk.key
method=tlsv1
```

Note the **protocol**, **cert\_file**, **priv\_key\_file**, and **method** options. Here, we're using the TLS protocol, we're specifying the keys that we generated earlier for **cert\_file** and **priv\_key\_file** and we're setting the **method** to TLSv1.

Next, you'll need to configure a TLS-capable endpoint. An example of one would resemble:

```
[malcolm]
type=aor
max_contacts=1
remove_existing=yes

[malcolm]
type=auth
auth_type=userpass
username=malcolm
password=useabetterpasswordplease

[malcolm]
type=endpoint
aors=malcolm
auth=malcolm
context=local
disallow=all
allow=g722
dtmf_mode=rfc4733
media_encryption=sdes
```

Note the **media\_encryption** option for the endpoint. In this case, we've configured an endpoint that will be using SDES encryption for RTP.

You might be tempted to add a **transport=transport-tls** to the endpoint but in pjproject versions at least as late as 2.4.5, this will cause issues like **Connection refused** in a few situations. Let pjproject do the transport selection on its own. If you still see issues, set **rewrite\_contact = yes** in the endpoint configuration.

## Asterisk chan\_sip configuration

Or, if you are using can\_sip, you can use the following to assist.

In the **sip.conf** configuration file, set the following:

```
tlsenable=yes
tlsbindaddr=0.0.0.0
tls-certfile=/etc/asterisk/keys/asterisk.pem
tls-cafile=/etc/asterisk/keys/ca.crt
tlscipher=ALL
tlsclientmethod=tlsv1 ;none of the others seem to work with Blink as the client
```

Here, we're enabling TLS support.

We're binding it to our local IPv4 wildcard (the port defaults to 5061 for TLS).

We've set the TLS certificate file to the one we created above.

We've set the Certificate Authority to the one we created above.

TLS Ciphers have been set to ALL, since it's the most permissive.

And we've set the TLS client method to TLSv1, since that's the preferred one for RFCs and for most clients.

Next, you'll need to configure a SIP peer within Asterisk to use TLS as a transport type. Here's an example:

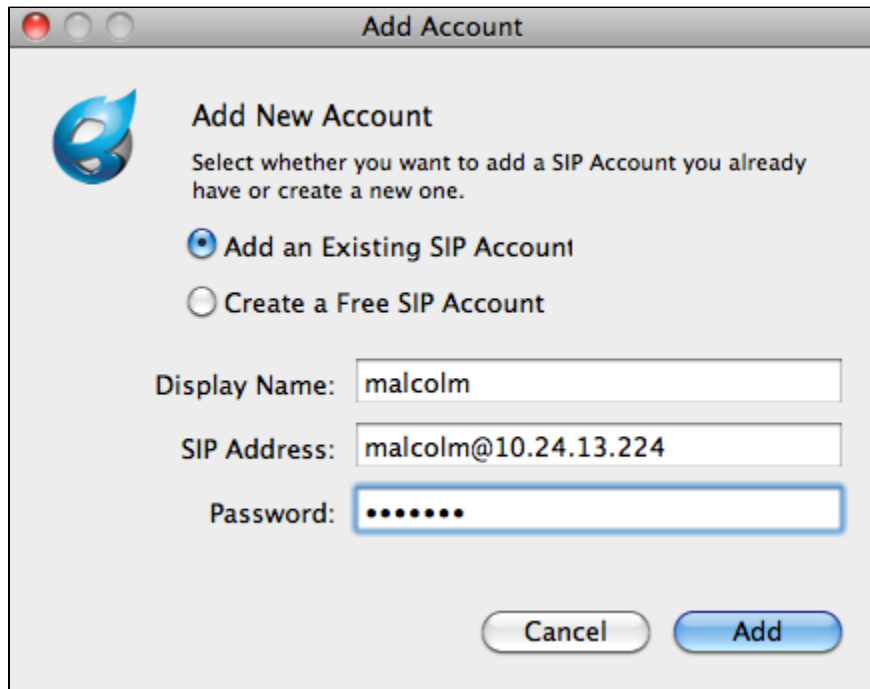
```
[malcolm]
type=peer
secret=malcolm ;note that this is NOT a secure password
host=dynamic
context=local
dtmfmode=rfc2833
disallow=all
allow=g722
transport=tls
```

