Asterisk Administrator Guide

Asterisk Development Team <asteriskteam@digium.com>

Getting Started

A Beginners Guide to Asterisk. Herein, you will find content related to installing Asterisk and basic usage concepts.

Precursors, Background and Business

Discovering Asterisk

This section of the documentation attempts to explain at a high level what Asterisk is and does. It also attempts to provide primers on the key technical disciplines that are required to successfully create and manage Asterisk solutions. Much of the material in this section is optional and may be redundant for those with a background in communications application development. For the other 99.9875% of the population, this is good stuff. Read on...

Asterisk Concepts

Asterisk is a very large application that does many things. It can be somewhat difficult to understand, especially if you are new to communications technologies. In the next few chapters we will do our best to explain what Asterisk is, what it is not, and how it came to be this way. This section doesn't cover the technology so much as the concept. If you're already familiar with the function of a telephony engine, feel free to jump ahead to the next section.

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.

A Brief History of the Asterisk Project

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.

Rather than give in and take out a small business loan, Mark made a fateful 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 take 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.

Beginning Asterisk

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

Thank you for downloading AsteriskNOW. This Linux distribution has been carefully customized and tested with Asterisk, and installs all of the packages needed for its use. It is the officially recommended development and runtime platform for Asterisk and Digium hardware, including Digium phones.

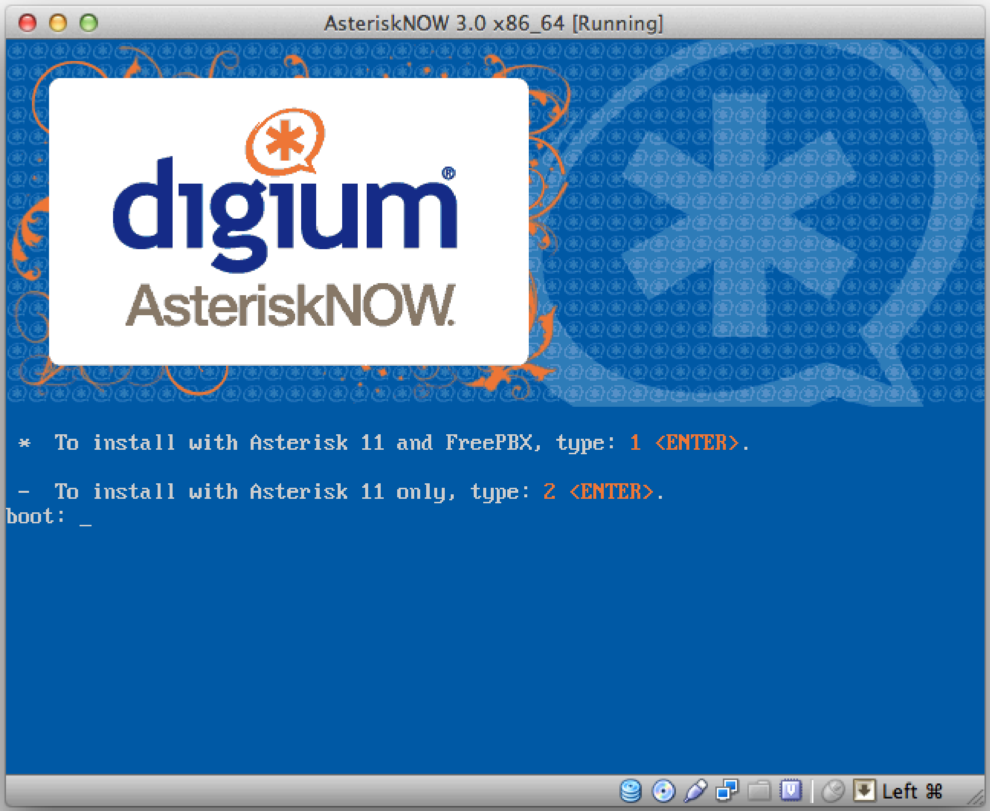
This guide provides a brief overview of installation, configuration, and maintenance of your system.

More information is available at http://wiki.centos.org/.

Please report any bugs at https://issues.asterisk.org/jira

Installation

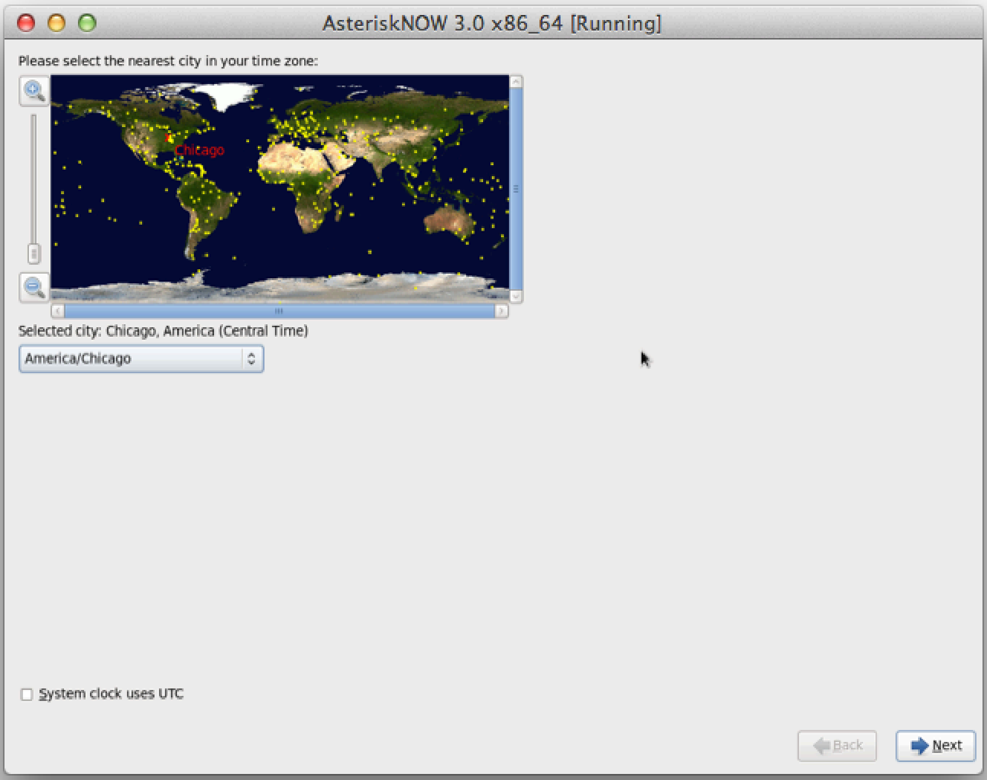
- Burn the AsteriskNOW DVD image to DVD disc and then boot from the DVD to begin the installation process.
  - If you are unfamiliar with burning disc images, the Ubuntu community has a great Burning ISO Howto available at https://help.ubuntu.com/community/BurningIsoHowto.
  - If you are unfamiliar with booting to DVD, the Ubuntu community has a wonderful Boot From DVD HOWTO available at https://help.ubuntu.com/community/BootFromCD.
- After booting from the AsteriskNOW DVD, you will be presented with the following screen and options for an installation with, or without the FreePBX web interface. This QuickStart assumes that the FreePBX web interface has been installed. To do this, selection option 1 and press <ENTER>:



```
 *  To install with Asterisk 11 and FreePBX, type: 1 <ENTER>.

 -  To install with Asterisk 11 only, type: 2 <ENTER>.
boot: _
```

This will begin the automated graphical installation process.

- During the installation, you are first presented with an option for setting the system Time Zone:



Please select the nearest city in your time zone:

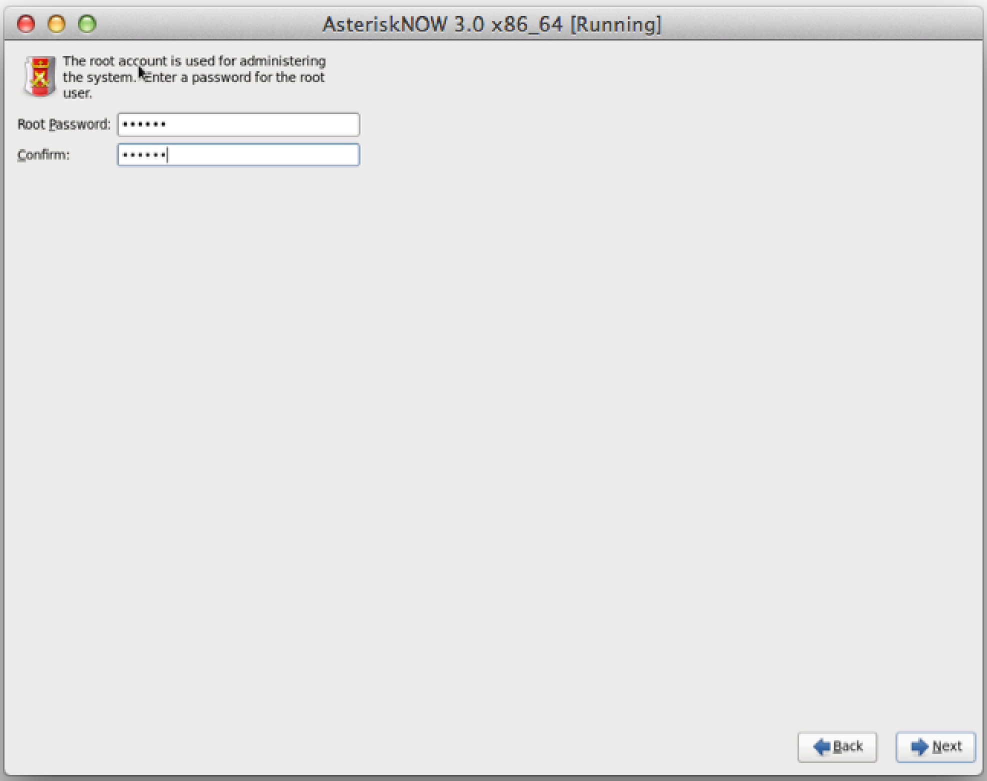
Selected city: Chicago, America (Central Time)

America/Chicago

☐ System clock uses UTC

** Choose the location that is nearest to you and move to the next screen.

- Next, you will be prompted to set a root password:


AsteriskNOW 3.0 x86_64 [Running]

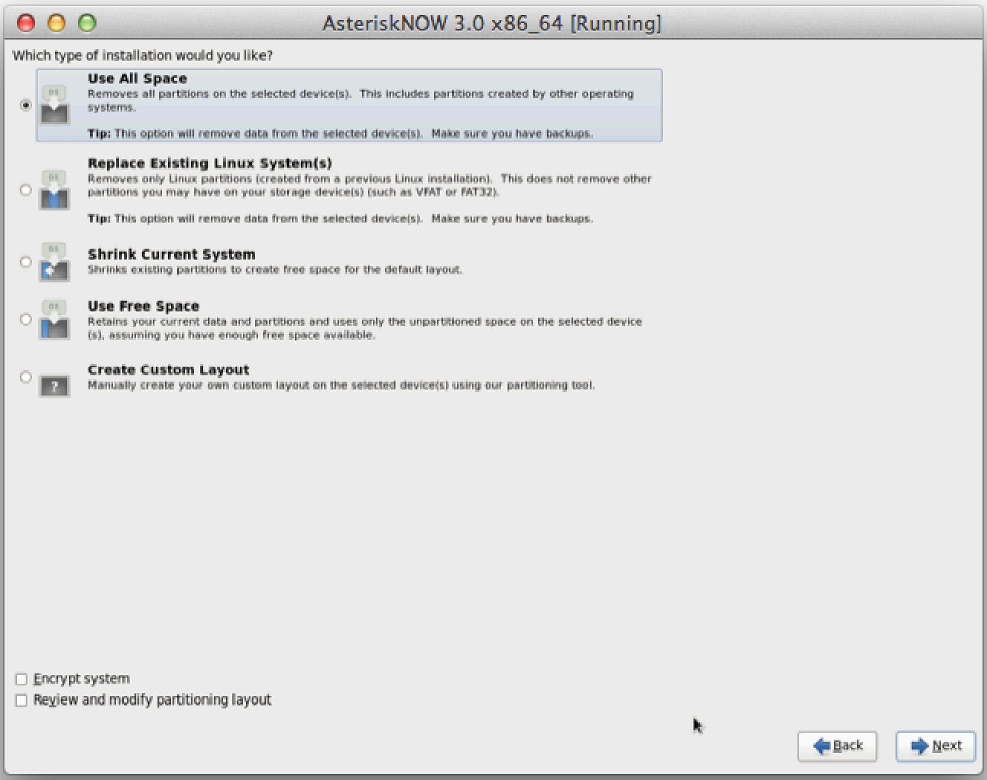
The root account is used for administering the system. Enter a password for the root user.

Root Password: ••••••

Confirm: ••••••

The 'root' user is the administrative account for Linux systems. Most system configuration requires 'root' access. If this password is lost, it is impossible to recover. It is recommend that your password contain a mix of lowercase and UPPERCASE letters, numbers, and/or symbols. Or, if you're into entropy, try a pass phrase.

- Then, you will choose your Hard Disk Layout:


AsteriskNOW 3.0 x86_64 [Running]

Which type of installation would you like?

● **Use All Space**
Removes all partitions on the selected device(s). This includes partitions created by other operating systems.

Tip: This option will remove data from the selected device(s). Make sure you have backups.

○ **Replace Existing Linux System(s)**
Removes only Linux partitions (created from a previous Linux installation). This does not remove other partitions you may have on your storage device(s) (such as VFAT or FAT32).

Tip: This option will remove data from the selected device(s). Make sure you have backups.

○ **Shrink Current System**
Shrinks existing partitions to create free space for the default layout.

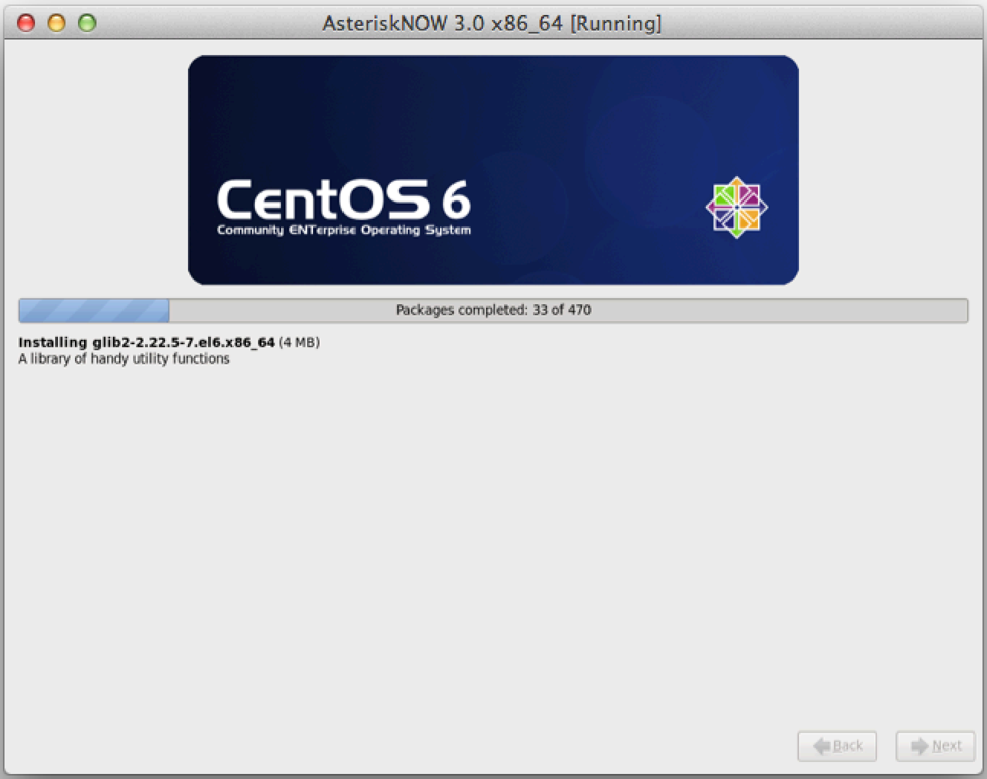
○ **Use Free Space**
Retains your current data and partitions and uses only the unpartitioned space on the selected device (s), assuming you have enough free space available.

○ **Create Custom Layout**
Manually create your own custom layout on the selected device(s) using our partitioning tool.

☐ Encrypt system
☐ Review and modify partitioning layout

It is recommended to select "Use All Space" and move to the next screen.

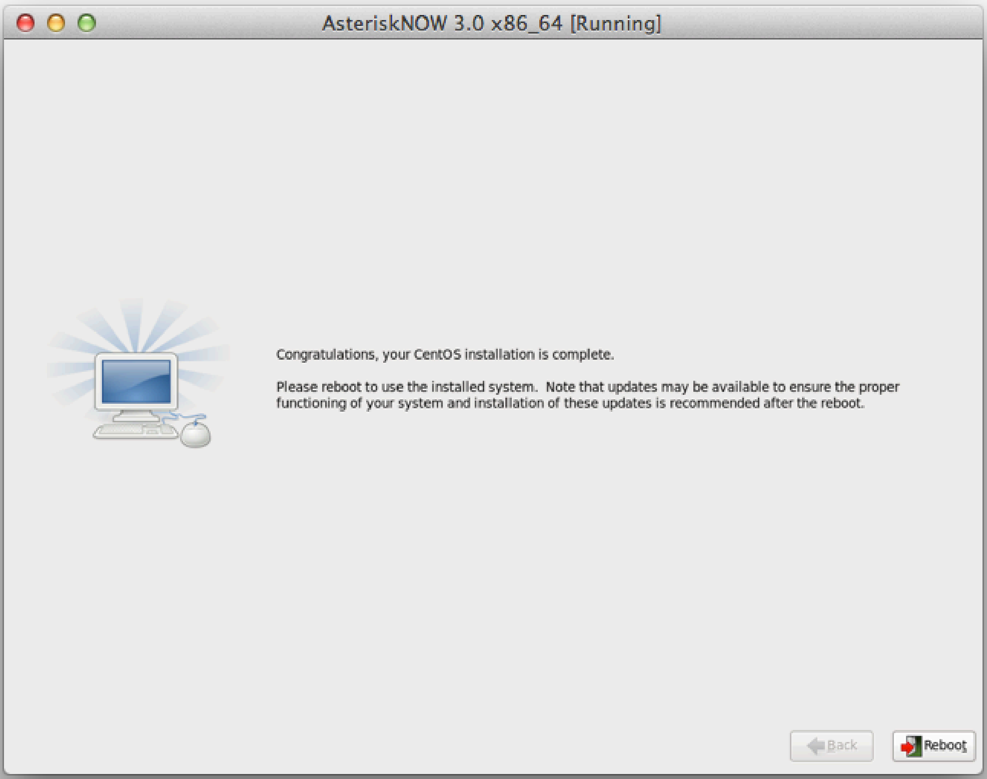
- Now, sit back, relax, have a cup of coffee and wait while the system is installed. This will take approximately 15-30 minutes. You will see a progress bar indicating the installation status.


AsteriskNOW 3.0 x86_64 [Running]
CentOS 6
Community ENTerprise Operating System
Back
Next

Packages completed: 33 of 470

Installing glib2-2.22.5-7.el6.x86_64 (4 MB)
A library of handy utility functions

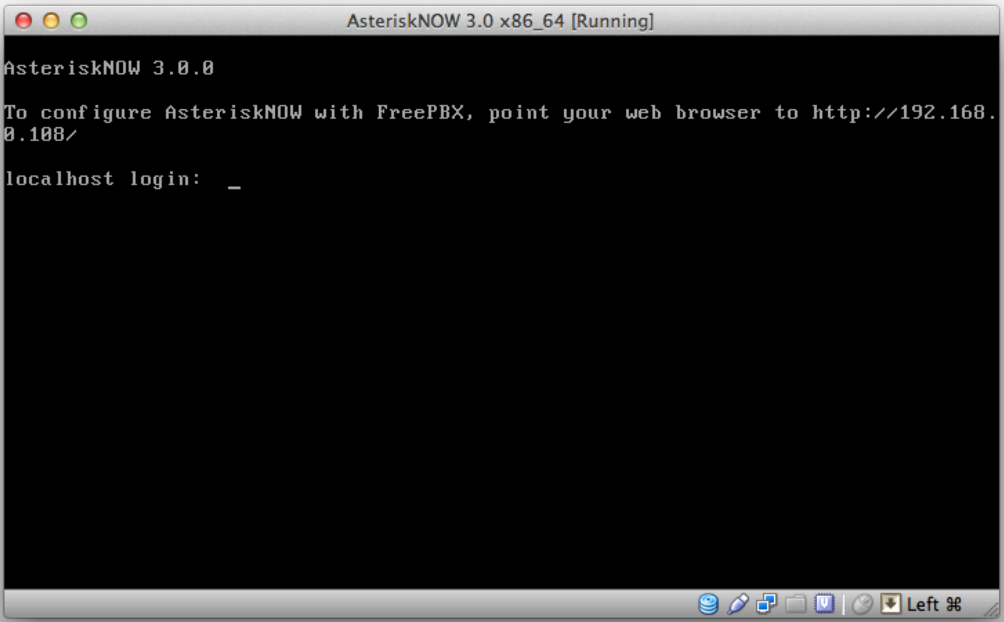
Once installation has completed, you will be prompted to reboot into your installation:



Congratulations, your CentOS installation is complete.

Please reboot to use the installed system. Note that updates may be available to ensure the proper functioning of your system and installation of these updates is recommended after the reboot.

- After the system reboots you will see this screen:

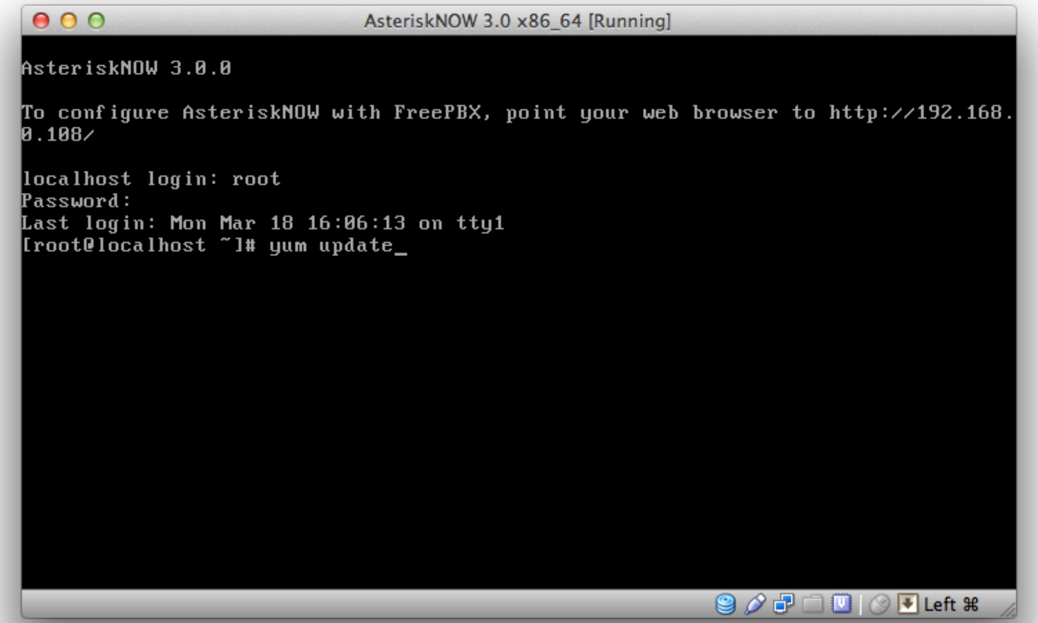


```
AsteriskNOW 3.0.0

To configure AsteriskNOW with FreePBX, point your web browser to http://192.168.
0.108/

localhost login:  _
```

Congratulations! You have successfully installed AsteriskNOW.

Notice the text that says "To configure AsteriskNOW with FreePBX, point your web browser to http://xx.xx.xx.xx/." Write this down, you will need it in the next section.

- Now, before you move on, it is important to update your AsteriskNOW system to the latest Linux packages. To do this, use the yum utility "yum." Perform a "yum update"



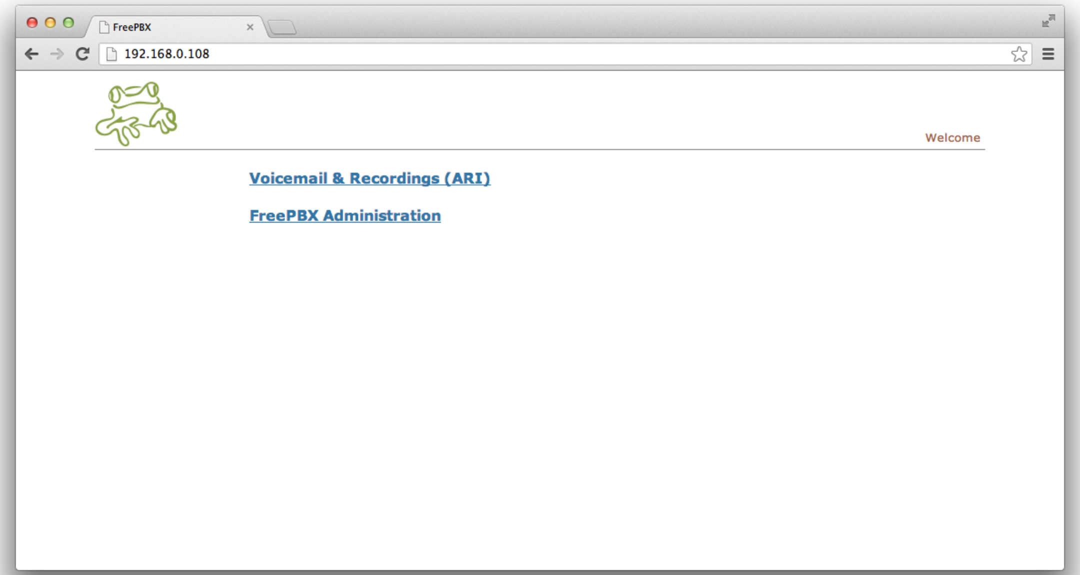
```
AsteriskNOW 3.0.0

To configure AsteriskNOW with FreePBX, point your web browser to http://192.168.
0.108/

localhost login: root
Password:
Last login: Mon Mar 18 16:06:13 on tty1
[root@localhost ~]# yum update_
```

** If new packages are available for installation, the utility will ask permission to install them. And, if the utility has not been run before, it may ask permission to accept a yum key. You should accept both to stay up to date.

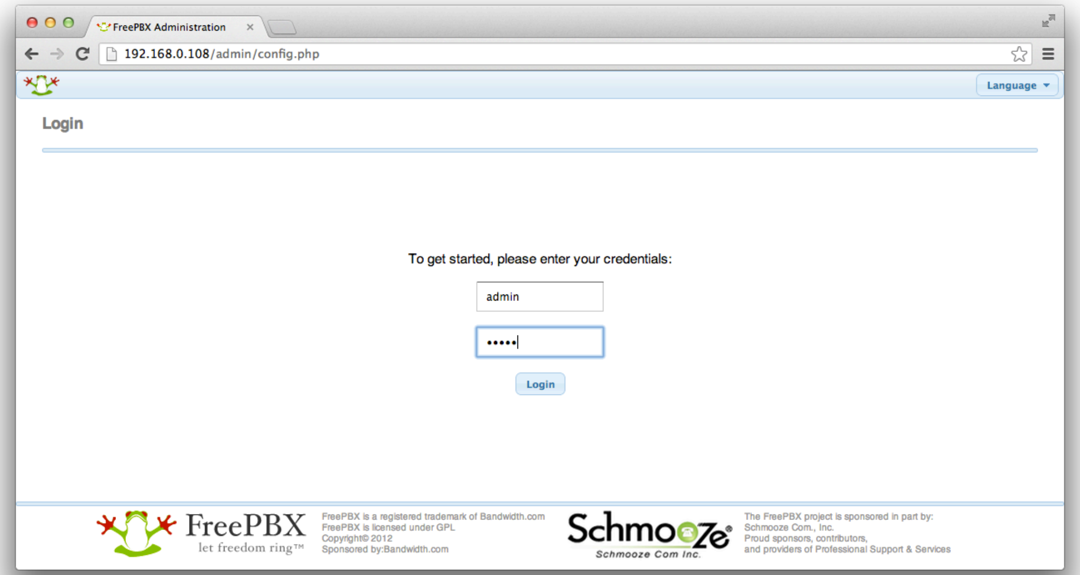
- You are now ready to move on to configuration of AsteriskNOW from the FreePBX web interface.