Notice the **transport** option. The Asterisk SIP channel driver supports three types: udp, tcp and tls. Since we're configuring for TLS, we'll set that. It's also possible to list several supported transport types for the peer by separating them with commas.

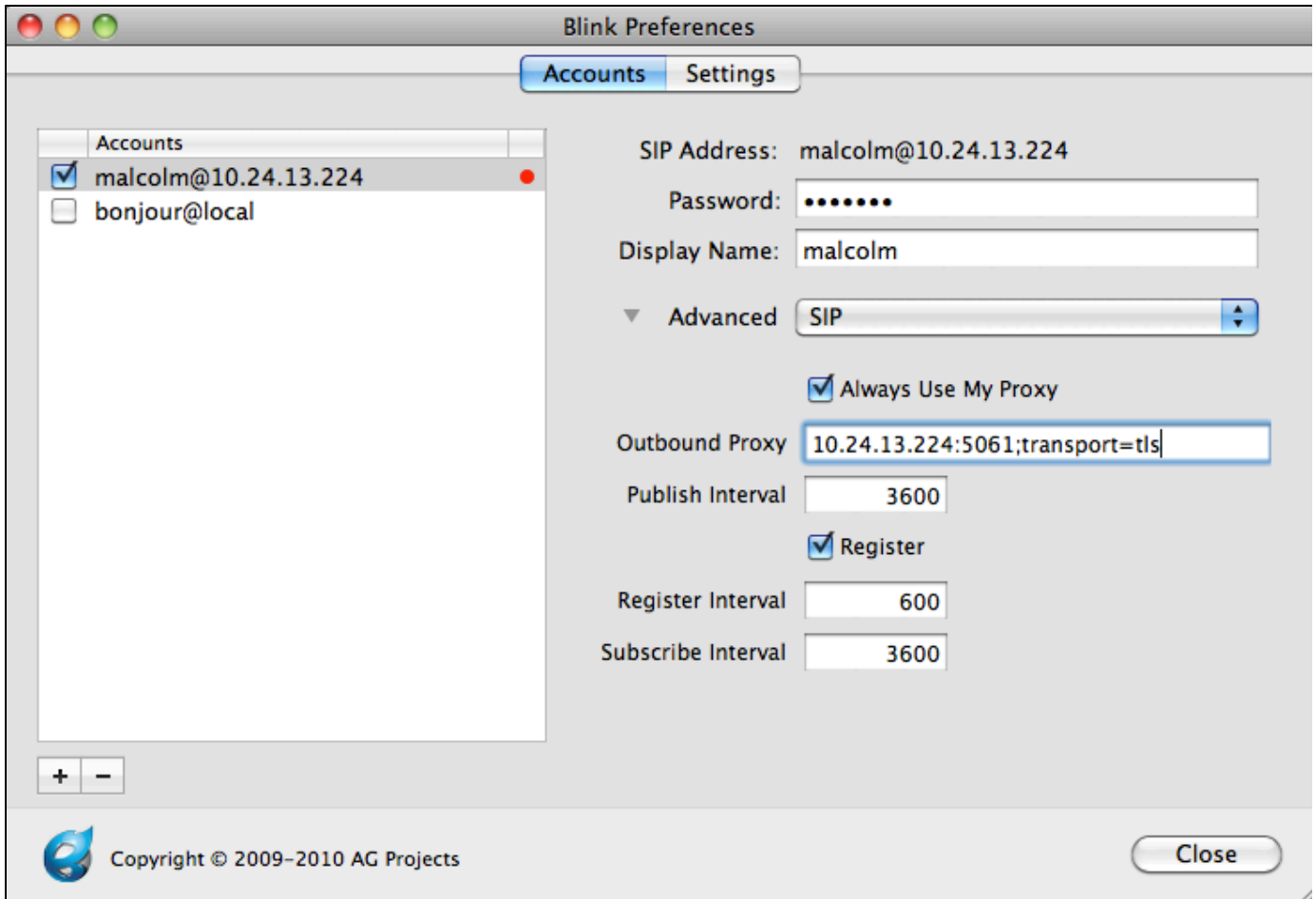
## Configuring a TLS-enabled SIP client to talk to Asterisk

Next, we'll configure Blink.

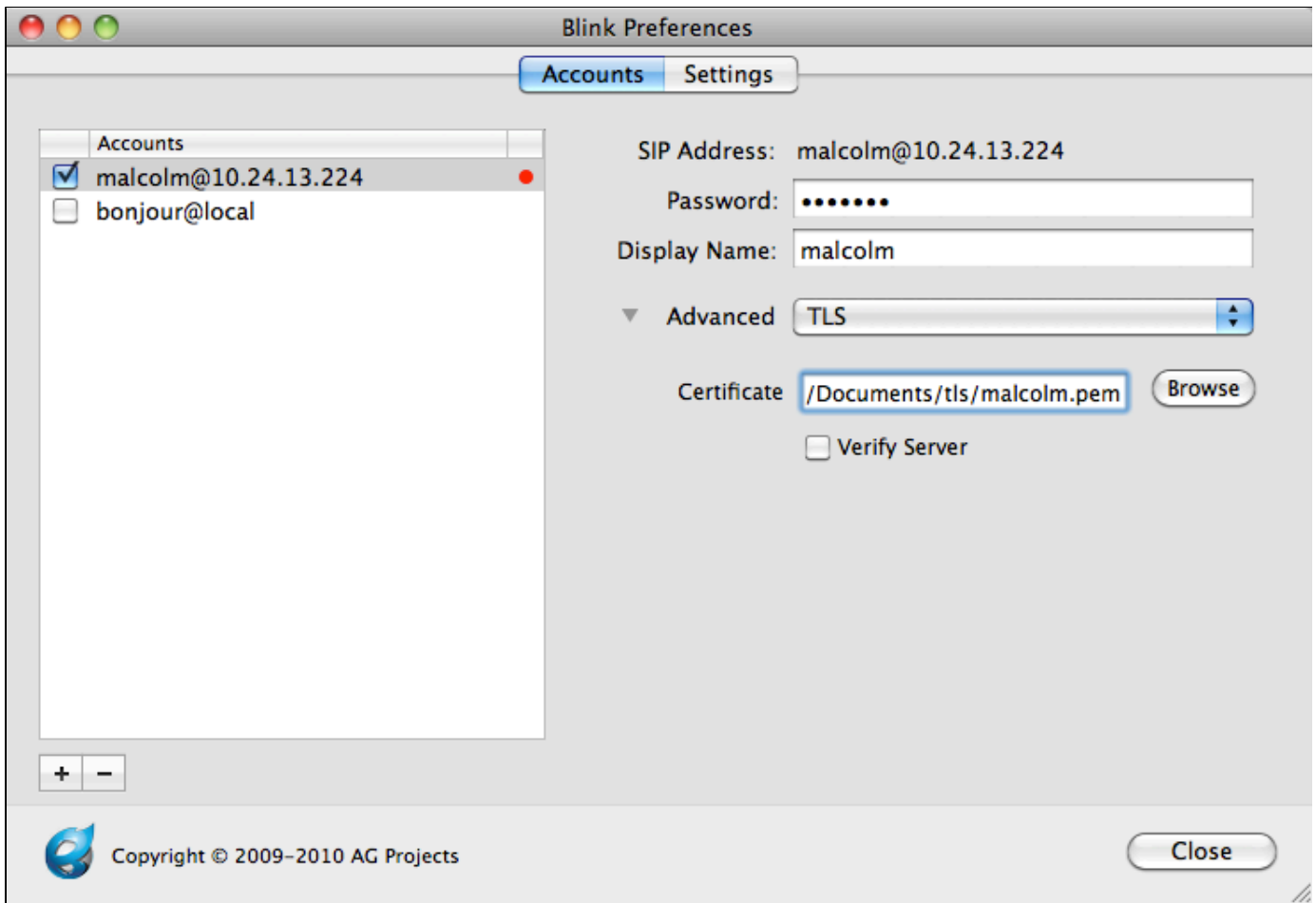
First, let's add a new account.



Then, we need to modify the Account Preferences, and under the SIP Settings, we need to set the outbound proxy to connect to the TLS port and transport type on our Asterisk server. In this case, there's an Asterisk server running on port 5061 on host 10.24.13.233.

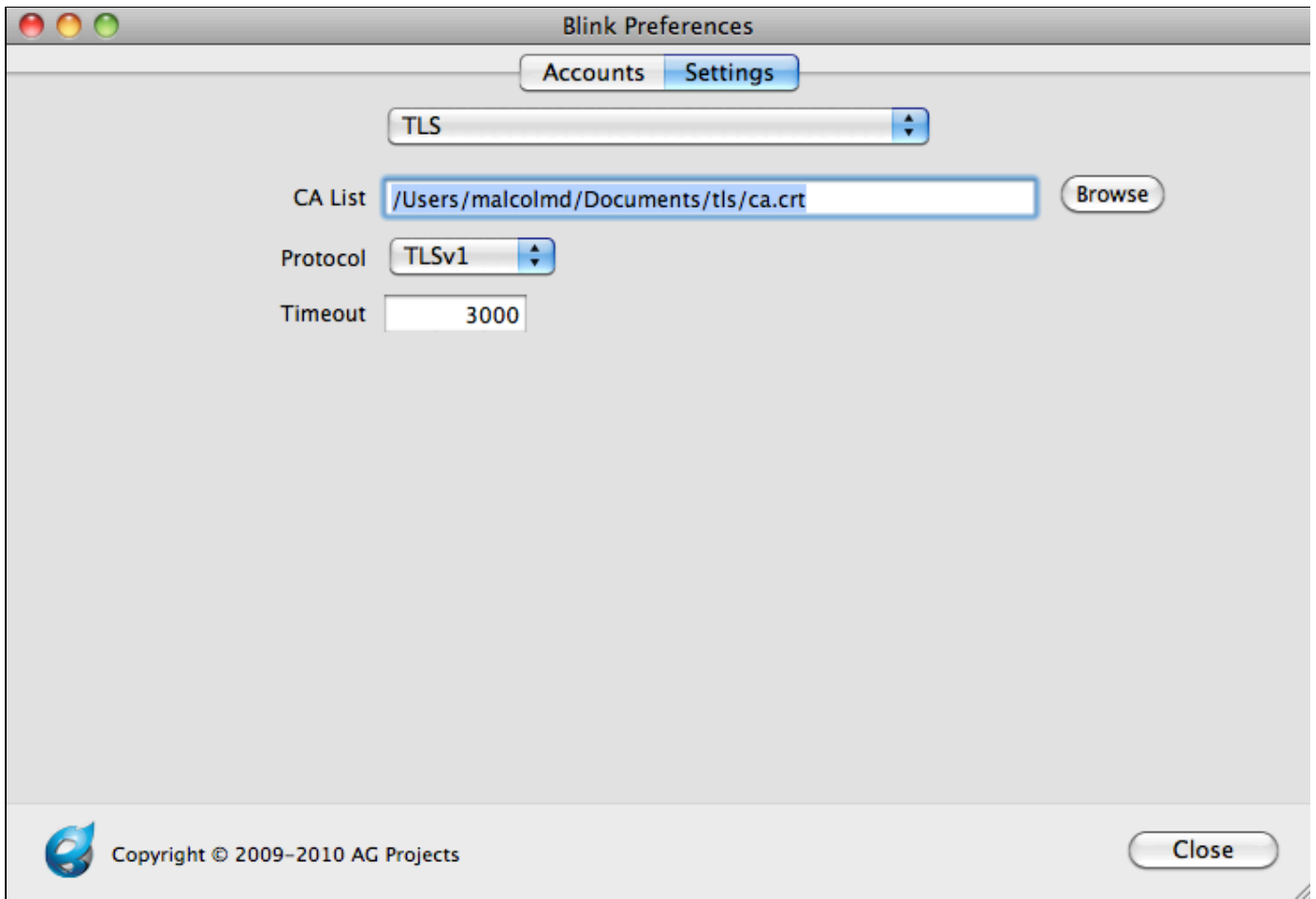


Now, we need to point the TLS account settings to the client certificate (malcolm.pem) that we copied to our computer.

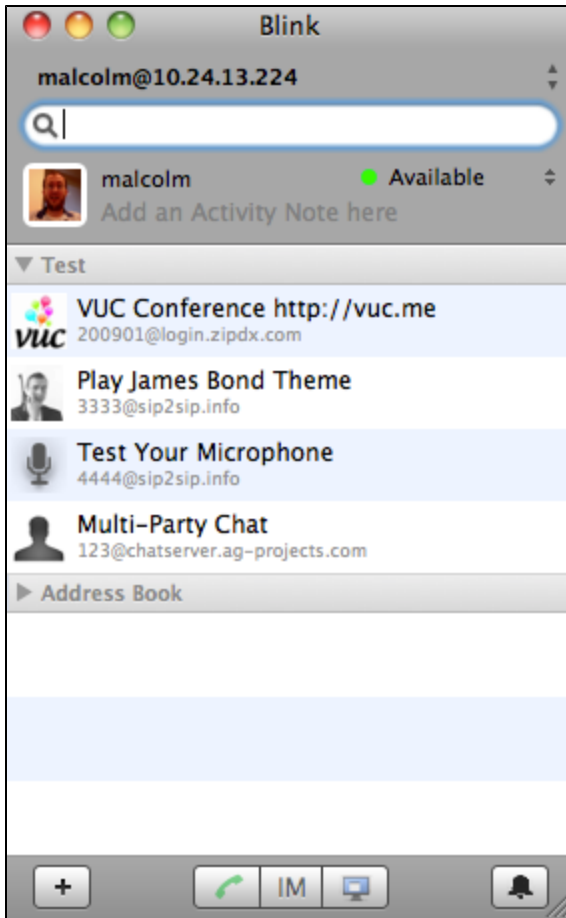


Then, we'll point the TLS server settings to the ca.crt file that we copied to our computer.





Press "close," and you should see Blink having successfully registered to Asterisk.

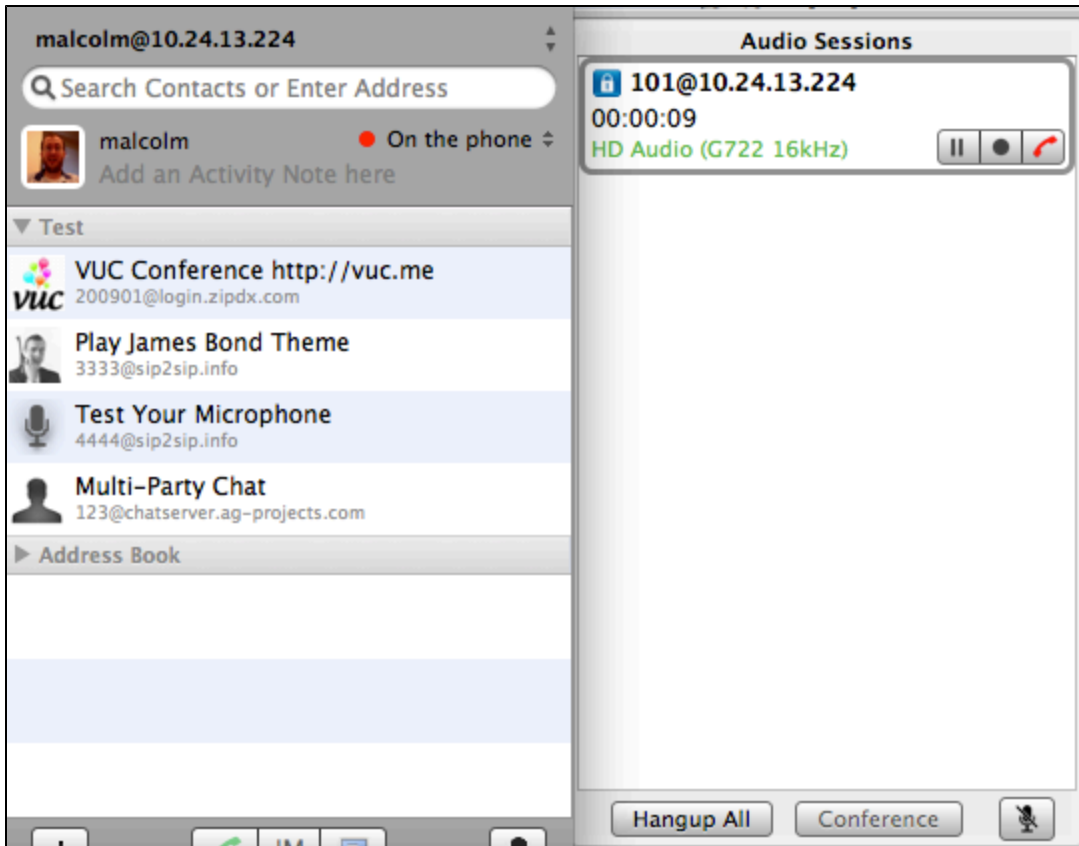


Depending on your Asterisk CLI logging levels, you should see something like:

```
-- Registered SIP 'malcolm' at 10.24.250.178:5061  
> Saved useragent "Blink 0.22.2 (MacOSX)" for peer malcolm
```

Notice that we registered on port 5061, the TLS port.

Now, make a call. You should see a small secure lockbox in your Blink calling window to indicate that the call was made using secure (TLS) signaling:



## Problems with server verification

If the host or IP you used for the common name on your cert doesn't match up with your server then you may run into problems when your client is calling Asterisk. Make sure the client is configured to **not** verify the server against the cert.

When calling **from** Asterisk to Blink or another client, you might run into an ERROR on the Asterisk CLI similar to this:

```
[Jan 29 16:04:11] DEBUG[11217]: tcptls.c:248 handle_tcptls_connection: SSL Common Name compare s1='10.24.18.124'
s2='phone1.mycompany.com'
[Jan 29 16:04:11] ERROR[11217]: tcptls.c:256 handle_tcptls_connection: Certificate common name did not match (10.24.18.124)
```

This is the opposite scenario, where Asterisk is acting as the client and by default attempting to verify the destination server against the cert.

You can set **tlsdontverifyserver=yes** in sip.conf to prevent Asterisk from attempting to verify the server.