FreePBX Configuration

- To configure your system using FreePBX, open a web browser on another PC to the address specified during boot, e.g. "To configure AsteriskNOW with FreePBX, point your web browser to http://xx.xx.xx.xx/. If successful, you will be presented with the FreePBX main screen:

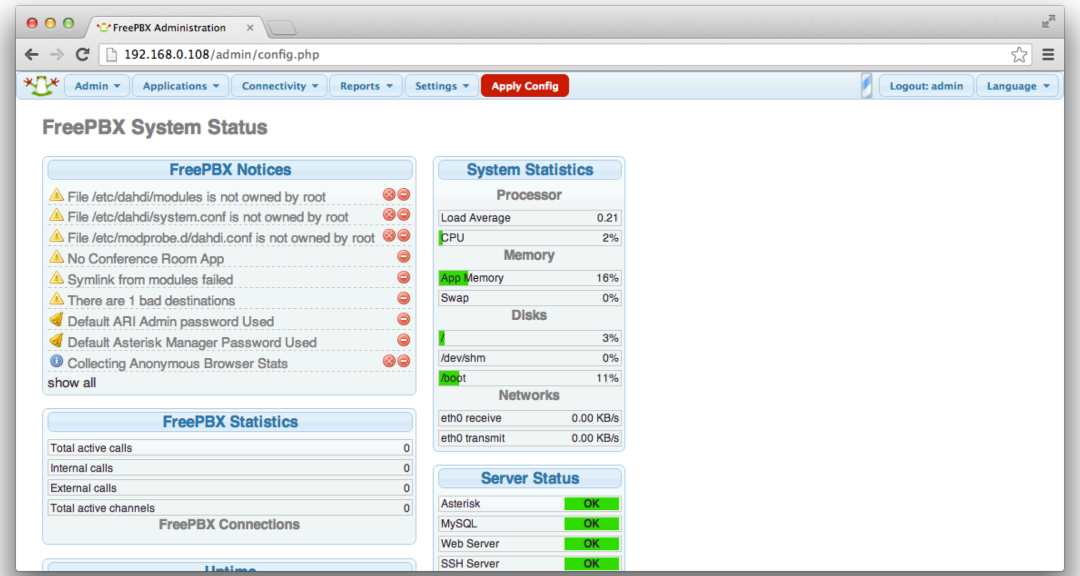


- From here, we want the "FreePBX Administration" link. Click it, and you will see the FreePBX login screen:


FreePBX Administration
192.168.0.108/admin/config.php
Language
Login
To get started, please enter your credentials:
admin
Login
FreePBX
let freedom ring™
FreePBX is a registered trademark of Bandwidth.com
FreePBX is licensed under GPL
Copyright© 2012
Sponsored by:Bandwidth.com
Schmooze
Schmooze Com Inc.
The FreePBX project is sponsored in part by:
Schmooze Com., Inc.
Proud sponsors, contributors,
and providers of Professional Support & Services

The default username is **admin**
The default password is **admin**

- Having successfully logged into FreePBX, you will see the FreePBX dashboard:



FreePBX System Status

FreePBX Notices
- File /etc/dahdi/modules is not owned by root
- File /etc/dahdi/system.conf is not owned by root
- File /etc/modprobe.d/dahdi.conf is not owned by root
- No Conference Room App
- Symlink from modules failed
- There are 1 bad destinations
- Default ARI Admin password Used
- Default Asterisk Manager Password Used
- Collecting Anonymous Browser Stats

show all

FreePBX Statistics

Total active calls	0
Internal calls	0
External calls	0
Total active channels	0

FreePBX Connections

Uptime

System Statistics

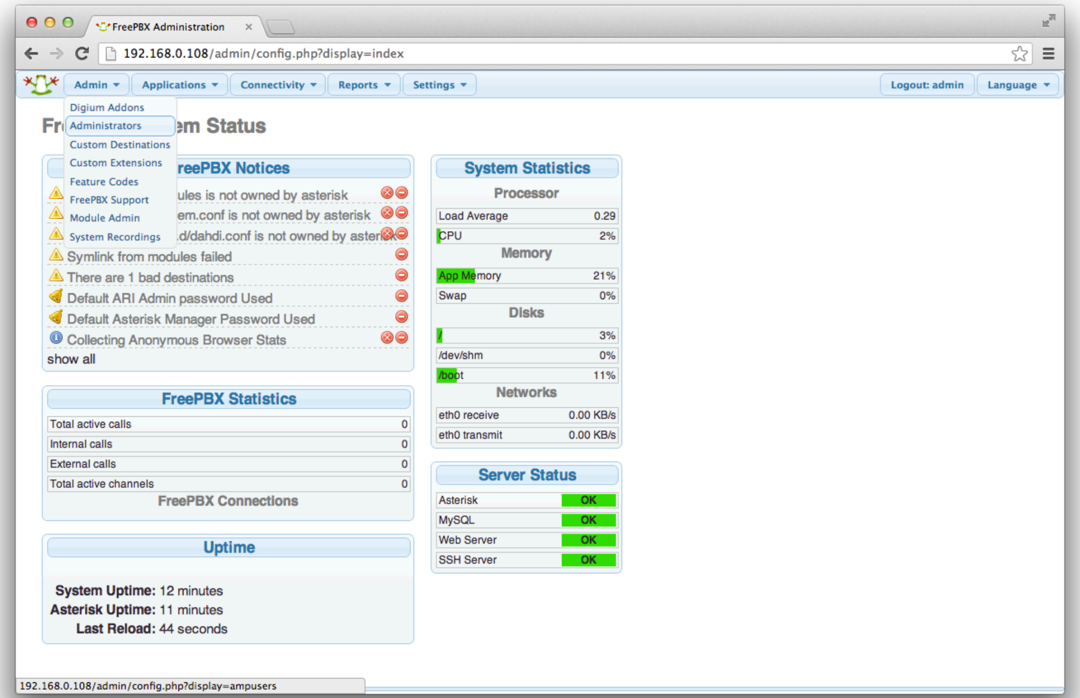
Processor	
Load Average	0.21
CPU	2%
Memory	
App Memory	16%
Swap	0%
Disks	
/	3%
/dev/shm	0%
/boot	11%
Networks	
eth0 receive	0.00 KB/s
eth0 transmit	0.00 KB/s

Server Status

Asterisk	OK
MySQL	OK
Web Server	OK
SSH Server	OK

Notice the **Red** reload button. It will appear after changes are made to any page. If you see it, it should be clicked, it will affect any changes on the system that FreePBX needs to make. This guide assumes that whenever you see it, you will click it.

- Next, we will change the default admin password. This is imperative! Failure to do this is inviting disaster. The importance of doing this **C ANNOT be understated.**
  - First, visit the Admin tool



Fr…em Status

…reePBX Notices
- …ules is not owned by asterisk
- …em.conf is not owned by asterisk
- …d/dahdi.conf is not owned by asterisk
- Symlink from modules failed
- There are 1 bad destinations
- Default ARI Admin password Used
- Default Asterisk Manager Password Used
- Collecting Anonymous Browser Stats

show all

FreePBX Statistics

Total active calls	0
Internal calls	0
External calls	0
Total active channels	0

FreePBX Connections

Uptime

System Uptime: 12 minutes
Asterisk Uptime: 11 minutes
Last Reload: 44 seconds

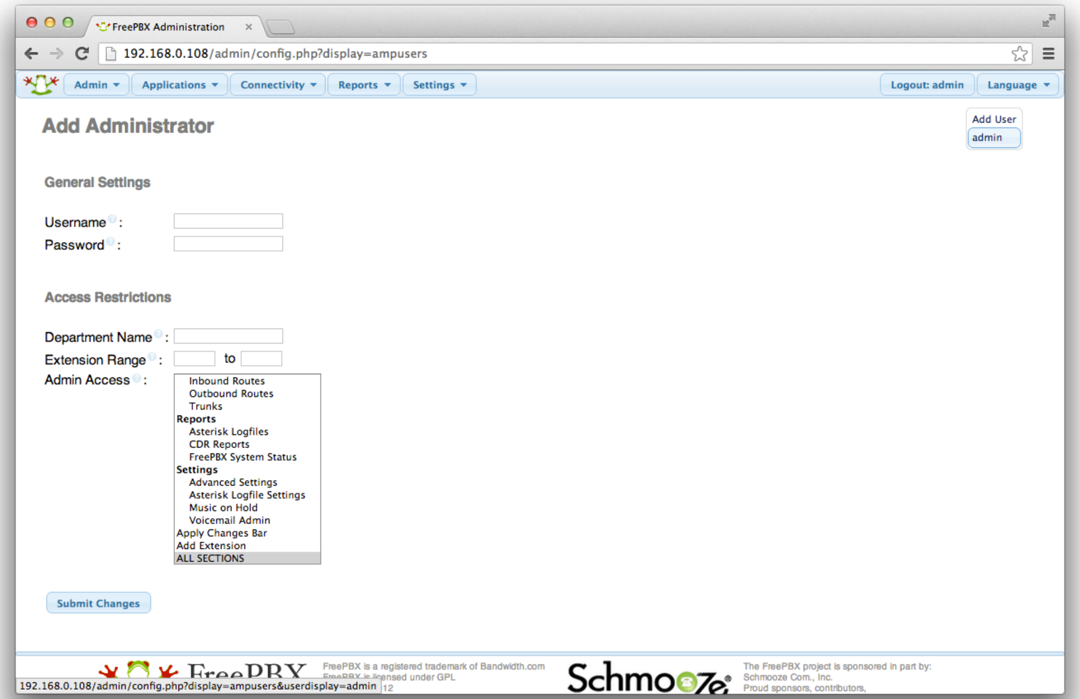
System Statistics

Processor	
Load Average	0.29
CPU	2%
Memory	
App Memory	21%
Swap	0%
Disks	
/	3%
/dev/shm	0%
/boot	11%
Networks	
eth0 receive	0.00 KB/s
eth0 transmit	0.00 KB/s

Server Status

Asterisk	OK
MySQL	OK
Web Server	OK
SSH Server	OK

  - Next, select admin from the right column



192.168.0.108/admin/config.php?display=ampusers

Add Administrator

General Settings

Username :
Password :

Access Restrictions

Department Name :
Extension Range : to
Admin Access :

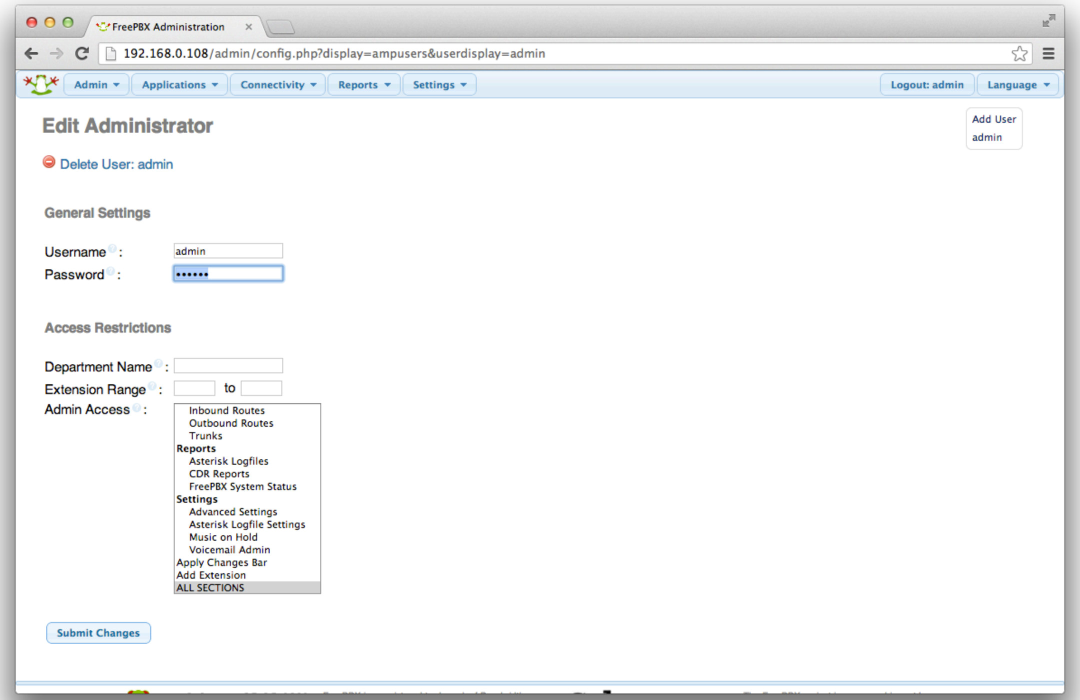
Inbound Routes
Outbound Routes
Trunks
Reports
Asterisk Logfiles
CDR Reports
FreePBX System Status
Settings
Advanced Settings
Asterisk Logfile Settings
Music on Hold
Voicemail Admin
Apply Changes Bar
Add Extension
ALL SECTIONS

FreePBX is a registered trademark of Bandwidth.com
FreePBX is licensed under GPL

The FreePBX project is sponsored in part by:
Schmooze Com., Inc.
Proud sponsors, contributors,

192.168.0.108/admin/config.php?display=ampusers&userdisplay=admin

- Then, change the admin password



192.168.0.108/admin/config.php?display=ampusers&userdisplay=admin

Edit Administrator

Delete User: admin

General Settings

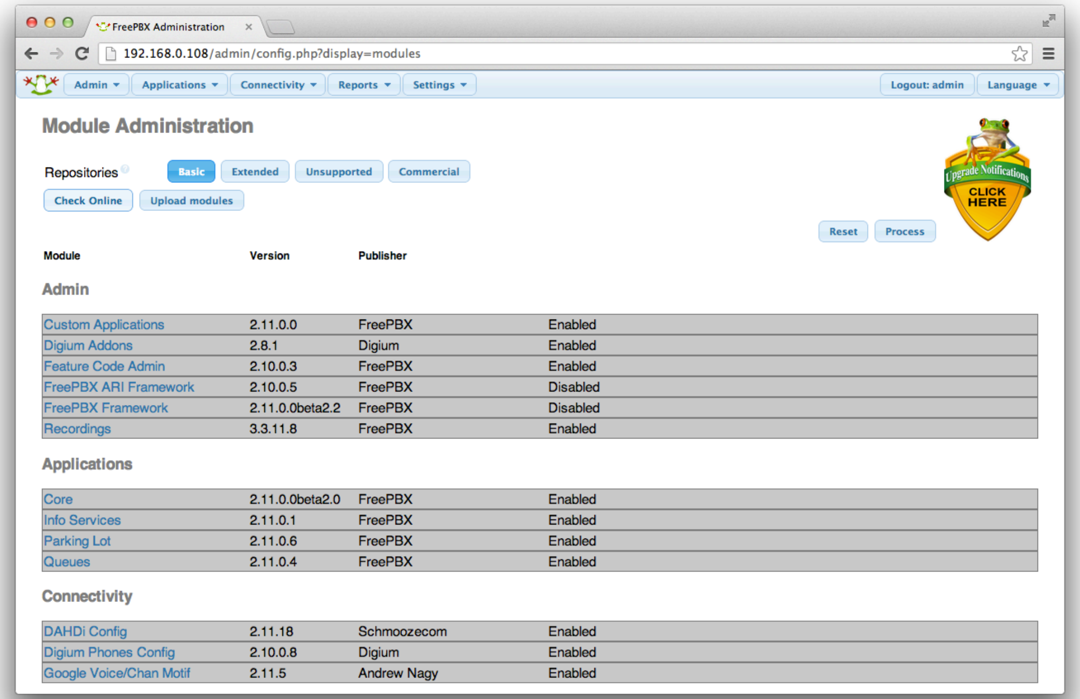
Username : admin
Password : ••••••

Access Restrictions

Department Name :
Extension Range : to
Admin Access :

Inbound Routes
Outbound Routes
Trunks
Reports
Asterisk Logfiles
CDR Reports
FreePBX System Status
Settings
Advanced Settings
Asterisk Logfile Settings
Music on Hold
Voicemail Admin
Apply Changes Bar
Add Extension
ALL SECTIONS

- Finally, one should update any out of date modules on the system. To do this, we will visit the Module Admin tool:



Module Administration

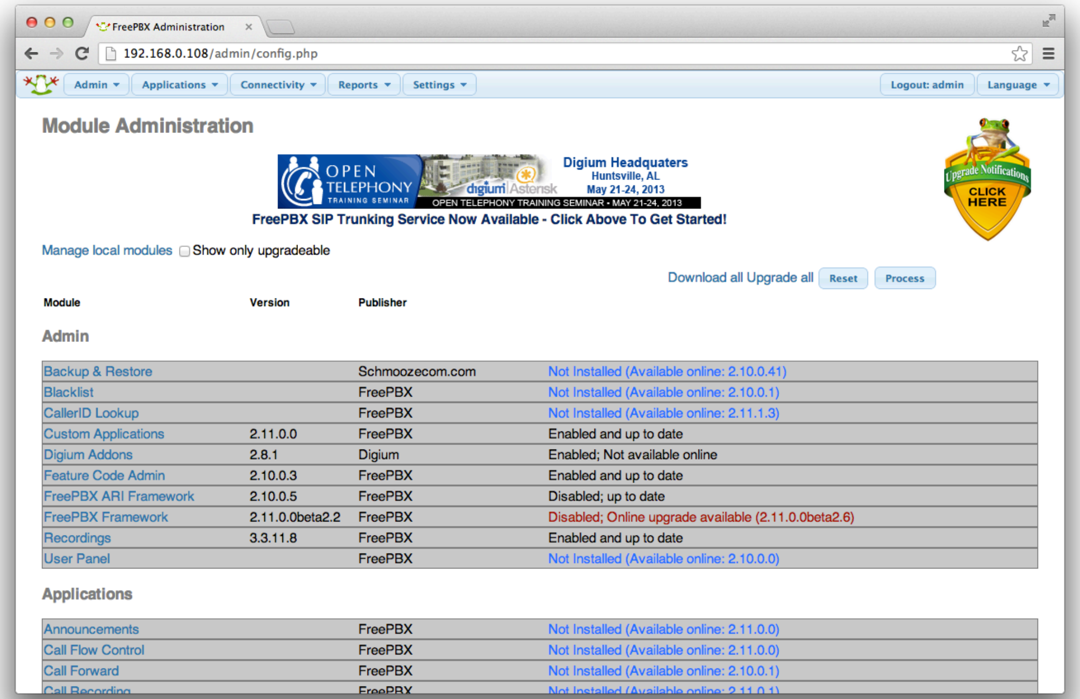
Repositories: Basic | Extended | Unsupported | Commercial

Check Online | Upload modules

Reset | Process

Module	Version	Publisher	
Admin			
Custom Applications	2.11.0.0	FreePBX	Enabled
Digium Addons	2.8.1	Digium	Enabled
Feature Code Admin	2.10.0.3	FreePBX	Enabled
FreePBX ARI Framework	2.10.0.5	FreePBX	Disabled
FreePBX Framework	2.11.0.0beta2.2	FreePBX	Disabled
Recordings	3.3.11.8	FreePBX	Enabled
Applications			
Core	2.11.0.0beta2.0	FreePBX	Enabled
Info Services	2.11.0.1	FreePBX	Enabled
Parking Lot	2.11.0.6	FreePBX	Enabled
Queues	2.11.0.4	FreePBX	Enabled
Connectivity			
DAHDi Config	2.11.18	Schmoozecom	Enabled
Digium Phones Config	2.10.0.8	Digium	Enabled
Google Voice/Chan Motif	2.11.5	Andrew Nagy	Enabled

- Click the Check Online button and you will see any out of date modules



Module Administration

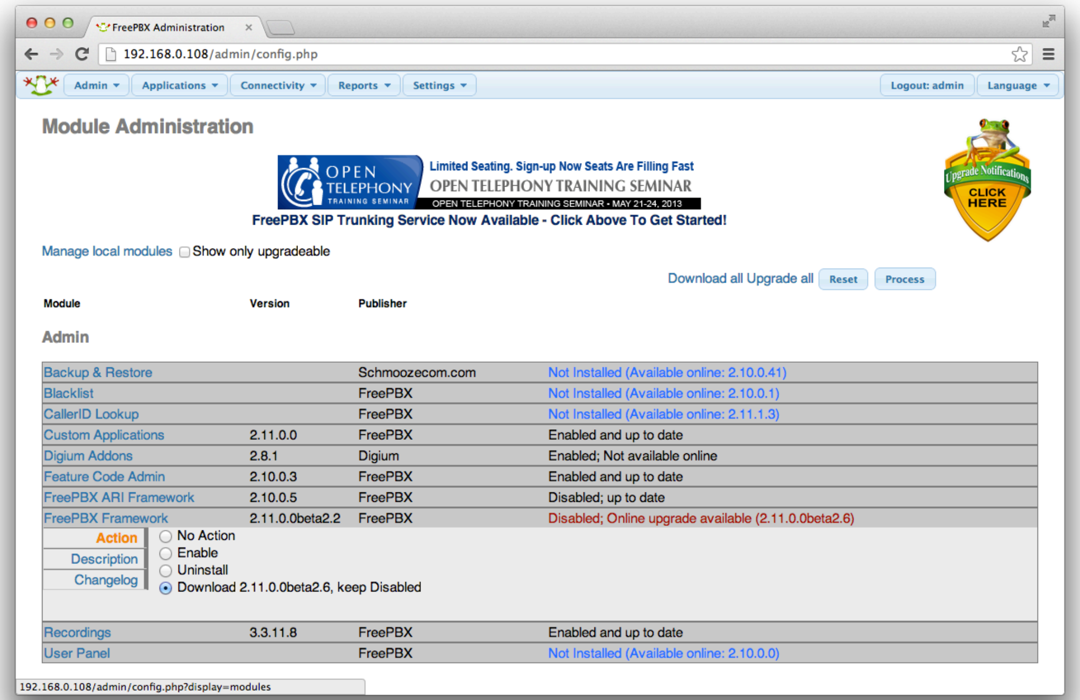
FreePBX SIP Trunking Service Now Available - Click Above To Get Started!

Manage local modules ☐ Show only upgradeable

Download all Upgrade all | Reset | Process

Module	Version	Publisher	
Admin			
Backup & Restore		Schmoozecom.com	Not Installed (Available online: 2.10.0.41)
Blacklist		FreePBX	Not Installed (Available online: 2.10.0.1)
CallerID Lookup		FreePBX	Not Installed (Available online: 2.11.1.3)
Custom Applications	2.11.0.0	FreePBX	Enabled and up to date
Digium Addons	2.8.1	Digium	Enabled; Not available online
Feature Code Admin	2.10.0.3	FreePBX	Enabled and up to date
FreePBX ARI Framework	2.10.0.5	FreePBX	Disabled; up to date
FreePBX Framework	2.11.0.0beta2.2	FreePBX	Disabled; Online upgrade available (2.11.0.0beta2.6)
Recordings	3.3.11.8	FreePBX	Enabled and up to date
User Panel		FreePBX	Not Installed (Available online: 2.10.0.0)
Applications			
Announcements		FreePBX	Not Installed (Available online: 2.11.0.0)
Call Flow Control		FreePBX	Not Installed (Available online: 2.11.0.0)
Call Forward		FreePBX	Not Installed (Available online: 2.10.0.1)
Call Recording		FreePBX	Not Installed (Available online: 2.11.0.1)

- To update a module, click it, and then select the Download option


FreePBX Administration
192.168.0.108/admin/config.php
Admin
Applications
Connectivity
Reports
Settings
Logout: admin
Language
Limited Seating. Sign-up Now Seats Are Filling Fast
OPEN TELEPHONY TRAINING SEMINAR
OPEN TELEPHONY TRAINING SEMINAR • MAY 21-24, 2013
Upgrade Notifications
CLICK HERE
Manage local modules
Show only upgradeable
Download all Upgrade all
Reset
Process
192.168.0.108/admin/config.php?display=modules

Module Administration

FreePBX SIP Trunking Service Now Available - Click Above To Get Started!

Admin

Module	Version	Publisher	
Backup & Restore		Schmoozecom.com	Not Installed (Available online: 2.10.0.41)
Blacklist		FreePBX	Not Installed (Available online: 2.10.0.1)
CallerID Lookup		FreePBX	Not Installed (Available online: 2.11.1.3)
Custom Applications	2.11.0.0	FreePBX	Enabled and up to date
Digium Addons	2.8.1	Digium	Enabled; Not available online
Feature Code Admin	2.10.0.3	FreePBX	Enabled and up to date
FreePBX ARI Framework	2.10.0.5	FreePBX	Disabled; up to date
FreePBX Framework	2.11.0.0beta2.2	FreePBX	Disabled; Online upgrade available (2.11.0.0beta2.6)
Recordings	3.3.11.8	FreePBX	Enabled and up to date
User Panel		FreePBX	Not Installed (Available online: 2.10.0.0)

Action / Description / Changelog

No Action / Enable / Uninstall / [✓ Download 2.11.0.0beta2.6, keep Disabled]

- Finally, press the Process button and follow the instructions to complete the module update.

Updating, Querying, Removing Packages

After completing installation of AsteriskNOW, all of the packages for running Asterisk are installed. However system updates are often available.

AsteriskNOW contains several yum repositories in addition to the ones provided by CentOS. These are asterisk-current/asterisk-tested and digium-current/digium-tested. The asterisk- repositories contain packages for Digium-provided Open Source software (such as Asterisk, libpri, and DAHDI). The digium- repositories contain non-free or commercial software (such as the Digium Phone module for Asterisk, G.729 for Asterisk, Fax For Asterisk, and the HPEC echo cancellation module). This allows you to install additional software and to stay up to date with the latest changes.

Packages can be installed or removed by using `yum install <package>` or `yum remove <package>`. Updates should be regularly installed by using `yum update`. For a very full list of available and installed packages, you can use `yum list | less`. For more information about Yum, visit http://yum.baseurl.org/wiki/YumCommands or type `man yum`.

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. Packages for Red Hat Enterprise Linux and CentOS are also available at http://packages.asterisk.org/ (see Asterisk Packages for instructions on use).

Validating Your AsteriskNOW 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. After logging in as the **root** user 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-1.6.X.Y]# lsmod | grep dahdi
```

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

```
[root@server asterisk-1.6.X.Y]# service dadhi 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 ~]# lsmod | grep dahdi

dahdi_dummy          4288   0

dahdi_transcode      7928   1 wctc4xxp

dahdi_voicebus       40464  2 wctdm24xxp,wcte12xp

dahdi                196544 12 dahdi_dummy,wctdm24xxp,wcte11xp,wct1xxp,wcte12xp,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 ~]# service asterisk status
asterisk is stopped
```

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

```
[root@server ~]# service 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 by using the command below. (The process identifier, or pid, will obviously be different on your system.)

```
[root@server ~]# service asterisk status
asterisk (pid 32117) is running...
```

And there you have it... you have an Asterisk system up and running! You should now continue on in Section 202. Getting Started with Asterisk.

Asterisk Configuration Files

Intro to Asterisk Configuration Files

In this section, we'll introduce you to the Asterisk configuration files, and show you how to use some advanced features.

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 section, 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. 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
```

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 and exec Constructs

There are two other constructs we can use within 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 **#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.

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.

Let's look at example of both constructs in action.

```
[section-name]
setting=true
#include otherconfig.conf      ; include another configuration file
#exec otherprogram             ; include output of otherprogram
```

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, place an exclamation mark in parentheses after the section heading, as shown in the example below.

```
[template-name](!)
setting=value
```

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

[section-name](template-name,template-2)
setting3=value3
```

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
```

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 experiement.

The Most Basic PBX

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 access to two 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. Soft phones 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.

Creating SIP Accounts

In order for our two 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 both the phones are using the SIP protocol, we'll configure accounts in the SIP channel driver configuration file, called **sip.conf**. (This file resides in the Asterisk configuration directory, which is typically **/etc/asterisk**.) Let's name your phones **Alice** and **Bob**, so that we can easily differentiate between them.

Open **sip.conf** with your favorite text editor, and spend a minute or two looking at the 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. Scroll to the bottom of the file, and add a section for Alice and Bob. You'll need to choose your own unique password for each account, and change the **permit** line to match the settings for your local network.

```
[demo-alice]
type=friend
host=dynamic
secret=verysecretpassword ; put a strong, unique password here instead
context=users
deny=0.0.0.0/0
permit=192.168.5.0/255.255.255.0 ; replace with your network settings

[demo-bob]
type=friend
host=dynamic
secret=othersecretpassword ; put a strong, unique password here instead
context=users
deny=0.0.0.0/0
permit=192.168.5.0/255.255.255.0 ; replace with your network settings
```

Be Serious About Account Security

We can't stress enough how important it is for you to pick a strong password for all accounts on Asterisk, and to only allow access from trusted networks. Unfortunately, we've found many instances of people exposing their Asterisk to the internet at large with easily-guessable passwords, or no passwords at all. You could be at risk of toll fraud, scams, and other malicious behavior.

For more information on Asterisk security and how you can protect yourself, check out http://www.asterisk.org/security/webinar/.

After adding the two sections above to your **sip.conf** file, go to the Asterisk command-line interface and run the **sip reload** command to tell Asterisk to re-read the **sip.conf** configuration file.

```
server*CLI> sip reload

Reloading SIP

server*CLI>
```

Reloading Configuration Files

Don't forget to reload the appropriate Asterisk configuration files after you have made changes to them.

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 square brackets 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.

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** colum n 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.

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'll cover contexts more in Dialplan Fundamentals, 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.

Go to the bottom of your **extensions.conf** file, and add a new context named [users].

Naming Your Dialplan Contexts

There's nothing special about the name **users** 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 for the users of our PBX system, we'll call our context **users**.

Underneath that context name, we'll create an extesion numbered **6001** which attempts to ring Alice's phone for twenty seconds, and an extension **6002** wh ich attempts to rings Bob's phone for twenty seconds.

```
[users]
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 **dial plan reload**. You can verify that Asterisk successfully read the configuration file by typing **dialplan show users** at the CLI.

```
server*CLI> dialplan show users
[ Context 'users' 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!

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@users: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 (users, 6002, 1) exited non-zero on 'SIP/demo-alice-00000000'
```

As you can see, Alice called extension **6002** in the [users] 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** at the Asterisk CLI.

To see which context your SIP phones will send calls to, type **sip show users**.

To ensure that you've created the extensions correctly in the **[users]** context in the dialplan, type **dialplan show users**.

To see which extension will be executed when you dial extension **6002**, type **dialplan show 6002@users**.

Sound Prompt Searching based on Channel Language

Dialplan Fundamentals

The dialplan is essential to the operation of any successful Asterisk system. In this module, we'll help you learn the fundamental components of the Asterisk dialplan, and how to combine them to begin scripting your own dialplan. We'll also add voice mail and a dial-by-name directory features to your dialplan.

Contexts, Extensions, and Priorities

The dialplan is organized into various 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. We'll use contexts to enforce security boundaries between the various parts of our dialplan, as well as to provide different classes of service to groups of users.