```
;tlsdontverifyserver=[yes|no]
;   If set to yes, don't verify the servers certificate when acting as
;   a client.  If you don't have the server's CA certificate you can
;   set this and it will connect without requiring tlscacfile to be set.
;   Default is no.
```

## Part 2 (SRTP)

Now that we've got TLS enabled, our signaling is secure - so no one knows what extensions on the PBX we're dialing. But, our media is still not secure - so someone can snoop our RTP conversations from the wire. Let's fix that.

SRTP support is provided by libsrtp. libsrtp has to be installed on the machine before Asterisk is compiled, otherwise you're going to see something like:

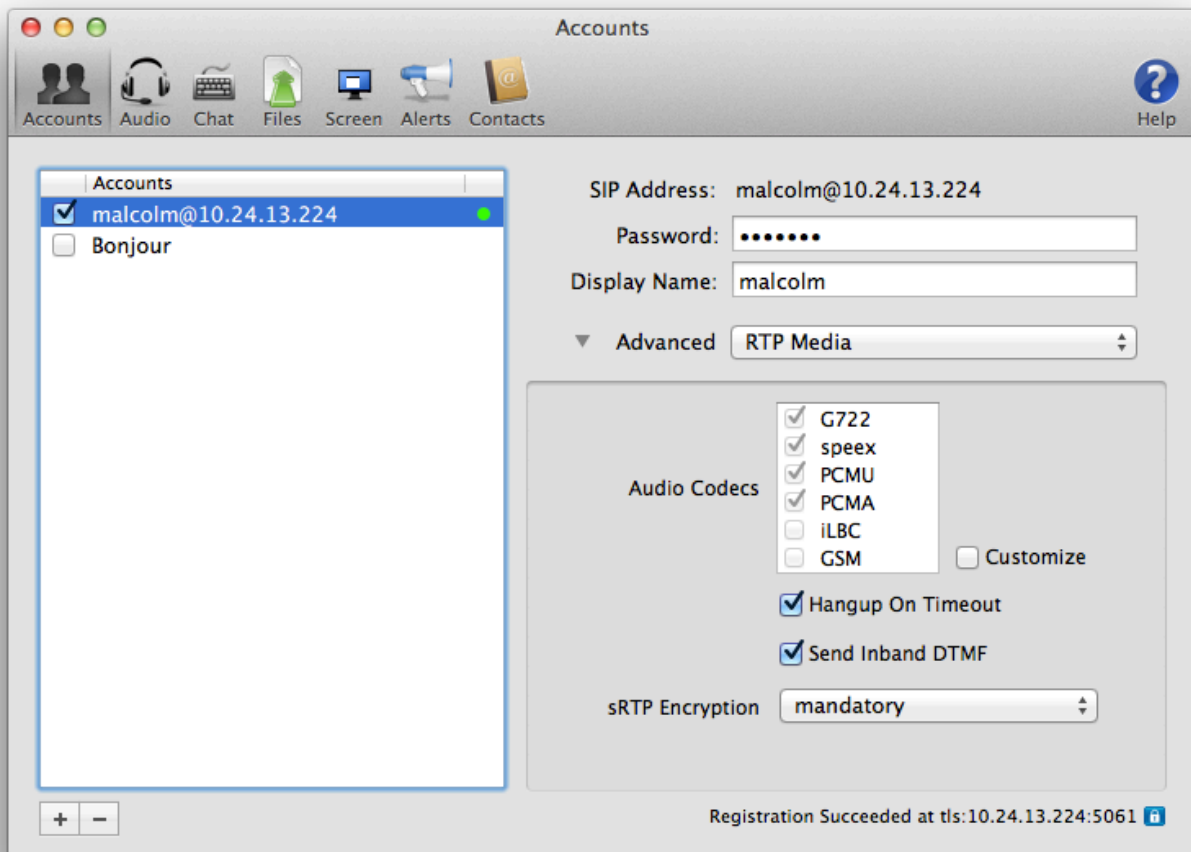
```
[Jan 24 09:29:16] ERROR[10167]: chan_sip.c:27987 setup_srtp: No SRTP module loaded, can't setup SRTP session.
```

on your Asterisk CLI. If you do see that, install libsrtp (and the development headers), and then reinstall Asterisk (./configure; make; make install).

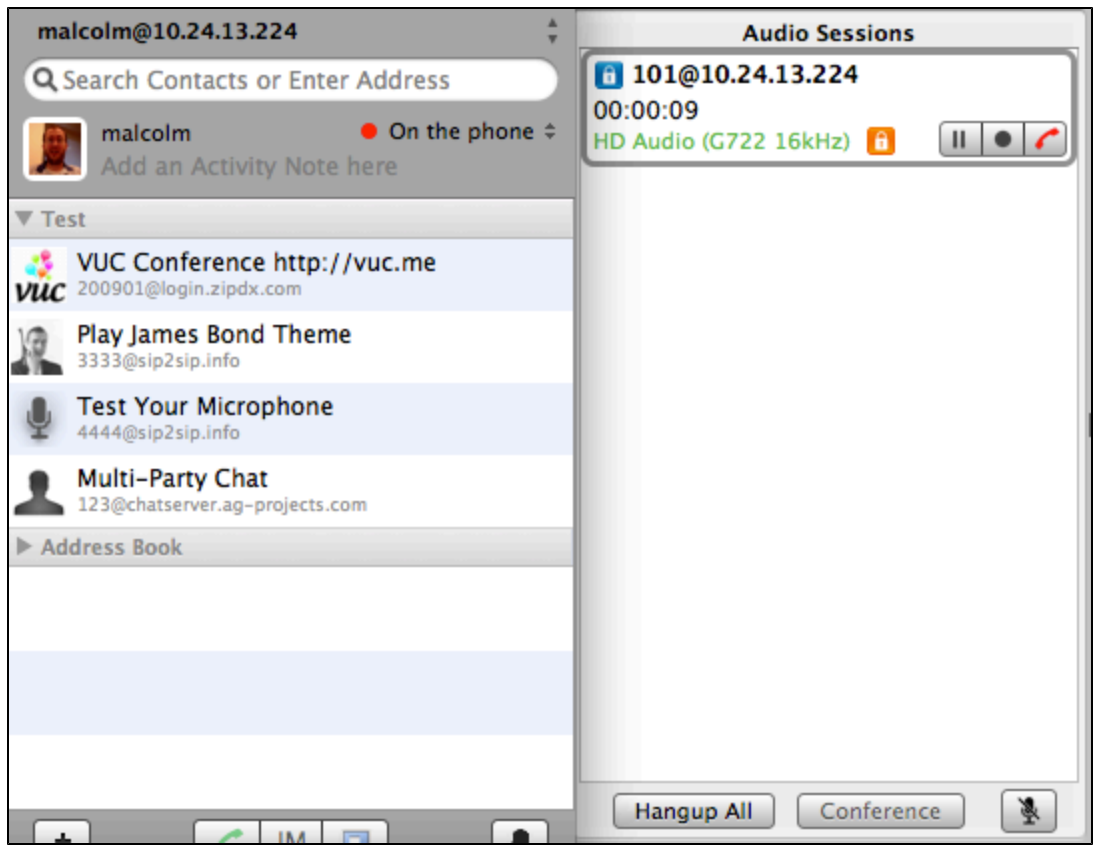
With that complete, let's first go back into our peer definition in **sip.conf**. We're going to add a new encryption line, like:

```
[malcolm]
type=peer
secret=malcolm ;note that this is NOT a secure password
host=dynamic
context=local
dtmfmode=rfc2833
disallow=all
allow=g722
transport=tls
encryption=yes
context=local
```

Next, we'll set Blink to use SRTP:



Reload Asterisk's SIP configuration (sip reload), make a call, and voilà:



We're making secure calls with TLS (signaling) and SRTP (media).

# Installing Blink SIP client

## Overview

I wanted to provide some brief instructions on installing the Blink SIP client on Linux since it is useful for running the [Secure Calling Tutorial](#).

## How to install Blink on Ubuntu 12.04 Precise Pangolin

### Install some dependencies

1. \*Get latest Python 2.X
2. Setup repo from here: <http://projects.ag-projects.com/projects/documentation/wiki/Repositories>
3. In your terminal, run the following commands

```
sudo apt-get update
sudo apt-get install python-sipsimple python-gnutls python-eventlib python-application python-cjson python-eventlet
python-qt4 python-twisted-core python-zope.interface
```

If you have issues see: <http://sipsimpleclient.org/projects/sipsimpleclient/wiki/SipInstallation>

### Download and install Blink

1. Download Blink source code from <http://download.ag-projects.com/BlinkQt>
2. In your terminal, extract the tar.gz file contents to a folder and then **cd** to that folder in the terminal
3. Change directory into the Blink folder and run **sudo python setup.py install**

```
root@newtonr-laptop:/usr/src/blink-0.6.0# sudo python setup.py install
running install
running build
running build_py
running build_scripts
running install_lib
running install_scripts
changing mode of /usr/local/bin/blink to 755
running install_data
running install_egg_info
Removing /usr/local/lib/python2.7/dist-packages/blink-0.6.0.egg-info
Writing /usr/local/lib/python2.7/dist-packages/blink-0.6.0.egg-info
```

### Run Blink!

1. From the command line, run Blink by executing blink.