The syntax for a context is exactly the same as any other section heading in the configuration files, as explained in Section 206.2.1. Sections and Settings. Simply place the context name in square brackets. For example, here is the context we defined in the previous module:

```
[users]
```

Within each context, we can define one or more **extensions**. As explained in the previous module, 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...]])
```

As an example, let's review extension **6001** from the previous module. It looks like:

```
exten => 6001,1,Dial(SIP/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 **SIP/demo-alice** and **20**.
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 priorites one and two, but would then terminate the call, because it couldn't find priority number three.

Priority number can also be simplied 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,do something
exten => 6123,n,do something else
exten => 6123,n,do something different
```

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,do something
exten => 6123,n(repeat),do something else
exten => 6123,n,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 => _1NXXNXXXXXX,1,do something
same => n(repeat),do something else
same => n,do something different
```

Applications

Each priority in the dialplan calls an application. An application does 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. For a complete list of the dialplan applications available to your installation of Asterisk, type **core show applications** at the Asterisk CLI.

Most applications take one or more parameters, which provide additional information to the application or change its behavior. Parameters should be separated by commas.

Syntax for Parameters
You'll often find examples of Asterisk dialplan code online and in print which use the pipe character or vertical bar character (|) between parameters, as shown in this example:

```
exten => 6123,1,application(one|two|three)
```

This is a deprecated syntax, and will no longer work in newer versions of Asterisk. Simply replace the pipe character with a comma, like this:

```
exten => 6123,1,application(one,two,three)
```

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 endpoing 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, **Playba ck()** 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)
```

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, an 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

Exploring Sound Prompts

Asterisk comes with a wide variety of pre-recorded sound prompts. When you install Asterisk, you can choose to install both core and extra sound packages in several different file formats. Prompts are also available in several languages. To explore the sound files on your system, simply find the sounds directory (this will be **/var/lib/asterisk/sounds** on most systems) and look at the filenames. You'll find useful prompts ("Please enter the extension of the person you are looking for..."), as well as as a number of off-the-wall prompts (such as "Weasels have eaten our phone system", "The office has been overrun with iguanas", and "Try to spend your time on hold not thinking about a blue-eyed polar bear") as well.

Sound Prompt Formats
Sound prompts come in a variety of file formats, such as **.wav** and **.ulaw** files. When asked to play a sound prompt from disk, Asterisk plays the sound prompt with the file format that can most easily be converted to the CODEC of the current call. For example, if the inbound call is using the **alaw** CODEC and the sound prompt is available in **.gsm** and **.ulaw** format, Asterisk will play the **.ulaw** file because it requires fewer CPU cycles to transcode to the **alaw** CODEC.
You can type the command **core show translation** at the Asterisk CLI to see the transcoding times for various CODECs. The times reported (in Asterisk 1.6.0 and later releases) are the number of microseconds it takes Asterisk to transcode one second worth of audio. These times are calculated when Asterisk loads the codec modules, and often vary slightly from machine to machine. To perform a current calculation of translation times, you can type the command **core show translation recalc 60**.

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.

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. Place this extension in your **[docs:users]** context:

```
exten => 6000,1,Answer(500)
exten => 6000,n,Playback(hello-world)
exten => 6000,n,Hangup()
```

Dial Application

Now that you've learned the basics of using dialplan applications, let's take a closer look at the **Dial()** application that we used earlier in extensions **6001** an d **6002**. **Dial()** attempts to ring an external device, and if the call is answered it bridges the two channels together and does any necessary protocol or CODEC conversion. It also handles call progress responses (busy, no-answer, ringing).

Dial() and the Dialplan
Please note that if the Dial() application successfully bridges two channels together, that the call does not progress in the dialplan. The call will only continue on to the next priority if the Dial() application is unable to bridge the calling channel to the dialed device.

The **Dial()** application takes four parameters:

1. **Devices**
    - A list of the device(s) you want to call. Devices are specified as technology or channel driver, a forward slash, and the device or account name. For example, **SIP/demo-alice** would use the SIP channel driver to call the device specified in the **demo-alice** sec tion of **sip.conf**. Devices using the IAX2 channel driver take the form of **IAX2/demo-george**, and DAHDI channels take the form of **DAHDI/1**.
    - When calling through a device (such as a gateway) or service provider to reach another number, the syntax is **technology/devic e/number** such as **SIP/my_provider/5551212** or **DAHDI/4/5551212**.
    - To dial multiple devices at once, simply concatenate the devices together whith the ampersand character (**&**). The first device to answer will get bridged with the caller, and the other endpoints will stop ringing.

        -
          ```
          exten => 6003,1,Dial(SIP/demo-alice&SIP/demo-bob,30)
          ```
2. **Timeout**
    - The number of seconds to allow the device(s) to ring before giving up and moving on to the next priority in the extension.
3. **Options**
    - There are dozens of options that you can set on the outbound call, including call screening, distinctive ringing and more. Type **co re show application dial** at the Asterisk CLI for a complete list of all available options. If you want to specify multiple options, simply concatenate them together. For example, if you want to use both the *m*and **H** options, you would set **mH** as the options parameter.
4. **URL**
    - The fourth parameter is a URL that will be sent to the endpoint. Few endpoints do anything with the URL, but there are a few (softphones mostly) that do act on the URL.

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.

VoiceMail Application

The **VoiceMail()** applications 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.

VoiceMailMain Application

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 Voice mail
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.

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 **voicem ail.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?

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.

Auto-attendant and IVR Menus

In this section, we'll cover the how to build voice menus, often referred to as auto-attedants 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. For more information on DTMF tones, see Section 440.3. DTMF Dialing.

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 problems 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 **Say Digits()** 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 througout 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 independant 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. The most common special extensions include:

i: the 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.

t: the reponse 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.

s: the 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.

h: the 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.

o: the 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.

a: the 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.

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 **R ecord()** 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 **gs m** 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 specifiying 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.

Dialplan Architecture

In this section, we'll begin adding structure to our dialplan. We'll begin by talking about variables and how to use them, as well as how to manipulate them. Then we'll cover more advanced topics, such as pattern matching and using include statements to build classes of functionality.

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 Basics

Channel variables are variables that are set for the current channel (one leg of a bridged phone call). They exist for the lifetime of the channel, and then go away when that channel is hung up. Channel variables on one particular channel are completely independent of channel variables on any other channels; in other words, two channels could each have variables called COUNT with different values.

To assign a value to a channel variable, we use the **Set()** application. Here's an example of setting a variable called **COUNT** to a value of **3**.

```
exten=>6123,1,Set(COUNT=3)
```

To retrieve the value of a variable, we use a special syntax. We put a dollar sign and curly braces around the variable name, like **${COUNT}**

When Asterisk sees the dollar sign and curly braces around a variable name, it substitutes in the value of the variable. Let's look at an example with the **SayNumber()** application.

```
exten=>6123,1,Set(COUNT=3)
exten=>6123,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)**.

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**.

Variable Inheritance Basics

When building your Asterisk dialplan, it may be useful to have one channel inherit variables from another channel. For example, imagine that Alice's call has a channel variable containing an account code, and you'd like to pass that variable on to Bob's channel when Alice's call gets bridged to Bob. We call this variable inheritance. There are two levels of variable inheritance in Asterisk: *single inheritance* and *multiple inheritance*.

Multiple Inheritance

Multiple inheritance means that a channel variable will be inherited by created (spawned) channels, and it will continue to be inherited by any other channels created by the spawned channels. To set multiple inheritance on a channel, preface the variable name with two underscores when giving it a value with the **Set()** application, as shown below.

```
exten=>6123,1,Set(__ACCOUNT=5551212)
```

Single Inheritance

Single inheritance means that a channel variable will be inherited by created (spawned) channels, but not propogate from there to any other swawned channels. To follow our example above, if Alice sets a channel variable with single inheritance and calls Bob, Bob's channel will inherit that channel variable, but the channel variable won't get inherited by any channels that might get spawned by Bob's channel (if the call gets transferred, for example). To set single inheritance on a channel, preface the variable name with an underscore when giving it a value with the **Set()** application, as shown below.

```
exten=>6123,1,Set(_ACCOUNT=5551212)
```

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.

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.

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.

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.

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.

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".

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!

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.

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 first. 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 problem, unless you purposely want to fall through to a less specific match.

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 **[docs:features]**. We'll leave our extensions **6001** and **6002** for Alice and Bob in the **[docs:users]** context, and place extensions such as **6500** in the new **[docs:features]** c ontext. 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 **[docs:fea tures]** 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 **[docs: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 assuming 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. For more information on configuring a SIP provider, see Section 420. The SIP Protocol. For analog connectivity information, see Section 441. Analog Telephony with DAHDI. For more information on connectivity via digital circuits, see Section 450. Basics of Digital Telephony

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 => _NXXNXXXXXX,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 **longdist ance** context.

```
[longdistance]
; 1+ ten digit long-distance numbers
exten => _1NXXNXXXXXX,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 **[docs: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 lease 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 **[docs:local]** context, it could only make local and internal calls. On the other hand, if we were to point it at the **[docs: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.

Many people find it instructive to look at a visual diagram at this point, so let's draw ourselves a map of the contexts we've created so far.

Insert graphic showing chain of includes from international through long-distance to local and to users and features

In this graphic, we've illustrated the various contexts and how they work together. We've also shown that Alice's phone is pointed at the **[docs:international]** context, while Bob's phone is only pointed at the **[docs:local]** context.

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.

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?

On a typical system, you'll want to download three components:

- Asterisk
- DAHDI
- libpri

The **libpri** library allows Asterisk to communicate with ISDN connections. (We'll cover more about ISDN connections in Section 450.8, "Intro to ISDN PRI and BRI Connections".) While not always necessary, we recommend you install it on new systems.

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. It should also be installed on new systems, even if you don't immediately plan on using analog or digital connections to your Asterisk system.

DAHDI

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.

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.

The current version of libpri, DAHDI, and Asterisk can be downloaded from http://downloads.digium.com/pub/telephony/.

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. While any C compiler should be able to compile the Asterisk code, we strongly recommend that you use the **GCC** compiler. Not only is it the most popular free C compiler on Linux and Unix systems, but it's also the compiler that the Asterisk developers are using.

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 the C++ portion of GCC as well, as certain Asterisk modules will use it.

System Libraries

In addition to the C compiler, you'll also need a set of system libraries. These libraries are used by Asterisk and must be installed before you can compile Asterisk. On most operating systems, you'll need to install both the library and it's 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.

A list of libraries you'll need to install include:

- OpenSSL
- ncurses
- newt
- libxml2
- Kernel headers (for building DAHDI drivers)

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

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-1.8.X.Y.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

Let's install DAHDI! On Linux, we will use the DAHDI-linux-complete tarball, which contains both the DAHDI Linux drivers as well as the DAHDI tools. 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.

LibPRI 1.4.13 and later source code depends on DAHDI include files. So, as a change from older versions, one must install DAHDI before installing libPRI.

With Internet Access

```
[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
```

Without Internet Access

When installing on a system without internet access, there are a few additional steps that are required to build DAHDI.
The firmware files for the various VPM modules will need to be downloaded and extracted in the source directory. The file specific links provided below are the current versions as of this writing. Please check the link below for the full list of versions.

http://downloads.digium.com/pub/telephony/firmware/releases/

On a system with internet access, download the following files:

```
wget http://downloads.digium.com/pub/telephony/firmware/releases/dahdi-fw-hx8-2.06.tar.gz

wget http://downloads.digium.com/pub/telephony/firmware/releases/dahdi-fw-oct6114-064-1.05.01.tar.gz

wget http://downloads.digium.com/pub/telephony/firmware/releases/dahdi-fw-oct6114-128-1.05.01.tar.gz

wget http://downloads.digium.com/pub/telephony/firmware/releases/dahdi-fw-vpmoct032-1.8.0.tar.gz

wget http://downloads.digium.com/pub/telephony/firmware/releases/dahdi-fw-tc400m-MR6.12.tar.gz

wget http://downloads.digium.com/pub/telephony/firmware/releases/dahdi-fwload-vpmadt032-1.25.0.tar.gz
```

Now send these to the Asterisk system and store them in

```
/usr/local/src/dahdi-linux-complete-2.X.Y+2.X.Y/linux/drivers/dahdi/firmware/
```

Now we can continue the installation on the Asterisk system using the steps below.

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

[root@server dahdi-linux-complete-2.X.Y+2.X.Y]# cd linux/drivers/dahdi/firmware

[root@server firmware]# for tarball in $(ls dahdi-fw-*.tar.gz); do tar -zxf $tarball; done;

[root@server firmware]# cd -

[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

Before you can build libpri, you'll need to Building and Installing DAHDI

Having finished that, let's compile and install libpri. Again, 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

Checking Asterisk Requirements

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

```
[root@server dahdi-linux-complete-2.X.Y+2.X.Y]# cd /usr/local/src/asterisk-1.8.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-1.8.X.Y]# ./configure
```

This will run for a couple of minutes, and warn you of any missing system libraries or other dependencies.

If you have missing dependencies then you should install them now and then run **configure** again to make sure they are recognized. A helpful way to install most of the dependencies you need is to use the **install_prereq** script included in the **contrib/scripts/** directory of your Asterisk source. It's quite straightforward to use, but may not work on all systems. Run the script with no arguments to see the usage help.

Upon 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$   .?.   $$$$$   .?.       7$$$.
   $.$.   .$$$7. $$$$7 .7$$$.      .$$$.
 .777.   .$$$$$$77$$$77$$$$$7.      $$$,
 $$$~      .7$$$$$$$$$$$$$7.       .$$$.
.$$7          .7$$$$$$$7:          ?$$$.
$$$          ?7$$$$$$$$$$I        .$$$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-1.8.X.Y]# make distclean
```

Using Menuselect to Select Asterisk Options

The next step in the build process is to tell Asterisk which modules[docs:1] 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-1.8.X.Y]# make menuselect
```

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

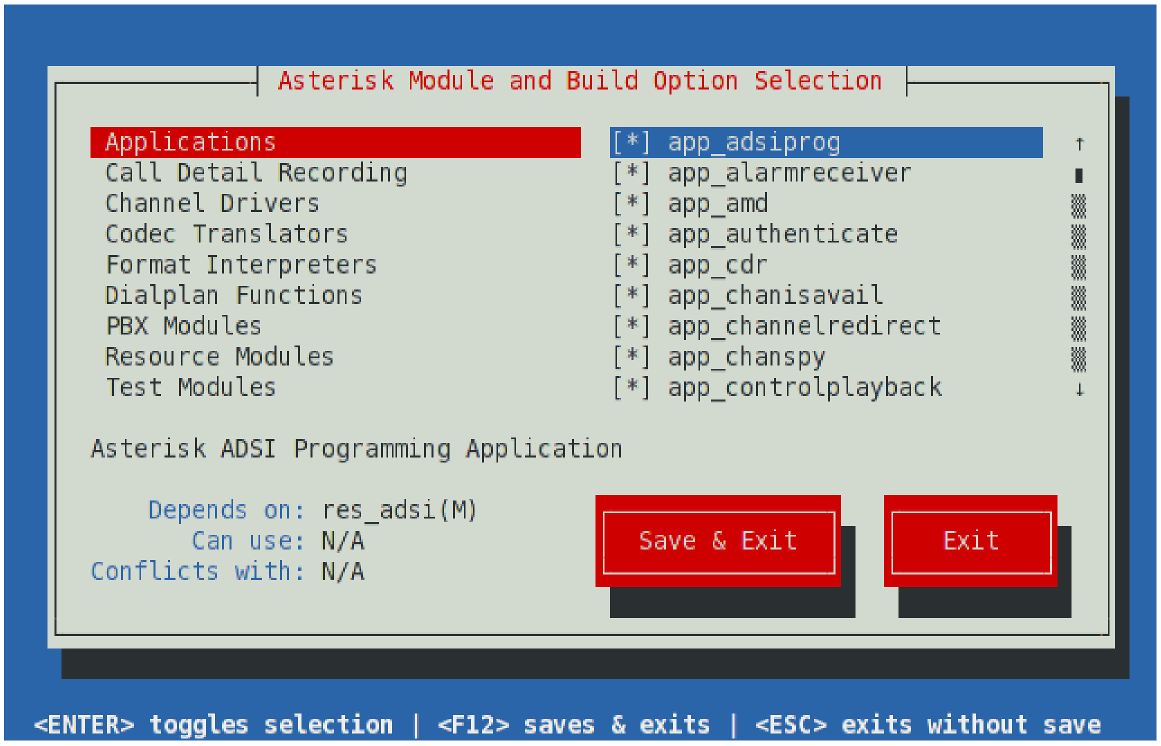
```
Terminal must be at least 80 x 21.
```

Asterisk 1.8+

```
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 [docs:**] 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.