```
root@newtonr-laptop:/usr/src/blink-0.6.0# blink
using set_wakeup_fd
"sni-qt/6493" WARN 08:11:15.444 void StatusNotifierItemFactory::connectToSsw() Invalid interface to SNW_SERVICE
```

2. Blink should launch and show up within your graphical desktop.



Blink doesn't appear to support call forwarding or call transfers, so don't expect to do anything too fancy!

# SIP TLS Transport

## Asterisk SIP/TLS Transport

When using TLS the client will typically check the validity of the certificate chain. So that means you either need a certificate that is signed by one of the larger CAs, or if you use a self signed certificate you must install a copy of your CA certificate on the client.

So far this code has been tested with:

- Asterisk as client and server (TLS and TCP)
- Polycom Soundpoint IP Phones (TLS and TCP) - Polycom phones require that the host (ip or hostname) that is configured match the 'common name' in the certificate
- Minisip Softphone (TLS and TCP)
- Cisco IOS Gateways (TCP only)
- SNOM 360 (TLS only)
- Zoiper Biz Softphone (TLS and TCP)

### sip.conf options

- `tlsenable=yes` - Enable TLS server, default is `no`
- `tlsbindaddr=<ip address>` - Specify IP address to bind TLS server to, default is `0.0.0.0`
- `tlscertfile=</path/to/certificate>` - The server's certificate file. Should include the key and certificate. This is mandatory if you're going to run a TLS server.
- `tlscacfile=</path/to/certificate>` - If the server you're connecting to uses a self signed certificate you should have their certificate installed here so the code can verify the authenticity of their certificate.
- `tlscapath=</path/to/ca/dir>` - A directory full of CA certificates. The files must be named with the CA subject name hash value. (see `man SSL_CTX_load_verify_locations` for more info)
- `tlsdontverifyserver=yes` - If set to `yes`, don't verify the servers certificate when acting as a client. If you don't have the server's CA certificate you can set this and it will connect without requiring `tlscacfile` to be set. Default is `no`.
- `tlscipher=<SSL cipher string>` - A string specifying which SSL ciphers to use or not use. A list of valid SSL cipher strings can be found at [http://www.openssl.org/docs/apps/ciphers.html#CIPHER\\_STRINGS](http://www.openssl.org/docs/apps/ciphers.html#CIPHER_STRINGS)

### Sample config

Here are the relevant bits of config for setting up TLS between 2 Asterisk servers. With `server_a` registering to `server_b`

On `server_a`:

```
[general]
tlsenable=yes
tlscertfile=/etc/asterisk/asterisk.pem
tlscacfile=/etc/ssl/ca.pem ; This is the CA file used to generate both certificates
register => tls://100:test@192.168.0.100:5061

[101]
type=friend
context=internal
host=192.168.0.100 ; The host should be either IP or hostname and should
                  ; match the 'common name' field in the servers certificate
secret=test
dtmfmode=rfc2833
disallow=all
allow=ulaw
transport=tls
port=5061
```

On `server_b`:

```
[general]
tlsenable=yes
tlscertfile=/etc/asterisk/asterisk.pem

[100]
type=friend
context=internal
host=dynamic
secret=test
dtmfmode=rfc2833
disallow=all
allow=ulaw
;You can specify transport= and port=5061 for TLS, but its not necessary in
;the server configuration, any type of SIP transport will work
;transport=tls
;port=5061
```



## Reference Use Cases for Asterisk

This section contains fictitious reference customers and users to serve as a platform for use cases to base all future documentation, tutorials and example files on.

# Super Awesome Company

## Overview

This page describes the usage of Asterisk within Super Awesome Company (SAC), a fictitious entity for the purpose of example only. All companies, entities and peoples on this page should be assumed fictitious.

Original creator and humourist - the notorious Malcolm Davenport.

This document will be one of a few [Reference Use Cases for Asterisk](#) and will be used as a basis for examples and tutorials on this wiki and included with Asterisk.

## Background

SAC, founded in 2015 by the noted daredevil Lindsey Freddie, the first person to BASE jump the Burj Khalifa with a paper napkin in place of a parachute, designs and delivers strategic software solutions for the multi-level marketing industry. Humbly beginning in Freddie's garage as a subsidiary of her paper delivery route, SAC has recently moved into modern offices, complete with running water, garbage collection, and access to the Internet and the regular telephone network - a substantial upgrade from the garage-based two cans and a string, water well and composting landfill.

### On this Page

- [Overview](#)
- [Background](#)
- [Phase 1, "We're a Business"](#)
  - [Company Organization](#)
  - [Outside Connectivity and Networking](#)
  - [Internal Calls](#)
  - [Main IVR](#)
  - [Employee Number Plan](#)
  - [Sales Queue](#)
  - [Customer Advocate Queue](#)
  - [Operator](#)
  - [Deskphones](#)
  - [Outbound calls](#)

## Phase 1, "We're a Business"

Upon the advice of the IT staff, SAC has decided to deploy a communications system using Asterisk. For desktop endpoints, SAC has decided to deploy Digium's IP telephones.

SAC requires:

- desk phones for each of its employees
- the ability for each employee to receive direct calls from their loved ones at home as well as kind-hearted telemarketers
- the ability for each employee to place calls to their loved ones at home, their bookies, and also potential customers
- the ability for employees to call one another inside of the office
- a main number that can be dialed by the public and that is answered by a friendly and sarcastic voice that can direct callers about what might be best for them
- a queue for inbound customers inquiring about many of the fine SAC software offerings
- a queue for inbound customers wondering why many of the fine SAC software offerings fail to live up to what customers refer to as "promises"
- an operator to handle callers that spend too much time waiting in queues
- voicemail boxes for each of the employees
- a conference bridge for use by customers and employees
- a conference bridge for use by employees only

## Company Organization

SAC employees 15 people with the following corporate structure:

- Lindsey Freddie ([lindsey@example.com](mailto:lindsey@example.com)), President for Life
  - Temple Morgan ([temple@example.com](mailto:temple@example.com)), Life Assistant to the President for Life
  - Terry Jules ([terry@example.com](mailto:terry@example.com)), Director of Sales
    - Garnet Claude ([garnet@example.com](mailto:garnet@example.com)), Sales Associate
    - Franny Ocean ([franny@example.com](mailto:franny@example.com)), Sales Associate
  - Maria Berny ([maria@example.com](mailto:maria@example.com)), Director of Customer Experience
    - Dusty Williams ([dusty@example.com](mailto:dusty@example.com)), Customer Advocate
    - Tommie Briar ([tommie@example.com](mailto:tommie@example.com)), Customer Advocate
  - Penelope Bronte ([penelope@example.com](mailto:penelope@example.com)), Director of Engineering

- Hollis Justy (hollis@ example.com ), Software Engineer
- Richard Casey (richard@ example.com ), Software Engineer
- Sal Smith (sal@ example.com ), Software Engineer
- Laverne Roberts (laverne@ example.com ), Software Engineer
- Colby Hildred (colby@ example.com ), IT Systems
- Aaron Courtney (aaron@ example.com ), Accounting and Records

By a stroke of luck, Freddie was also the original registrant of the [example.com](#) domain name, which, having been lost in years past, she recently managed to win back from ICANN in a game of high-stakes chubby bunny.

## Outside Connectivity and Networking

SAC, located in the ghost town of Waldo, Alabama, is fortunate to have access to any type of telecommunications. In spite of Waldo's sole telecom company, WaldoCom, having turned off the lights and locked the doors on their way out, gigabit Internet connectivity over Metro Ethernet is available and operational. It is rumored that WaldoCom left behind a skeleton crew to maintain equipment on an ongoing basis.

WaldoCom, as a traditional communications provider will allow SAC to purchase telephone service. But, because the calling plans across the WaldoCom telephone network must be paid in Confederate dollars, SAC has instead decided to contract with Digium, Inc. for voice services. Digium provides inbound and outbound calling over the Internet as a "SIP Trunk" using SAC's existing Internet connectivity.



WaldoCom has ceased trading in Bitcoin after having been accused by the FBI of running the Silk Road as the Dread Pirate Roberts.

SAC has purchased a well-loved Linksys WRT54G, aka "Old Unreliable," from the now defunct Waldo Happy Hands Club. They intend to use it to terminate the Ethernet connectivity from WaldoCom. WaldoCom provides a single IPv4 address across the link - 203.0.113.1. The Linksys will provide NAT translation from the Internet to the internal SAC campus network. Within the SAC network, the 10.0.0.0/8 address space will be used.

### DIDs / Telephone Numbers

Digium has provided SAC with a block of 300 DIDs (Telephone numbers) beginning with 256-555-1100 and ending with 256-555-1299.

For inbound calls, each SAC employee has a direct DID that rings them.

For outbound calls, each SAC employee's 10-digit DID number is provided as their Caller ID.

### Internal Calls

As a shortcut to dialing 10 digits, SAC uses 4-digit dialing for calls between internal employees. The 4 digits assigned to each employee match that employee's last 4 digits of their inbound DID. Each employee has been individually assigned a number that can be dialed by any other employee.

SAC has other direct-dial numbers assigned to Queues and menus.

SAC also has several internal extensions that do not map directly to DIDs.

### Calls to an Extension

Calls made directly to an extension should ring for 30 sec. After 30 seconds, callers should be directed to the voicemail box of the employee. If the employee was on the phone at the time, callers should hear the voicemail busy greeting; otherwise, callers should hear the voicemail unavailable greeting.

### Voicemail

When Asterisk records a voicemail for an SAC employee, it should forward a copy of that voicemail to the employee's e-mail.

### Main IVR

SAC has a main IVR. Potential multi-level marketing brand ambassadors or disgruntled bottom-tier customers who find SAC in the WaldoCom Yellow Pages are greeted with a friendly voice and the following helpful message:

"Thank you for calling Super Awesome Company, Waldo's premier provider of perfect products. If you know your party's extension, you may dial it at any time. To establish a sales partnership, press one. To speak with a customer advocate, press two. For accounting and other receivables, press three. For a company directory, press four. For an operator, press zero."

If no DTMF entry is detected, the main IVR repeats twice and then hangs up on the caller.

### Employee Number Plan

Temple Morgan, Esquire, Life Assistant to the President for Life, Lindsey Freddie, has randomly assigned the following phone numbers for each of the employees:

Internal Extensions

- Maria Berny - 256-555-1101
- Tommie Briar - 256-555-1102
- Penelope Bronte - 256-555-1103
- Richard Casey - 256-555-1104
- Garnet Claude - 256-555-1105
- Aaron Courtney - 256-555-1106
- Lindsey Freddie - 256-555-1107
- Colby Hildred - 256-555-1108
- Terry Jules - 256-555-1109
- Hollis Justy - 256-555-1110
- Temple Morgan - 256-555-1111
- Franny Ocean - 256-555-1112
- Laverne Roberts - 256-555-1113
- Sal Smith - 256-555-1114
- Dusty Williams - 256-555-1115

## Other Numbers

- SAC provides an extension, 8000, that lets internal employees directly dial their voicemail box. When internal SAC employees dial their voicemail box, they're not prompted for their mailbox number or their PIN code.
- Voicemail may be accessed remotely by employees who dial 256-555-1234. When employees dial voicemail remotely, they must input both their mailbox number and their pin code.
- The Main IVR may be reached externally by dialing 256-555-1100, or internally by dialing 1100.
- The Sales Queue may be reached externally by dialing 256-555-1200, or internally by dialing 1200.
- The Customer Experience Queue may be reached externally by dialing 256-555-1250, or internally by dialing 1250.
- An Operator may be reached from any IVR menu or calling queue by dialing 0.
- The company has a party-line conference room, for use by any employee, on extension 6000.
- The company has a conference room that can be used by customers and employees on extension 6500.

## Sales Queue

Calls to the Sales Queue should ring Terry, Garnet and Franny in ring-all fashion

If no one answers a call to the Sales Queue after 5 minutes, the caller should be directed to the Operator so that the Operator can take a message and have a Sales person contact the caller at a later time.

## Customer Advocate Queue

Calls to the Customer Advocate Queue should ring Maria, Dusty and Tommie in ring-all fashion.

If no one answers a call to the Customer Advocate queue after 20 minutes, the caller should be directed to the Operator so that the Operator can take a message and have a Customer Advocate contact the caller at a later time.

## Operator

Temple Morgan serves as the Operator for SAC.

## Deskphones

Each SAC employee is provided with a Digium D70 model telephone.

## Outbound calls

Because WaldoCom never upgraded their switches beyond 5-digit dialing, which now covers the entire metropolis of Waldo, SAC employees are not used to dialing 10-digit numbers. Since most calls to potential customers will be made inside the local Waldo area, SAC has managed, through extensive training, to convince their employees to dial 7-digits for local Waldo numbers dialed over their Digium trunk. Because Freddie believes there may be a market for the Broomshakalaka and other products outside of Waldo, perhaps even as far away as Montgomery or Mobile, she has also insisted that 10-digit dialing and 1+10-digit dialing be possible.

# Asterisk Security Vulnerabilities

The Asterisk project takes the issue of its users security seriously. If you believe you have found a security vulnerability in Asterisk, please follow the steps on this wiki page to report the security vulnerability to the Asterisk Development Team.



## The Issue Tracker is Public!

The [Asterisk Issue Tracker](#) is a public site, and all bug reports against Asterisk can be viewed openly by the public. While this results in a transparent, open process - which is good - reporting a security vulnerability on the issue tracker without properly locking the issue makes the entire Asterisk user community vulnerable.

The issue tracker does have the ability to lock issues privacy to only the bug reporter and bug marshals. We'll help you with that; please follow the instructions here and e-mail the team at [security@asterisk.org](mailto:security@asterisk.org).

## Reporting a Security Vulnerability

1. Send an e-mail to the Asterisk Development Team by e-mailing [security@asterisk.org](mailto:security@asterisk.org). Include the following:
  - a. A summary of the suspected vulnerability, e.g., 'Remotely exploitable buffer overflow in the FOO channel driver'
  - b. A detailed explanation of how the vulnerability can be exploited and/or reproduced. Test drivers/cases that can be used to demonstrate the vulnerability are highly appreciated.
2. A developer will respond to your inquiry. If you'd like, e-mails can be signed and/or encrypted.
3. A private issue will be created by a bug marshal in the Asterisk issue tracker for your vulnerability.
4. The bug marshal will lock the issue down and change the reporter to your user account. **only at that point** should you post details to the locked issue.

Security vulnerabilities are treated seriously by the developer community, and the Asterisk Development Team always attempts to address vulnerabilities in a timely fashion. Sometimes, external influences may impact when a security release can be made; feel free to e-mail the developer assigned to the issue or [security@asterisk.org](mailto:security@asterisk.org) to discuss the schedule for a security release for your issue.

## Past Security Vulnerabilities

Past security vulnerability reports are available on the [asterisk.org web site](#) and on the [Asterisk downloads site](#).

All security vulnerabilities are also issued a CVE number and can be queried in the [CVE database](#).

## Participating in Security Issues

All Asterisk Developers who have commit access are welcome to participate in the development of solutions to security issues. Security issues can be viewed in JIRA using the label `Security`:

```
project=ASTERISK and labels=Security
```

When a new security issue is created, an e-mail will be sent to the asterisk-dev mailing list notifying the community of the issue. This e-mail will not contain any information about the vulnerability, and will merely contain a link to the new security issue.

When a patch is ready to be peer reviewed on [Review Board](#), a review will be created using the review group `Security`. This group is not normally visible to review board users, is invite only, and generates no e-mails to the asterisk-dev mailing list. Users who have the Bug Marshal permission in JIRA are invited to participate in the review process.



Please exercise caution when participating in security issues. It is far better to 'test' a reply or message on an issue, e-mail, or review than to leak information.

Once the patch has been peer reviewed, it should **not** be committed. Committing the patch must be coordinated as an overall security release. This is typically handled by the affected branch maintainers.

# Asterisk Community

- [Asterisk Community Code of Conduct](#)
- [Asterisk Community Services](#)
- [Asterisk Issue Guidelines](#)
- [Asterisk Module Support States](#)
- [Community Services Signup](#)
- [IRC](#)
- [Mailing Lists](#)
- [Wiki Organization and Style Guide](#)

# Asterisk Community Code of Conduct

The Asterisk community is a large group of individuals, representing many nations, ethnicities, ages, technical professions and specialities. Working together on Asterisk can be a challenge with so many differing perspectives and backgrounds. Therefore to ensure the community is healthy, happy, and stress-free, participants in the Asterisk project agree to adhere to the following Community Code of Conduct.

Note that by joining and/or participating in the Asterisk community, you are agreeing that you accept and will adhere to the rules listed below, even if you do not explicitly state so.

## Acceptable Behaviour

### Be considerate

- Experience levels vary. Don't assume that someone can understand your particular explanation.
- Keep in mind that English is a second language for many users.

### Be respectful

- It is possible to strongly disagree without using harsh language or resorting to derogatory comments. If you disagree with someone, disagree with the argument, not the character of the person.
- Remember that everyone is entitled to an opinion.

### Ask for help

- If you don't know how to proceed with an aspect of development or documentation, ask for help! Always read the documentation, but don't be afraid to ask "silly" questions.
- When asking for help, take advantage of the resources that are available to you, including the [wiki](#), [mailing list archives](#), and [Asterisk: The Definitive Guide](#).

### Take responsibility

- If you did something wrong, apologize to the affected. Do your best to fix the issue and, if you can't, ask for help!
- If someone does take responsibility, be considerate.

### Give credit

- Give proper credit to everyone involved In any contribution to the project, be it documentation, tests, code, or anything in between.
- If someone fails to give adequate contribution, gently remind them while being considerate. Assume that the omission was accidental, not malicious.

## Unacceptable Behaviour

The Asterisk project reserves the right to take action in safe-guarding the community from those that participate in unacceptable behaviour. Unacceptable behaviour involves:

- Flaming - Arguing in a disrespectful way, attacking the character of others, rabidly ranting about things you dislike and refusing to drop the topic.
- Trolling - Intentionally baiting others into flaming or heated arguments for the sake of argument or drama itself.
- Mean-spirited or offensive talk - This could be combinations of the above, being rude, vulgar, and generally offensive to others.

In general, if community moderators and administrators are receiving many complaints about your behaviour, then you are likely doing something wrong. If you don't have anything nice to say, don't say anything.

Consequences to bad behaviour may involve bans from communication forums or channels and restrictions on privileges within the community.

Complaints about members behaviour or appeals in regards to bans or loss of privilege can be sent to [asteriskteam@digium.com](mailto:asteriskteam@digium.com).

## Open Community

We invite anybody, from any company, locale, or even other projects to participate in the Asterisk project. Our community is open, and contribution is welcome. Diversity makes a project strong, and we are proud to include anyone who wants to collaborate with others in a respectful fashion.

## Project Leadership

The role of project leadership is handled by the founders of the Asterisk project, [Digium Inc.](#) As a member of the Asterisk community, Digium develops the project in co-operation with the overall Asterisk community. Community members are always welcome to take positions of leadership as [module maintainers](#) within the Asterisk project, particularly when they are the author of the module.

In addition to providing development resources for Asterisk itself, Digium provides community resources including the bug tracker, mailing lists, wiki, version control, continuous integration services, and other necessary project infrastructure. [Asterisk goals and objectives](#) are decided upon along with the community at the annual [AstriDevCon](#) held at [AstriCon](#). Development discussions take place on the public [asterisk-dev mailing list](#) and the [#asterisk-dev](#) IRC channel. More information on the development of Asterisk can be found on the Asterisk wiki.



# Asterisk Community Services

There are several services provided for the development of Asterisk.

- [Issue Tracking](#)
- [Wiki](#)
- [Code Review](#)
- [Version Control](#)
- [Source Browsing](#)
- [Mailing Lists](#)

There is a regular maintenance window every Monday from 9:00 PM to 10:00 PM Central Time, during which services may have intermittent availability while we apply patches, upgrades, etc. If maintenance is required outside of this window, notice will be sent to the [asterisk-announce](#) mailing list. If any of the community services are unavailable outside of a scheduled maintenance time, please notify [the Asterisk development team](#).

# Asterisk Issue Guidelines

- [Purpose of the Asterisk issue tracker](#)
- [Bug Reporting Check List](#)
- [Submitting the bug report](#)
- [How to request a feature](#)
- [Patch and Code Submission](#)
- [Issue Lifetime](#)
- [Frequently Asked Questions](#)

## Purpose of the Asterisk issue tracker

The [Asterisk Issue Tracker](#) is used to track bugs and miscellaneous (documentation) elements within the Asterisk project. The issue tracker is designed to manage reports on both [core](#) and [extended components](#) of the Asterisk project.

The primary use of the issue tracker is to track bugs, where "bug" means anything that causes unexpected or detrimental results in the Asterisk system. The secondary purpose is to track some of the miscellaneous issues surrounding Asterisk, such as documentation, commentary and feature requests or improvements with associated patches.

Feature requests without patches are generally not accepted through the issue tracker. Instead, they are discussed openly on the mailing lists and Asterisk IRC channels. Please read the ["How to request a feature"](#) section of this article.

### What the issue tracker is not used for:

- **Information Requests:** (How does X parameter work?)  
See the forums, mailing lists, IRC channels, or this wiki. For even more information, see <http://www.asterisk.org/community>
- **Support requests:** (My phone doesn't register! My database connectivity doesn't work! How do I get it to work?)  
Search and ask on the forums, mailing lists, and IRC. Again, see <http://www.asterisk.org/community> for more information.
- **Random wishes and feature requests with no patch:** (I want Asterisk to support <insert obscure protocol or gadget>, but I don't know how to code!)  
See the [How to request a feature section](#) for more information on requesting a feature.
- **Business development requests** (I will pay you to make Asterisk support fancy unicorn protocol!)  
Please head to the asterisk-biz mailing list at <http://lists.digium.com>. If what you want is a specific feature or bug fixed, you may want to consider [requesting a bug bounty](#).

### Why should you read this?

The steps here will help you provide all the information the Asterisk team needs to solve your issue. Issues with clear, accurate, and complete information are often solved much faster than issues lacking the necessary information.

## Bug Reporting Check List



### Before filing a bug report...

Your issue may not be a bug or could have been fixed already. Run through the check list below to verify you have done your due diligence.

- **Are you reporting a suspected security vulnerability?**  
Please stop! The issue tracker is public, and once a security vulnerability is made public on the issue tracker, all users of Asterisk are made vulnerable. Please see [Asterisk Security Vulnerabilities](#) for more information on reporting a suspected security vulnerability.
- **Are you are on a supported version of Asterisk?**  
See the [Asterisk Versions](#) page for the currently supported versions of Asterisk. Bugs reported against versions of Asterisk that are no longer supported will be closed, and you will be asked to reproduce the issue against a supported version.
- **Are you using the latest version of your Asterisk branch?**  
Please check the release notes for newer versions of Asterisk to see if there is a potential fix for your issue. Even if you can't identify a fix in a newer version, it is preferable that you upgrade when reasonable to do so. Release notes are available in the UPGRADE.txt, CHANGES and ChangeLog files within the root directory of your particular release, and are also available at <http://downloads.asterisk.org/pub/telephony/asterisk/>
- **Are you using the latest third party software, firmware, model, etc?**  
If the error scenario involves phones, third party databases or other software, be sure it is all up to date and check their documentation.
- **Have you asked for help in the community? (mailing lists, IRC, forums)**  
You can locate all these services here: <http://www.asterisk.org/community>
- **Have you searched the Asterisk documentation in case this behavior is expected?**  
Search the [Asterisk wiki](#) for the problem or messages you are experiencing.
- **Have you searched the Asterisk bug tracker to see if an issue is already filed for this potential bug?**  
Search the [Asterisk issue tracker](#) for the issue you are seeing. You can search for issues by selecting **Issues -> Search for Issues** in the top menu bar.
- **Can you reproduce the problem?**  
Problems that can't be reproduced can often be difficult to solve, and your issue may be closed if you or the bug marshals cannot

reproduce the problem. If you can't find a way to simulate or reproduce the issue, then it is advisable to configure your system such that it is capturing relevant debug during the times failure occurs. Yes, that could mean running debug for days or weeks if necessary.

## Submitting the bug report

You'll report the issue through the tracker (<https://issues.asterisk.org/jira>), under the **ASTERISK** project.

### 1. Sign up for an asterisk.org account

Account signup for [asterisk.org](https://issues.asterisk.org) community services (including [JIRA](#), [Confluence](#) and [FishEye/Crucible](#)) can be found at <https://signup.asterisk.org>.

### 2. Create a new issue in the tracker

Select the **Create Issue** button in the top right hand corner of the page. This will prompt you for the project and issue type. Pick the **ASTERISK** project, and choose an appropriate issue type. The following are the supported issue types:

- Bug
- New feature
- Improvement



Please do not select the Information request issue type. This type is not currently used, and your issue is likely to be closed or redefined.

### 3. Fill out the issue form

For a bug you must include the following information:

- **Concise and descriptive summary**
  - Accurate and descriptive, not prescriptive. Provide the facts of what is happening and leave out assumptions as to what the issue might be.
  - Good example: "Crash occurs when exactly twelve SIP channels hang up at the same time inside of a queue"
  - Bad Examples: "asterisk crashes", "problem with queue", "asterisk doesn't work", "channel hangups cause crash"
- **Operating System detail** (Linux distribution, kernel version, architecture etc)
- **Asterisk version** (exact branch and point release, such as 1.8.12.0)
- **Information on any third party software involved in the scenario** (database software, libraries, etc)
- **Frequency and timing of the issue** (does this occur constantly, is there a trigger? Every 5 minutes? seemingly random?)
- **Symptoms** described in specific detail ("No audio in one direction on only inbound calls", "choppy noise on calls where trans-coding takes place")
- **Steps required to reproduce the issue** (tell the developer exactly how to reproduce the issue, just imagine you are making steps for a manual)
- **Workarounds in detail with specific steps** (if you found a workaround for a serious issue, please include it for others who may be affected)
- **Debugging output** - You'll almost always want to include `extensions.conf`, and config files for any involved component of Asterisk. Depending on the issue you may also need the following:
  - For crashes, provide a backtrace generated from an Asterisk core dump. See [Getting a Backtrace](#) for more information.
  - For apparent deadlocks, you may need to enable the compile time option `DEBUG_THREADS`. A backtrace may also be necessary. See [Getting a Backtrace](#) for more information.
  - For memory leaks or memory corruptions, [Valgrind](#) may be necessary. Valgrind can detect memory leaks and memory corruptions, although it does result in a substantial performance impact.
  - For debugging most problems, a properly generated debug log file will be needed. See [CLI commands useful for debugging](#) and [Collecting Debug Information](#) for more information. Note that for issues involving SIP, IAX2, or other channel drivers, you should enable that driver's enhanced debug mode through the CLI before collecting information. A pcap demonstrating the problem may also be needed.



Be courteous. Do not paste debug output in the description or a comment, instead please **attach** any debugging output as text files and reference them by file name.

### 4. Submit the Issue

Once you have created the issue, you can now attach debug as files. You are ready to wait for a response from a bug marshal or post additional information as comments. If you are responding to a request for feedback, be sure to use the workflow actions to "send back" the status to the developers.

## How you can speed up bug resolution

Follow the [checklist](#) and include all information that bug marshals require. Watch for emails where a bug marshal may ask for additional data and help the developers by testing any patches or possible fixes for the issue. A developer **cannot** fix your issue until they have sufficient data to reproduce - or at the very least understand - the problem.

## Reasons your report may be closed without resolution

If your bug:

- Is solely the result of a configuration error
- Is a request for Asterisk support
- Has incomplete information
- Cannot be reproduced

it may be closed immediately by a bug marshal. If you believe this to be in error, please comment on the issue with the reason why you feel the issue is still

a bug. You may also contact bug marshals directly in the #asterisk-bugs IRC channel. If that fails, bring it up on the mailing list(s) and it will be sorted out by the community.

## Working with Bug Marshals

If insufficient commentary or debug information was given in the ticket then bug marshals will request additional information from the reporter. If there are questions posted as follow-ups to your bug or patch, please try to answer them - the system automatically sends email to you for every update on a bug you reported. If the original reporter of the patch/bug does not reply within some period of time (usually 14 days) and there are outstanding questions, the bug/patch may get closed out, which would be unfortunate. The developers have a lot on their plate and only so much time to spend managing inactive issues with insufficient information.

If your bug was closed, but you get additional debug or data later on, you can always contact a bug marshal in #asterisk-bugs on irc.freenode.net to have them re-open the issue.

## How to request a feature

Feature requests without patches are not typically monitored or kept on the tracker. Even a feature request with a patch will still have to be approved by the core maintainers and community developers. Many people want Asterisk to do many different things; however, unless the feature has some extremely obvious (to many people) benefit, then it's best to have a community discussion and consensus on a feature before it goes into Asterisk. Feature requests are openly discussed on the [mailing lists](#) and [IRC channels](#). Most bug marshals and Asterisk developers actively participate and will note the request if it makes sense.

New features and major architecture changes are often discussed at the [AstriDevCon](#) event held before Astricon.

If you really need the feature, but are not a programmer, you'll need to find someone else to write the code for you. Luckily, there are many talented Asterisk developers out there that can be contacted on the [asterisk-biz](#) mailing list. You can also offer a bounty for a bug or new feature - see [Asterisk Bug Bounties](#) for more information.

## Patch and Code Submission

Patches for all issues are always welcome. Issues with patches are typically resolved much faster than those without. Please see the [Patch Contribution Process](#) for information on submitting a patch to the Asterisk project.

For information on writing a new feature for Asterisk, please see the [New Feature Guidelines](#).

## Issue Lifetime

The status of all issues in the tracker are reflected in the tracker. As issues are worked, they will move through several statuses. Issues that are "Closed" or "Complete" will have a disposition that determines their resolution. Issues with a disposition of "Fixed" have been committed to the project and will be released in the next bug fix release for all supported releases of Asterisk.

For any particular issue you'll want to look at the "Status" field and the comments tab under "Activity".

You can read more about the issue workflow at [Issue Tracker Workflow](#).

Status Field Value	Meaning
Triage	The issue has not been acknowledged yet. First a bug marshal needs to verify it's a valid report. Check the comments!
Open\Reopened	Issue has been acknowledged and is waiting for a developer to take it on. Typically we can't provide an ETA for development as priorities are complex and change constantly.
In Progress	A developer is working on this issue. Check the comments!
Waiting for Feedback	Check the comments. This issue is waiting on the "Assignee" to provide feedback needed for it to move forward. Once feedback is provided you need to click "Send Back".
Closed	Issue has been closed. Check the "Resolution" field for further information.
Complete	The intended resolution was reached, but additional tasks may remain before the issue can be completely closed out.

## Frequently Asked Questions

### What makes a useful bug report?

There are some key qualities to keep in mind. These should be reflected in your bug report and will increase the chance of your bug being fixed!

- Specific: Pertaining to a certain, clearly defined issue with data to provide evidence of said issue
- Reproducible: Not random - you have some idea when this issue occurs and how to make it happen again

- Concise: Brief but comprehensive. Don't provide an essay on what you think is wrong. Provide only the facts and debug output that supports them.

### How can I be notified when an issue's status changes?

**Watch an issue:** You can receive E-mails whenever an issue is updated. You'll see a "watch" link on the actual JIRA issue, click that!



### What's the deal with the clone of my issue (SWP-1234)?

You can safely ignore the issues starting with SWP. Digium has a whole group of developers dedicated to working on Asterisk open source issues. They use a separate project for internal time tracking purposes only to avoid cluttering up the main project. The clone being created does not indicate any particular status, and any updates to the issue will be made in the ASTERISK project.

### How can I speed up my issue resolution?

1. Follow the guidelines on this patch! Having good, accurate information that helps bug marshals reproduce the issue typically leads to much faster issue resolution.
2. Provide a patch! Issues with patches are also generally resolved much faster. If you can't write a patch, there are many smart, talented developers in the Asterisk community who may be worth helping you. You can contract with them on the [asterisk-biz](#) mailing list, or offer a [bug bounty](#).

# Asterisk Module Support States

## Introduction

In Asterisk, modules can take one of three supported states. These states include:

- Core
- Extended
- Deprecated

The definition of the various states is described in the following sections.

## Core

Most modules in Asterisk are in the Core state, which means issues found with these modules can freely be reported to the Asterisk issue tracker, where the issue will be triaged and placed into a queue for resolution.

## Extended

This module is supported by the Asterisk community, and may or may not have an active developer. Some extended modules have active community developers; others do not. Issues reported against these modules may have a low level of support. For more information about which extended support modules have an active developer supporting them, see [Asterisk Open Source Maintainers](#).

## Deprecated

The module will remain in the tree, but there is a better way to do it. After two release cycles issues that have been deprecated for some time will be listed in an email to the Asterisk-Dev list where the community will have an opportunity to comment on whether a deprecated module: still compiles, works sufficiently well, and is still being utilized in a system where there is a justification for not using the preferred module.

## MODULEINFO Configurations

At the top of modules there is the ability to set meta information about that module. Currently we have the following tags:

- `<defaultenabled>`
- `<use>`
- `<depend>`

The meta data added to **MODULEINFO** in the module causes data to be generated in *menuselect*. For example, when you use `<defaultenabled>no</defaultenabled>` the module will not be selected by default. We would use the `<defaultenabled>` tag for *deprecated* modules so that they are not built unless explicitly selected by an Asterisk administrator.

## Adding New Metadata

On top of the existing tags, we would add two additional tags to communicate via *menuselect* that a module was extended or deprecated (and what module supersedes the deprecated module). These tags include:

- `<support_level>`
  - *Example:* `<support_level>deprecated</support_level>` -- This module would be deprecated. Maintenance of this module may not exist. It is possible this module could eventually be tagged as deprecated, or removed from the tree entirely.
- `<replacement>`
  - *Example:* `<replacement>func_odbc</replacement>` -- The replacement for this module is the `func_odbc` module. This is used when the `<support_level>` is set to **deprecated**.

## Menuselect Display

The following two images show the suggested *menuselect* output based on the addition of the `<support_level>` and `<replacement>` tags. This would be a new line that has not been used before, and therefore would be added to *menuselect* as that new line.

If the **deprecated** value is used, then the value between the `<replacement>` tags will replace the value of `app_superQ` as shown in the image below. The text surrounding `app_superQ` would be static (same for all modules that used **deprecated**).