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.



```
            Asterisk Module and Build Option Selection

  Applications                  [*] app_adsiprog
  Call Detail Recording         [*] app_alarmreceiver
  Channel Drivers               [*] app_amd
  Codec Translators             [*] app_authenticate
  Format Interpreters           [*] app_cdr
  Dialplan Functions            [*] app_chanisavail
  PBX Modules                   [*] app_channelredirect
  Resource Modules              [*] app_chanspy
  Test Modules                  [*] app_controlplayback

  Asterisk ADSI Programming Application

      Depends on: res_adsi(M)
         Can use: N/A
  Conflicts with: N/A                 Save & Exit      Exit

<ENTER> toggles selection | <F12> saves & exits | <ESC> exits without save
```

Easier Debugging of Asterisk Crashes

If you're finding that Asterisk is crashing on you, there's a setting in menuselect that will help provide additional information to the Asterisk developers. Go into menuselect, select the the Compiler Flags section (you'll need to scroll down in the left-hand list), and select the DONT_OPTIMIZE setting. Then rebuild Asterisk as shown below. While the Asterisk application will be slightly larger, it will provide additional debugging symbols in the event of a crash.

We should also inform people that the sound prompts are selected in menuselect as well

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.

Building and Installing Asterisk

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

```
[root@server asterisk-1.8.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 ---------+
```

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-1.8.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.

If you installed Asterisk from a tarball (as shown above), the security information is located in a PDF file named asterisk.pdfin the tex/ sub-directory of the source code. If that file doesn't exist, please install the rubber application on your system, and then type:

```
[root@server asterisk-1.8.X.Y]# make pdf
```

Installing Sample Files

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

```
[root@server asterisk-1.8.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, and can be used to stop or restart Asterisk as well. To install the initscript, use the **make config** command.

```
[root@server asterisk-1.8.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-1.8.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 **dah di** in their name.

```
[root@server asterisk-1.8.X.Y]# lsmod | grep dahdi
```

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

```
[root@server asterisk-1.8.X.Y]# /etc/init.d/dadhi 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-1.8.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-1.8.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-1.8.X.Y]# lsmod | grep dahdi
dahdi_dummy      4288 0
dahdi_transcode  7928 1 wctc4xxp
dahdi_voicebus  40464 2 wctdm24xxp,wcte12xp
dahdi          196544 12 dahdi_dummy,wctdm24xxp,wcte11xp,wct1xxp,wcte12xp,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-1.8.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-1.8.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-1.8.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.

Getting Started with Asterisk

In this section, we'll show you how to get started with Asterisk, and how to get around on the Asterisk command-line interface (commonly abbreviated as CLI). We'll also show you how to troubleshoot common problems that you might encounter when first learning Asterisk

Connecting to the CLI

First, let's show you how to connect to the Asterisk command-line interface. As you should recall from the installation, Asterisk typically runs in the background as a service or daemon. If the Asterisk service is already running, type the command below to connect to its command-line interface.

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

The -r parameter tells the system that you want to re-connect to the Asterisk service. If the reconnection is successful, you'll see something like this:

```
[root@server ~]# asterisk -r
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.
=========================================================================
Connected to Asterisk version currently running on server (pid = 11187)
server*CLI>
```

Notice the ***CLI>** text? That's your Asterisk command-line prompt. All of the Asterisk CLI commands take the form of **module action parameters....** For example, type **core show uptime** to see how long Asterisk has been running.

```
server*CLI> core show uptime
System uptime: 1 hour, 34 minutes, 17 seconds
Last reload: 1 hour, 34 minutes, 17 seconds
```

You can use the built-in help to get more information about the various commands. Simply type **core show help** at the Asterisk prompt for a full list of commands, or **core show help command** for help on a particular command.

If you'd like to exit the Asterisk console and return to your shell, just use the **quit** command from the CLI. Such as:

```
server*CLI> quit
```

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"
```

Stopping and Restarting Asterisk

There are four 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.

Changing the Verbose and Debug Levels

Asterisk has two different classes of messages that appear in the command-line interface. The first class is called **verbose** messages. Verbose messages give information about the calls on the system, as well as notices, warnings, and errors. Verbose messages are intended for Asterisk administrators to be able to better manage their systems.

Asterisk allows you to control the verbosity level of the command-line interface. At a verbosity level of zero, you'll receive minimal information about calls on your system. As you increase the verbosity level, you'll see more and more information about the calls. For example, if you set the verbosity level to three or higher, you'll see each step a call takes as it makes its way through the dialplan. There are very few messages that only appear at verbosity levels higher than three.

To change the verbosity level, use the CLI command **core set verbose**, as shown below:

```
server*CLI> core set verbose 3
Verbosity was 0 and is now 3
```

You can also increase (but not decrease) the verbosity level when you connect to the Asterisk CLI from the Linux prompt, by using one or more **-v** paramet ers to the **asterisk** application. For example, this would connect to the Asterisk CLI and set the verbosity to three (if it wasn't already three or higher), because we added three -v parameters:

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

The second class of system messages is known as **debug** messages. These messages are intended for Asterisk developers, to give information about what's happening in the Asterisk program itself. They're often used by developers when trying to track down problems in the code, or to understand why Asterisk is behaving in a certain manner.

To change the debugging level, use the CLI command **core set debug**, as shown below:

```
server*CLI> core set debug 4
Core debug was 0 and is now 4
```

You can also increase (but not decrease) the debugging level when you connect to the Asterisk CLI from the Linux prompt. Simply add one or more **-d** para meters to the **asterisk** application.

```
[root@server ~]# asterisk \-ddddr
```

Verbose and Debug Levels
Please note that the verbose and debug levels are global settings, and apply to all of Asterisk, not just your command-line interface.

We recommend that you set your verbosity level to three while learning Asterisk, so that you can get a feel for what is happening as calls are processed. On a busy production system, however, you'll want to set the verbosity level lower. We also recommend that you use debug messages sparingly, as they tend to be quite verbose and can affect call volume on busy systems.

Simple CLI Tricks

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
```

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.

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>
```

As you can see, there's a wealth of information available from the Asterisk command-line interface, and we've only scratched the surface. In later sections, we'll go into more details about how to use the command-line interface for other purposes.

Troubleshooting

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** applicatio n. 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.

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. Each module that you loads 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 various types of modules and the capabilities they provide.

Types of Asterisk Modules

There are many different types of modules, many of which are shown in the diagram above.

- **Channel Drivers**

At the top of the diagram, we show 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, and so forth.

- **Dialplan Functions**

Functions 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.

- **Resources**

As the name suggests, resources provide resources to Asterisk. 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.

- **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.

Now let's go into more detail on each of the module types.

Channel Driver Modules

All calls from the outside come 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 SIP channel driver, 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 PBX, it's a multi-protocol PBX.

For more information on the various channel drivers, see Section 400. Channel Drivers and External Connectivity.

All channel drivers have a file name that look like **chan_xxxxx.so**, such as **chan_sip.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 Dialplan Fundamentals.

All application modules have file names that looks like **app_xxxxx.so**, such as **app_voicemail.so**.

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 application modules have file names that looks like **func_xxxxx.so**, such as **func_callerid.so**.

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 of asynchronous events such as playing hold music when a call gets placed on hold, or performing call parking.

Resource modules have file names that looks like **res_xxxxx.so**, such as **res_musiconhold.so**.

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 recieve media (audio and video). Asterisk also uses CODEC modules to convert (or transcode) media streams between different formats.

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

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

- ADPCM, 32kbit/s
- G.711 alaw, 64kbit/s
- G.711 ulaw, 64kbit/s
- G.722, 64kbit/s
- G.726, 32kbit/s
- GSM, 13kbit/s
- LPC-10, 2.4kbit/s

If the Speex (www.speex.org) development libraries are detected on your system when Asterisk is built, a CODEC module for Speex will also be installed.

If the iLBC (www.ilbcfreeware.org) development libraries are detected on your system when Asterisk is built, a CODEC module for 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 both software and hardware 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.

For more detailed information on CODECs, see CODECs.

File Format Drivers

Add a list of the file formats that Asterisk supports, then point them at the module in section 400 that goes into more detail?
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_xxxxx.so**, such as **format_wav.so** and **format_jpeg.so**.

Add a list of the file formats that Asterisk supports, then point them at the module in section 400 that goes into more detail?

Call Detail Record (CDR) Drivers

CDR modules are used to store call detail records in a variety of formats. Popular storage mechanisms include comma-separated value (CSV) files, as well as relational databases such as 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.

For more information on call detail records, see Section 370. Call Detail Records.

Call detail record modules have file names that look like **cdr_xxxxx.so**, such as **cdr_csv.so** and **cdr_pgsql.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 that 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_xxxxx.so**, such as **bridge_simple.so** and **bridge_multiplexed.so**.

Call Flow and Bridging Model

Now that you know about the various modules that Asterisk uses, let's talk about the ways that calls flow through an Asterisk system. To explain this clearly, let's say that Alice wants to talk to Bob, and they both have SIP phones connected to their Asterisk system. Let's see what happens!

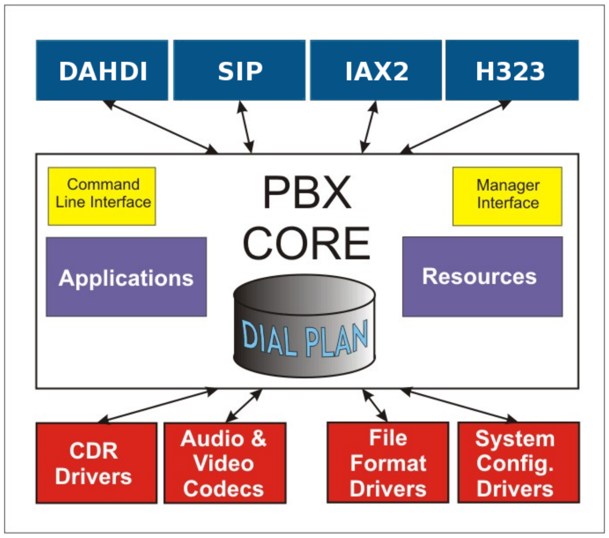
Should we add a graphic to help explain the call flow model?

1. Alice dials extension 6002, which is Bob's extension on the Asterisk system.
2. A SIP message goes from Alice's phone to the SIP channel driver in Asterisk
3. The SIP channel driver authenticates the call. If Alice's phone does not provide the proper credentials, Asterisk rejects the call.
4. At this point, we have Alice's phone communicating with Asterisk.
5. Now the call goes from the SIP channel driver into the core of Asterisk. Asterisk looks for a set of instructions to follow for extension 6002 in the dialplan.
6. Extension 6002 in the dialplan tells Asterisk to call Bob's phone
7. Asterisk makes a call out through the SIP channel driver to Bob's phone.
8. Bob answers his phone.
9. Now we have two independent calls on the Asterisk system: one from Alice, and to Bob. Asterisk now bridges the audio between these two calls (known as **channels** in Asterisk parlance).
10. When one channel hangs up, Asterisk signals the other channel to hang up.

And there we have it! We've shown how calls flow from external devices, through the channel drivers to the core of Asterisk, and back out through the channel drivers to external devices.

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 Architecture

We need to add CEL and Bridge modules to this picture, and take CLI and Manager out for now

The heart of any Asterisk system is the **core**. The PBX core is the essential component that takes care of bridging calls. The core also takes care of other items like reading the configuration files and loading the other modules. We'll talk more about the core below, but for now just remember that all the other modules connect to it.
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.

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 typical Asterisk system has over one hundred fifty different modules!

The core also contains 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.

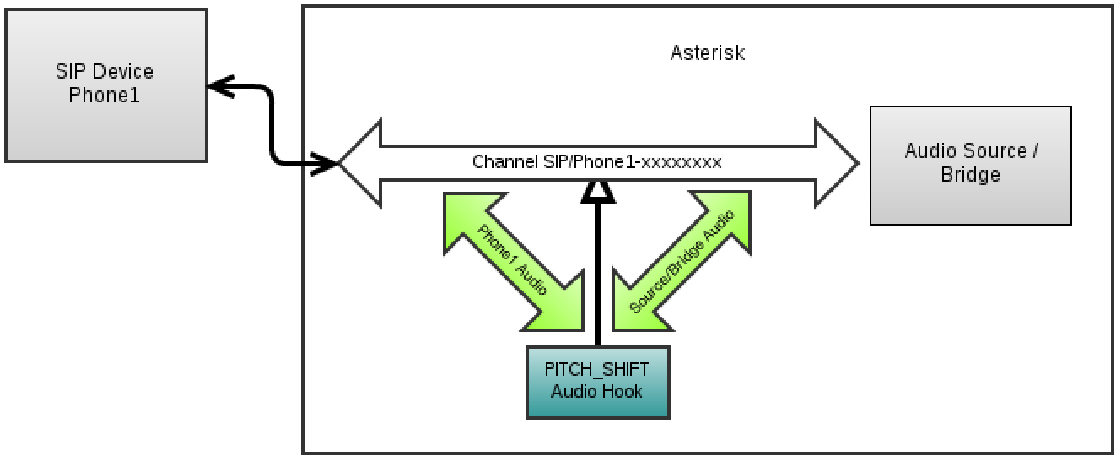
Asterisk modules which are part of the core have a file name that look like **pbx_xxxxx.so**.