```
*****
Asterisk Module and Build Option Selection
*****
```

Press 'h' for help.

```
[*] app_minivm
[*] app_mixmonitor
[*] app_morsecode
[*] app_mp3
[*] app_nbscat
[*] app_originate
XXX app_osplookup
[*] app_page
[*] app_parkandannounce
[*] app_playback
[*] app_playtones
[*] app_privacy
[*] app_queue
[*] app_read
[*] app_readexten
[*] app_record
[ ] app_rpt
[ ] app_saycounted
[*] app_sayunixtime
[*] app_senddtmf
[*] app_sendtext
[ ] app_skel
[*] app_sms
[*] app_softhangup
[*] app_speech_utils
[*] app_stack
... More ...
```

True Call Queueing

Can use: res\_monitor(M)  
DEPRECATED (Use: app\_superQ)

If the `<support_level>` tag is used, then the value of `extended` would cause the additional text of `** EXTENDED **` to be displayed.

```
*****
Asterisk Module and Build Option Selection
*****
```

Press 'h' for help.

```
[*] app_minivm
[*] app_mixmonitor
[*] app_morsecode
[*] app_mp3
[*] app_nbscat
[*] app_originate
XXX app_osplookup
[*] app_page
[*] app_parkandannounce
[*] app_playback
[*] app_playtones
[*] app_privacy
[*] app_queue
[*] app_read
[*] app_readexten
[*] app_record
[ ] app_rpt
[ ] app_saycounted
[*] app_sayunixtime
[*] app_senddtmf
[*] app_sendtext
[ ] app_skel
[*] app_sms
[*] app_softhangup
[*] app_speech_utils
[*] app_stack
... More ...
```

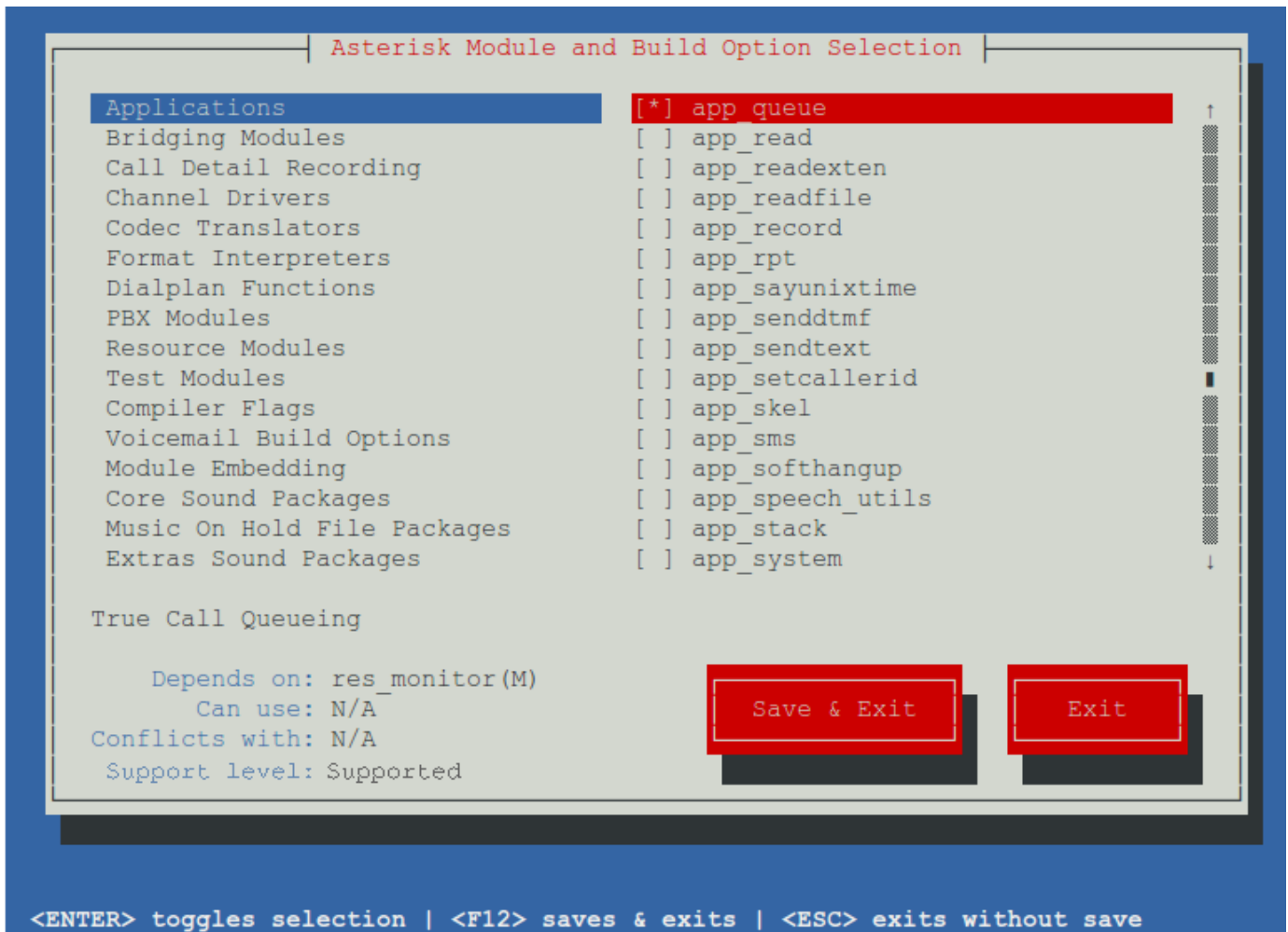
True Call Queueing

Can use: res\_monitor(M)

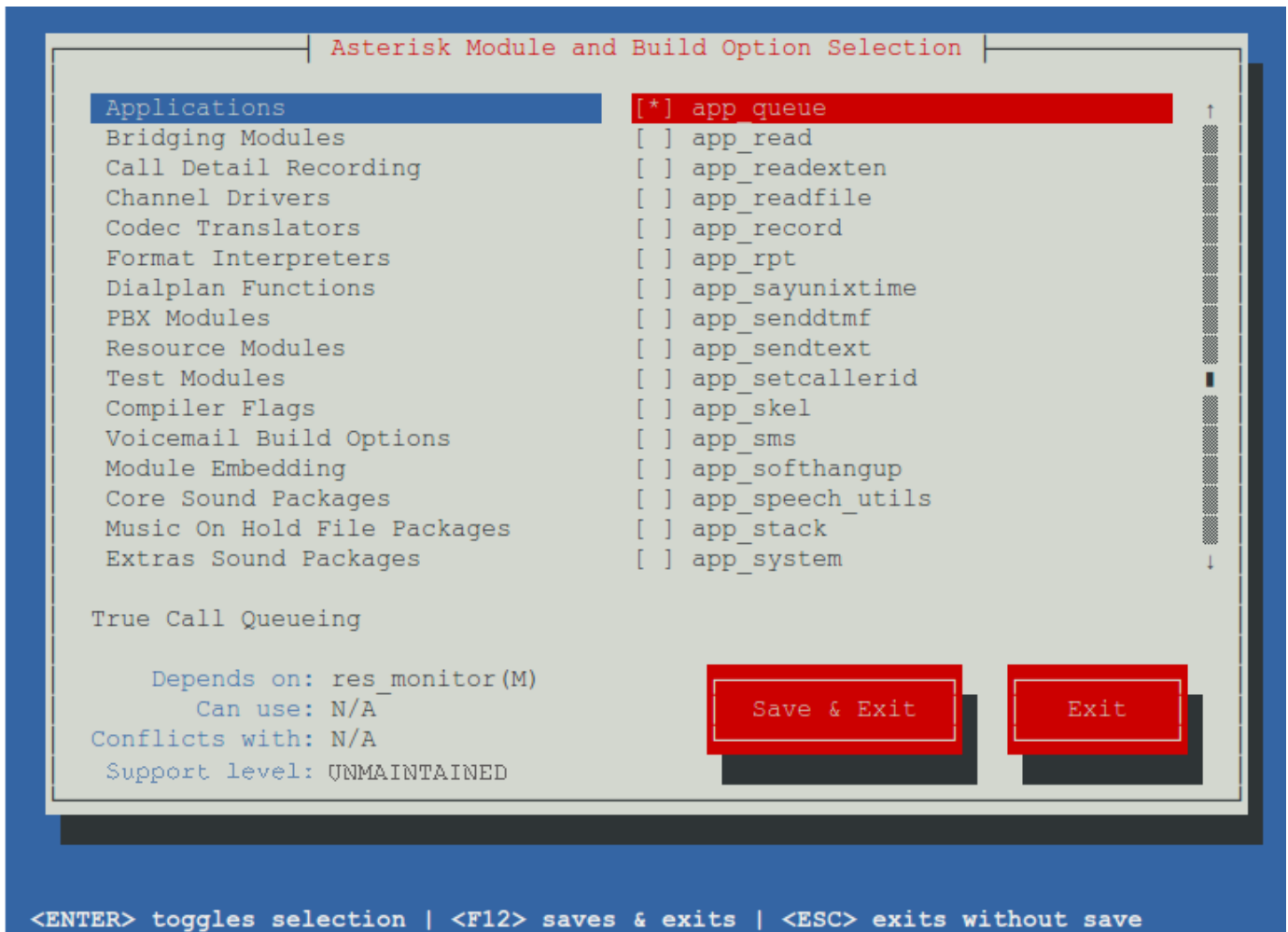
\*\*\* UNMAINTAINED \*\*\*

Display in menuselect-newt for supported modules. If no <support\_level> is specified, then it is assumed the module is supported:





Display in menuselect-newt for extended modules:



Display in menuselect-newt for deprecated modules:

Asterisk Module and Build Option Selection

Applications

Bridging Modules	[ ] app_read
Call Detail Recording	[ ] app_readexten
Channel Drivers	[ ] app_readfile
Codec Translators	[ ] app_record
Format Interpreters	[ ] app_rpt
Dialplan Functions	[ ] app_sayunixtime
PBX Modules	[ ] app_senddtmf
Resource Modules	[ ] app_sendtext
Test Modules	[ ] app_setcallerid
Compiler Flags	[ ] app_skel
Voicemail Build Options	[ ] app_sms
Module Embedding	[ ] app_softhangup
Core Sound Packages	[ ] app_speech_utils
Music On Hold File Packages	[ ] app_stack
Extras Sound Packages	[ ] app_system

True Call Queueing

Depends on: res\_monitor(M)

Can use: N/A

Conflicts with: N/A

Support level: DEPRECATED (app\_superQ)

Save & Exit

Exit

<ENTER> toggles selection | <F12> saves & exits | <ESC> exits without save

## Community Services Signup

The [asterisk.org](https://asterisk.org) community services use a centralized authentication system to manage user accounts. This means that the account that you use to log into [issues.asterisk.org](https://issues.asterisk.org) is the same account you can use to access [wiki.asterisk.org](https://wiki.asterisk.org).

To create an asterisk.org account, visit [signup.asterisk.org](https://signup.asterisk.org) and fill out the form.

The community services that currently support this centralized authentication are:

- [issues.asterisk.org](https://issues.asterisk.org)
- [wiki.asterisk.org](https://wiki.asterisk.org)
- [code.asterisk.org](https://code.asterisk.org)

Over time, accounts on our other community services will be moved to the same centralized authentication system.

# IRC

## IRC

Use <http://www.freenode.net> IRC server to connect with Asterisk developers and users in realtime.

### Asterisk Users Room

### Asterisk Developers Room

# Mailing Lists

There are several [mailing lists](#) where community members can go do discuss Asterisk. Some of the key lists are:

List	Description
<a href="#">asterisk-addons-commits</a>	SVN commits to the Asterisk addons project
<a href="#">asterisk-announce</a>	Asterisk releases and community service announcements
<a href="#">asterisk-biz</a>	Commercial and Business-Oriented Asterisk Discussion
<a href="#">Asterisk-BSD</a>	Asterisk on BSD discussion
<a href="#">asterisk-bugs</a>	Bugs
<a href="#">asterisk-commits</a>	SVN commits to the Asterisk project
<a href="#">asterisk-dev</a>	Discussions about the development of Asterisk. <a href="#">#Development List Note</a>
<a href="#">asterisk-app-dev</a>	Discussions about the development of applications that use Asterisk. <a href="#">#Development List Note</a>
<a href="#">asterisk-doc</a>	Discussions regarding The Asterisk Documentation Project
<a href="#">asterisk-embedded</a>	Asterisk Embedded Development
<a href="#">asterisk-gui</a>	Asterisk GUI project discussion
<a href="#">asterisk-gui-commits</a>	SVN commits to the Asterisk-GUI project
<a href="#">asterisk-ha-clustering</a>	Asterisk High Availability and Clustering List - Non-Commercial Discussion
<a href="#">Asterisk-i18n</a>	Discussion of Asterisk internationalization
<a href="#">asterisk-security</a>	Asterisk Security Discussion
<a href="#">asterisk-speech-rec</a>	Use of speech recognition in Asterisk
<a href="#">asterisk-users</a>	Discussions about the use and configuration of Asterisk.
<a href="#">asterisk-video</a>	Development discussion of video media support in Asterisk
<a href="#">asterisk-wiki-changes</a>	Changes to the Asterisk space on <a href="#">wiki.asterisk.org</a>
<a href="#">asterisknow</a>	AsteriskNOW Discussion
<a href="#">svn-commits</a>	SVN commits to the Digium repositories
<a href="#">Test-results</a>	Results from automated testing

## Development List Note

The [asterisk-dev](#) list is geared toward the development of Asterisk itself. For discussions about developing applications that use Asterisk, please see the [asterisk-app-dev](#) list. For discussions about deploying or using Asterisk, please see the [asterisk-users](#) mailing list.

# Wiki Organization and Style Guide

## Overview

This page should serve as a guide for creating organized, consistent, easy to consume content on the Asterisk wiki. It covers organization, content design and formatting.

You can find research all over the web on why these goals are important. Here's a link to [Nielsen Norman Group on How Users Read on the Web](#)

## Organization of Pages

### Spaces, parent and child pages

All Asterisk documentation pages are contained within the Asterisk space. Most of those editing on [wiki.asterisk.org](#) will only have edit permissions within the Asterisk space.

Within the Asterisk space there are many top-level parent pages available. We've included some info about each below to help you decide where a page goes.

- [About the Project](#) - Any general information about the project. Licensing, History, What is Asterisk?
- [Getting Started](#) - Information relevant to completely new users, information on installation and how to get rolling into configuration and the rest of the documentation.
- [Operation](#) - Details concerning Asterisk's operation. That is, starting and stopping the Asterisk daemon, command line operation and other non-configuration tasks.
- [Fundamentals](#) - Basic, key and core concepts of Asterisk. Some of the most important foundational things to know about Asterisk.
- [Configuration](#) - How everything is configured. Where are the files? How do I use them? How do I program dialplan? How do I use the APIs?
- [Deployment](#) - Examples, tutorials, how-tos and recommendations for specific use-cases or scenarios of deployment. How do I deal with Asterisk in a NATed environment? How do I build a simple PBX?
- [Development](#) - All information regarding the development of Asterisk itself.
- [Asterisk Test Suite Documentation](#) - Documentation for the test suite! Primarily for developers.
- [Asterisk Security Vulnerabilities](#) - How the project handles security vulnerabilities.
- [Asterisk Community](#) - Anything falling under community. The places we meet, our code of conduct, community services.

Plus there are a few sections titled "Asterisk X Documentation" where X is the version. These sections are reserved for command reference content, mostly pushed to the wiki via scripts that pull from the source documentation.



Pages created or modified by "wikibot" are generated by scripts.

### On this Page

- [Overview](#)
- [Organization of Pages](#)
  - [Spaces, parent and child pages](#)
  - [Considerations before creating new content](#)
- [Content Design](#)
  - [Design the content for a particular audience](#)
  - [Style based on purpose of content](#)
  - [Grammar, Punctuation, Spelling](#)
- [Content Formatting and Markup](#)
  - [Naming a page](#)
  - [Headings](#)
  - [Linking](#)
  - [Common macros](#)

## Considerations before creating new content

Find out if a new page is really needed. Check the following:

- Have you searched the Asterisk space to see if it exists already?
- Is there already *some* content that you could use to build on?
- Is there related content which should be consolidated with your new content?

If obsolete related content exists and you decide to delete, rename or move it, then check the following:

- [check for inbound links](#). If there are many referrers from external sources, you should ask the Asterisk project manager and others before renaming or deleting that page.
- If there are inbound links from [wiki.asterisk.org](#) pages, then we can of course go update those. For links from external sources, we have to decide how much we value breaking or not breaking those links.

Will the content fill more than a single screen-worth of space? If so, then you probably want to split it into multiple pages, or one page with sub-pages. When linking on a forum or mailing list, this makes it easier to link to specific chunks of content with shorter URLs.

Use the General Asterisk Wiki Page template for creating a new page with a very basic outline. It should include the Table of Contents macro discussed later on. The template is listed in the templates dialog when creating a new page.

## Content Design

### Design the content for a particular audience

**Consider the audience** for the content you are writing. A good clue is the section of the wiki you are placing the article in.

For example, pages in the Getting Started section should be oriented towards a completely new Asterisk user. That means the content should have a very narrow focus, making sure to explain concepts discussed and if a tutorial, provide small manageable chunks of tasks.

While it is acceptable to add content for very experienced Asterisk users, on the other end of the scale we do want to limit ourselves. Though Asterisk runs on Linux, the purpose of this wiki is not to teach Linux fundamentals. Pages in the Getting Started section do set the expectation that you should be familiar with Linux to use Asterisk.

**Set expectations.** In a section like Deployment, which could contain content for new users or experienced administrators, you should state requirements or experience at the beginning of the content.

### Style based on purpose of content

As the heading says, the purpose of your content should determine what style you craft it in.

- **Explanatory or Descriptive** - These styles are useful for reference material. Pages describing configuration sections, (as opposed to how-to configure) and feature or functionality descriptions all should be written in these styles.
- **Procedural** - Configuration how-to, tutorials, examples and guides often include this style which often features lots of bulleted lists and step by step instructions.
- **Frequently Asked Questions** - Often a FAQ appears as a list of questions with their answers. Only use this style when there is no other better way to do it. You could format a lot of content this way, but it doesn't make sense most of the time.

## Grammar, Punctuation, Spelling

The Ubuntu documentation team has a good guide for [grammar, punctuation and spelling](#). Other than those rules, follow typical English language and writing conventions. It is always a good idea to have at least one or two other editors review your content for obvious problems.

## Content Formatting and Markup

### Naming a page

Page names should be concise. i.e., The simplest summary of the content within it or under it. That means taking into consideration sub-pages if the page in question is a top-level parent page.

Page names should be written in [Title Case](#).

The page name will be included in the URL the wiki generates, so try to avoid any special characters or symbols and keep it as short as possible.

### Headings

Good use of headings allows a reader to quickly break-down the content on the page to find what they need. Each heading should be a concise summary of the content under that heading.

Most pages won't use more than two or three levels of headings. h1 for sections, h2 for sub-sections, and h3 if you really need to break those sub-sections down further. If you find yourself getting into a lot of h3 or h4 headings, consider creating sub-pages rather than including so much nested content on a single page.

Use [Title Case](#) for h1 headings. For lower headings use only sentence case; that is, only use capitalization how you would in a typical sentence.

### Linking

As you review content that you have written, try to link words to other content relevant to what you are discussing. Especially if it would help the user



understand that topic.

There are a variety of ways to link to other pages and content. All of them are described in the confluence documentation, in [Working with Links](#).

When linking from an external site, outside Confluence, it is important to use a permalink. Here is an excerpt from the Confluence documentation:

The best way to link to a Confluence page from outside Confluence, for example on a website or in an email message, is to use the tiny link which is a permanent URL. This ensures that the link to the page is not broken if the page name changes.

To access the permanent URL for a page:

1. View the page you wish to link to.
2. Choose **Tools > Link to this page**.
3. Copy the **Tiny Link**.
4. Use the tiny link in your website or email message.

You do not need to use the tiny link to link to pages within your Confluence site. Confluence automatically updates links when you rename or move a page to another space.

## Common macros

Confluence provides a variety of macros serving all sorts of functions. The confluence documentation on [Working with Macros](#) is very helpful to understand their usage. Below we'll list some macros that are commonly used with notes on how to use them specific to the Asterisk wiki.

### Macro - Table of Contents

The TOC macro should be put at the top right-hand side of the page.

To make sure it aligns correctly you need to place a section macro, then two column macros inside the section. After that put a panel macro inside the second column and finally the TOC goes inside the panel. You can then edit the panel title to say something like "On this Page". For an example you can edit this page itself. Look at the very top and see how the TOC is laid out inside the the other macros.

Always set the TOC maxlevel to 2, as it is generally not necessary for a page-level TOC to have more detail.

### Macro - No Format

Often when pasting text from an Asterisk log or the CLI you can run into issues as characters may be interpreted by the WYSIWYG editor as markup. Use the No Format macro to include a block of text to which no formatting will be applied.

### Macro - Expand

This macro gives you a click-able link where upon being clicked provides a slide-out panel of content. Use this when you need to fit many large chunks of example code or terminal output into a small section. It'll prevent the code from taking up all the screen real-estate and then requiring the user to scroll through it all.

### Macro - Warning, Info, Note, Tip

Try to avoid using these call-out boxes too often as they can be visually distracting and a few too many can make a page noisy. They are most useful with sparing application.

- Info - Most tame visually. Use this when you want to call-out a specific, brief piece of info that isn't necessarily critical, but could be very helpful.
- Note - A bright yellow exclamation sign. Maybe you want to share a word of caution or advise about requirements. "This example only works with a feature added in 11.1.1!"
- Tip - A bright green check mark. Useful when you want to mention shortcuts to a process already described detail, or call out brief command-line trick related to your content.
- Warning - This is bright red and very eye-catching. Use when you need to call out a piece of information very critical that could cause major problems for the user if not read or understood.