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:

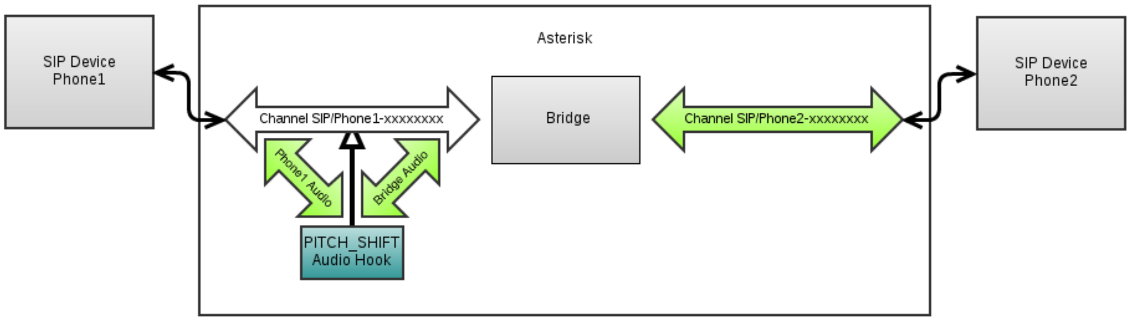
```
exten => 1,1,Answer()
exten => 1,n,Set(PITCH_SHIFT(both)=higher)
exten => 1,n,Voicemail(501)
```

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 transfered, 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 its 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.

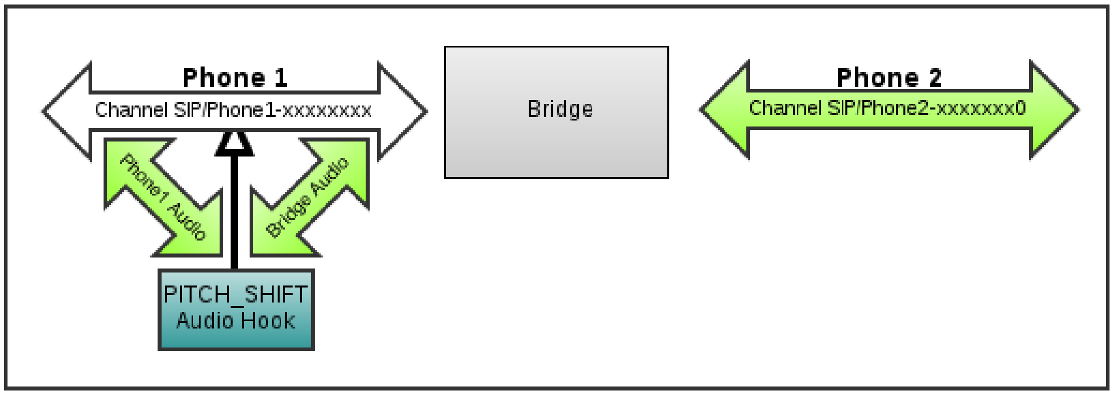
Using AUDIOHOOK_INHERT only requires that AUDIOHOOK_INHERIT(source)=yes is set where source is the name given for the source of the audiohook. For more information on the sources available, see the description of the source argument in the documentation for AUDIOHOOK_INHERIT.

So to fix the above example so that mixmonitor continues to record after the attended transfer, only one extra line is needed.

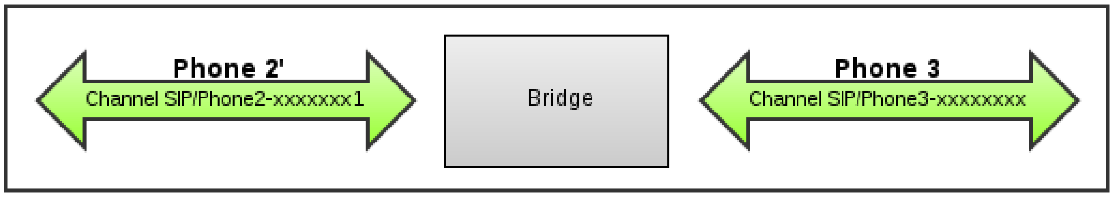
```
exten => 1,1,Answer()
exten => 1,n,MixMonitor(training_recording.wav)
exten => 1,n,Set(AUDIOHOOK_INHERIT(MixMonitor)=yes)
exten => 1,n,Queue(techsupport)
```

Below is an illustrated example of how the masquerade process impacts an audiohook (in the case of the example, PITCH_SHIFT)

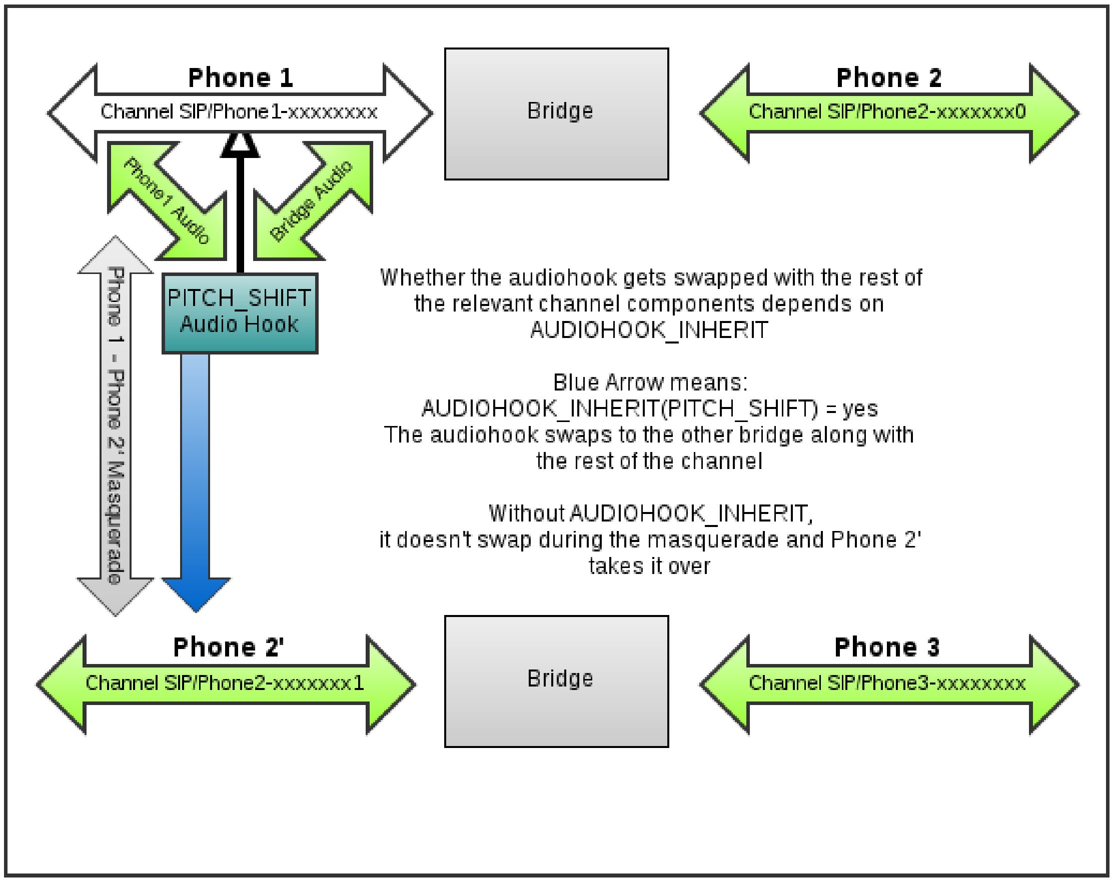
Initial Call Setup



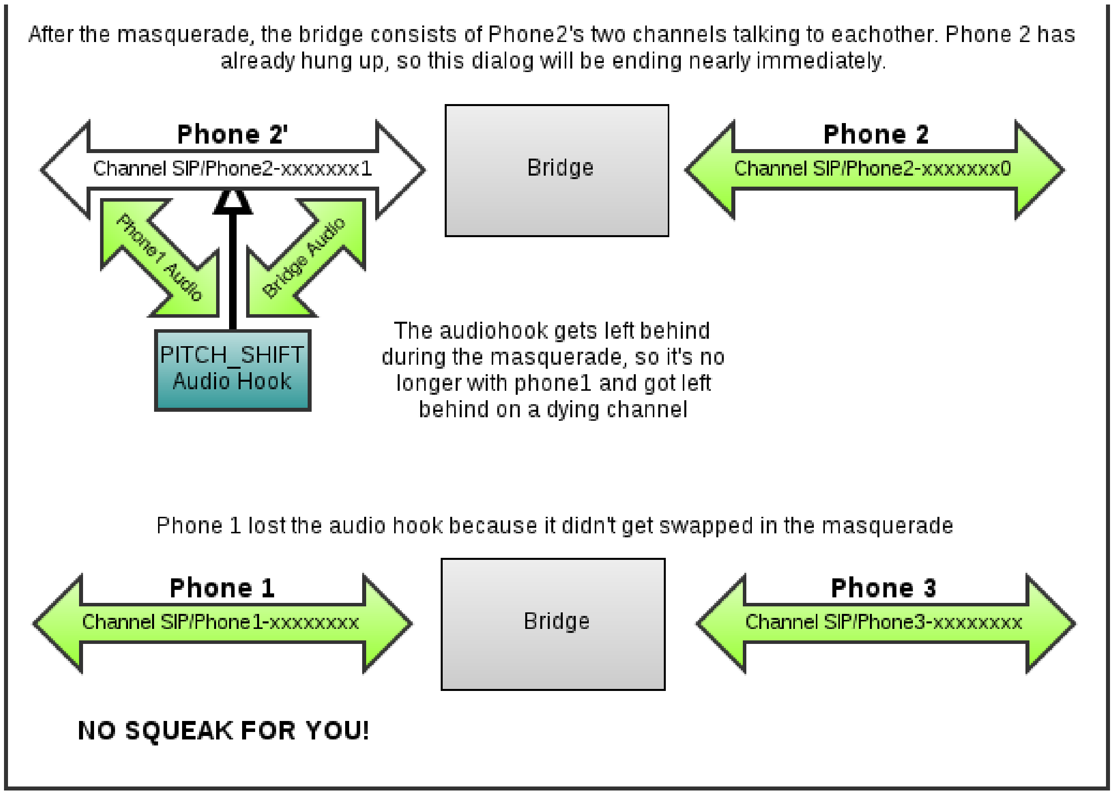
SIP/Phone2 attempts starts to attended transfer SIP/Phone1 to SIP/Phone3



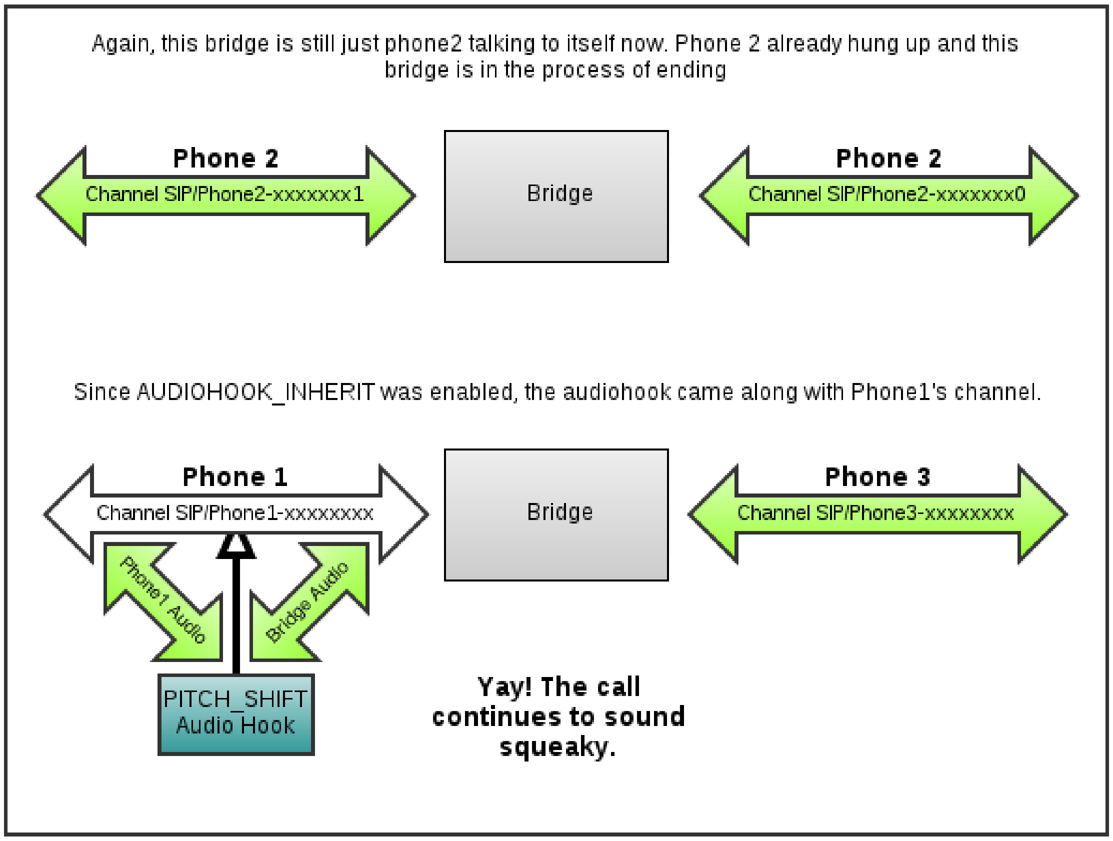
Phone 2 hangs up on Phone 3, initiating the transfer. This requires a masquerade.


Phone 1
Channel SIP/Phone1-xxxxxxxx
Phone1 Audio
Bridge Audio
PITCH_SHIFT Audio Hook
Phone 1 - Phone 2' Masquerade
Bridge
Phone 2
Channel SIP/Phone2-xxxxxxx0
Whether the audiohook gets swapped with the rest of the relevant channel components depends on AUDIOHOOK_INHERIT
Blue Arrow means: AUDIOHOOK_INHERIT(PITCH_SHIFT) = yes The audiohook swaps to the other bridge along with the rest of the channel
Without AUDIOHOOK_INHERIT, it doesn't swap during the masquerade and Phone 2' takes it over
Phone 2'
Channel SIP/Phone2-xxxxxxx1
Bridge
Phone 3
Channel SIP/Phone3-xxxxxxxx

Bridges after Transfer: Without AUDIOHOOK_INHERIT(PITCH_SHIFT)


After the masquerade, the bridge consists of Phone2's two channels talking to eachother. Phone 2 has already hung up, so this dialog will be ending nearly immediately.
Phone 2'
Channel SIP/Phone2-xxxxxxx1
Bridge
Phone 2
Channel SIP/Phone2-xxxxxxx0
Phone1 Audio
Bridge Audio
PITCH_SHIFT Audio Hook
The audiohook gets left behind during the masquerade, so it's no longer with phone1 and got left behind on a dying channel
Phone 1 lost the audio hook because it didn't get swapped in the masquerade
Phone 1
Channel SIP/Phone1-xxxxxxxx
Bridge
Phone 3
Channel SIP/Phone3-xxxxxxxx
NO SQUEAK FOR YOU!

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
Phone 2
Channel SIP/Phone2-xxxxxxx1
Bridge
Phone 2
Channel SIP/Phone2-xxxxxxx0
Since AUDIOHOOK_INHERIT was enabled, the audiohook came along with Phone1's channel.
Phone 1
Channel SIP/Phone1-xxxxxxxx
Bridge
Phone 3
Channel SIP/Phone3-xxxxxxxx
Phone1 Audio
Bridge Audio
PITCH_SHIFT Audio Hook
Yay! The call continues to sound squeaky.

Inheritance of audiohooks can be turned off in the same way by setting AUDIOHOOK_INHERIT(source)=no.

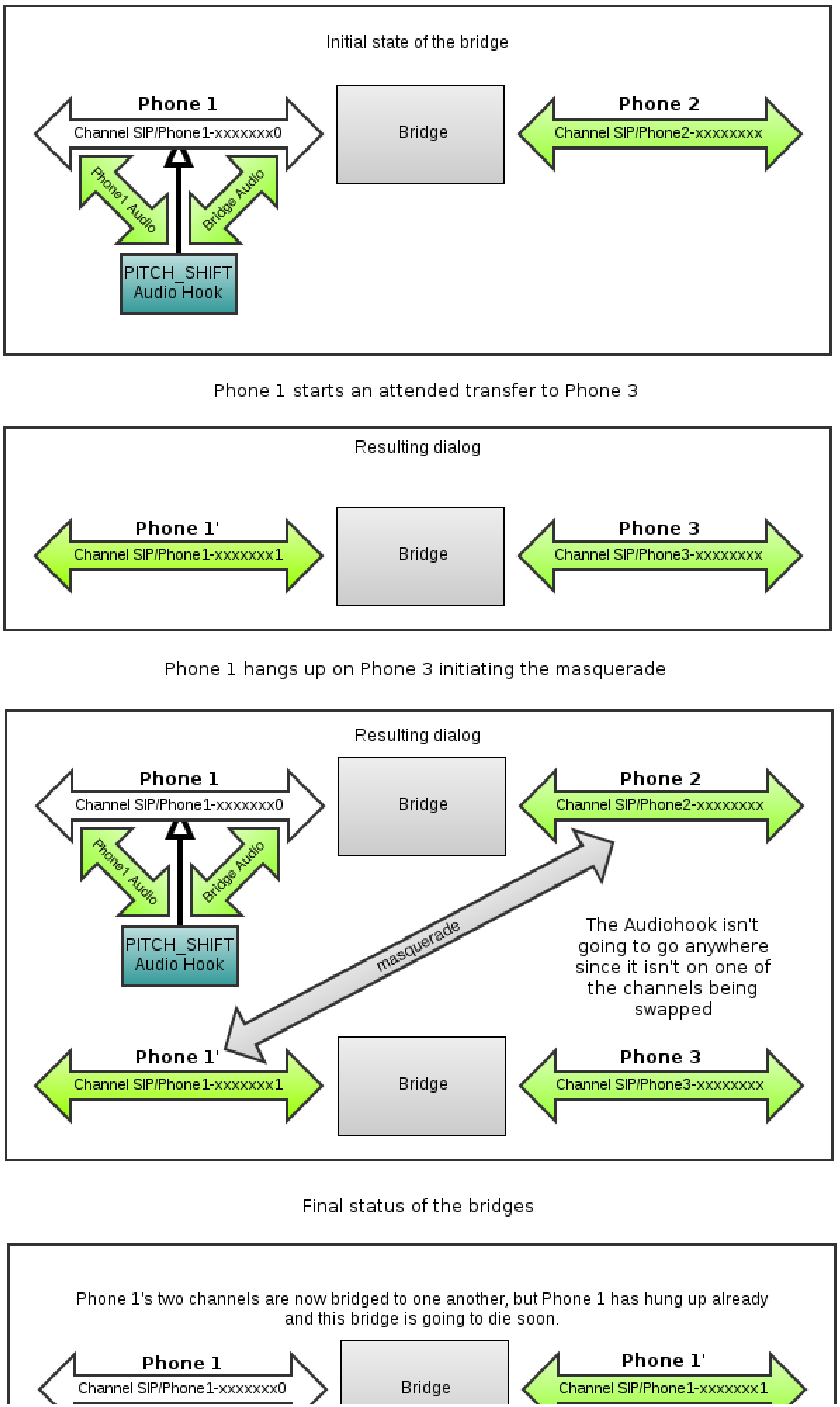
Audiohook Sources

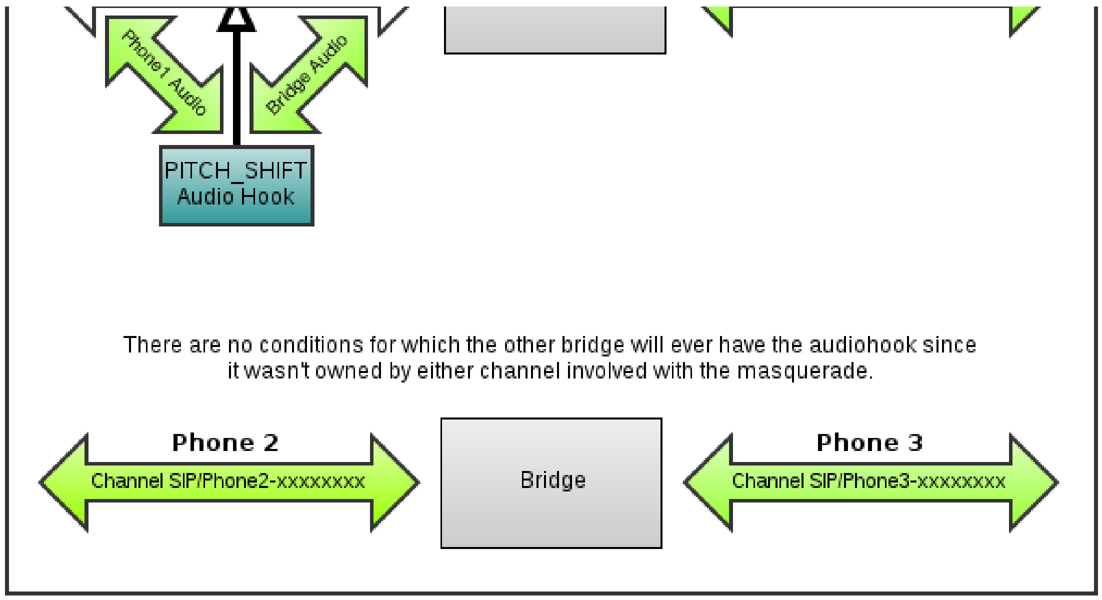
Audiohooks have a source name and can come from a number of sources. An up to date list of possible sources should always be available from the documentation for AUDIOHOOK_INHERIT.

- 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.


Initial state of the bridge
Phone 1
Channel SIP/Phone1-xxxxxxx0
Phone1 Audio
Bridge Audio
PITCH_SHIFT Audio Hook
Bridge
Phone 2
Channel SIP/Phone2-xxxxxxxx
Phone 1 starts an attended transfer to Phone 3
Resulting dialog
Phone 1'
Channel SIP/Phone1-xxxxxxx1
Bridge
Phone 3
Channel SIP/Phone3-xxxxxxxx
Phone 1 hangs up on Phone 3 initiating the masquerade
Resulting dialog
Phone 1
Channel SIP/Phone1-xxxxxxx0
Phone1 Audio
Bridge Audio
PITCH_SHIFT Audio Hook
Bridge
Phone 2
Channel SIP/Phone2-xxxxxxxx
masquerade
The Audiohook isn't going to go anywhere since it isn't on one of the channels being swapped
Phone 1'
Channel SIP/Phone1-xxxxxxx1
Bridge
Phone 3
Channel SIP/Phone3-xxxxxxxx
Final status of the bridges
Phone 1's two channels are now bridged to one another, but Phone 1 has hung up already and this bridge is going to die soon.
Phone 1
Channel SIP/Phone1-xxxxxxx0
Bridge
Phone 1'
Channel SIP/Phone1-xxxxxxx1


Phone1 Audio
Bridge Audio
PITCH_SHIFT Audio Hook
There are no conditions for which the other bridge will ever have the audiohook since it wasn't owned by either channel involved with the masquerade.
Phone 2
Channel SIP/Phone2-xxxxxxxx
Bridge
Phone 3
Channel SIP/Phone3-xxxxxxxx

Asterisk on (Open)Solaris

Asterisk on Solaris 10 and OpenSolaris

On this page

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)

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.

Configuration and Operation

Here is the top-level page for all of the Asterisk Reference Information, formerly found in the doc/ and doc/tex subdirectories of the Asterisk source.

It's been there all along, but now it's here, in an easy to view format (no need to install 800MB of dependancies in Debian just to convert .tex into PDF), that's also searchable. Hoo-ray!

Asterisk Calendaring

The Asterisk Calendaring API 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 three calendaring modules that ship with Asterisk that provide support for iCalendar, CalDAV, and Microsoft Exchange Server calendars. All three modules support event notification. Both CalDAV and Exchange support reading and writing calendars, while iCalendar is a read-only format.

Configuring Asterisk Calendaring

All asterisk calendaring modules are configured through calender.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
autoreminder = 10
channel = SIP/joe
context = calendar_event_notify
extension = s
waittime = 30
```

Module-independent settings

The settings related to calendar event notification are handled by the core calendaring API. These settings are:

- autoreminder - 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.

Calendaring 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)
```

Calendaring Dialplan Examples

Office hours

A common business PBX scenario is 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 adminstrator 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 overrided notification time is set with the autoreminder 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
autoreminder = 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 Channel Drivers

All about Asterisk and its Channel Drivers

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.

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 losspct 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.

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/Installi ng_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 'I', 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.

Local Channel

Introduction to Local Channels

In Asterisk, Local channels are a method used to treat an extension in the dialplan as if it were an external device. In essense, Asterisk will send the call back into the dialplan as the destination of the call, versus sending the call to a device.

Two of the most common areas where Local channels are used include members configured for queues, and in use with callfiles. There are also other uses where you want to ring two destinations, but with different information, such as different callerID for each outgoing request.

Local Channel Examples

Local channels are best demonstrated through the use of an example. Our first example isn't terribly useful, but will demonstrate how Local channels can execute dialplan logic by dialing from the Dial() application.

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.

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 '0004f204001' (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.

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/1&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/1&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.

Understanding when to use (slash)n

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)
```

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.

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 definition, we can keep the Local channel in the call path, along with any channel variables, or other channel specific information.

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, our Local channel will now stay in the call path and not go away.

Why does adding the **/n** option all of a suddon 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 no /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

There are additional modifiers for the Local channel as well. They include:

- 'n' - Adding "/n" at the end of the string will make the Local channel not do a native transfer (the "n" stands for "n"o 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.
- 'j' - Adding "/j" at the end of the string 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' - Using the "/m" option 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' - The "/b" 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.

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/nnnnnnn) 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 adapater 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 somthing 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 ${SMSRC}
${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 rfcomm 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 a 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 Coutry 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
UNIStim 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
UNIStim 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
UNIStim 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
UNIStim 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
UNIStim 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
UNIStim 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 fo
rthe 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 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 * 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*keepalive (ie 60secs), maybe 5*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 (TRUNC((KA+1)/2)*2)*3+20
- before the first keepalive, add 1 if KA<30, add 15 if 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.

Asterisk Configuration

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

General Configuration Information

The top-level page for general (typical) Asterisk configuration information.

Configuration Parser

Introduction

The Asterisk configuration parser in the 1.2 version and beyond series has been improved in a number of ways. In addition to the realtime architecture, we now have the ability to create templates in configuration files, and use these as templates when we configure phones, voicemail accounts and queues. These changes are general to the configuration parser, and works in all configuration files.

General syntax

Asterisk configuration files are defined as follows:

```
[section]
label = value
label2 = value
```

In some files, (e.g. mgcp.conf, dahdi.conf and agents.conf), the syntax is a bit different. In these files the syntax is as follows:

```
[section]
label1 = value1
label2 = value2
object => name
label3 = value3
label2 = value4
object2 => name2
```

In this syntax, we create objects with the settings defined above the object creation. Note that settings are inherited from the top, so in the example above object2 has inherited the setting for "label1" from the first object.
For template configurations, the syntax for defining a section is changed to:

```
[section](options)
label = value
```

The options field is used to define templates, refer to templates and hide templates. Any object can be used as a template.
No whitespace is allowed between the closing "]" and the parenthesis "(".

Comments

All lines that starts with semi-colon ";" is treated as comments and is not parsed.
The ";~~" is a marker for a multi-line comment. Everything after that marker will be treated as a comment until the end-marker "~~;" is found. Parsing begins directly after the end-marker.

```
;This is a comment
label = value
;-- This is
a comment -;
;- Comment --; exten=> 1000,1,dial(SIP/lisa)
```

Including other files

In all of the configuration files, you may include the content of another file with the #include statement. The content of the other file will be included at the row that the #include statement occurred.

```
#include myusers.conf
```

You may also include the output of a program with the #exec directive, if you enable it in asterisk.conf

In asterisk.conf, add the execincludes = yes statement in the options section:

```
[options]
execincludes=yes
```

The exec directive is used like this:

```
#exec /usr/local/bin/myasteriskconfigurator.sh
```

Adding to an existing section

```
[section]
label = value

[section](+)
label2 = value2
```

In this case, the plus sign indicates that the second section (with the same name) is an addition to the first section. The second section can be in another file (by using the #include statement). If the section name referred to before the plus is missing, the configuration will fail to load.

Defining a template-only section

```
[section](!)
label = value
```

The exclamation mark indicates to the config parser that this is a only a template and should not itself be used by the Asterisk module for configuration. The section can be inherited by other sections (see section "Using templates" below) but is not used by itself.

Using templates (or other configuration sections)

```
[section](name[,name])
label = value
```

The name within the parenthesis refers to other sections, either templates or standard sections. The referred sections are included before the configuration engine parses the local settings within the section as though their entire contents (and anything they were previously based upon) were included in the new section. For example consider the following:

```
[foo]
disallow=all
allow=ulaw
allow=alaw

[bar]
allow=gsm
allow=g729
permit=192.168.2.1

[baz](foo,bar)
type=friend
permit=192.168.3.1
context=incoming host=bnm
```

The [baz] section will be processed as though it had been written in the following way:

```
[baz]
disallow=all
allow=ulaw
allow=alaw
allow=gsm
allow=g729
permit=192.168.2.1
type=friend
permit=192.168.3.1
context=incoming host=bnm
```

It should also be noted that there are no guaranteed overriding semantics, meaning that if you define something in one template, you should not expect to be able to override it by defining it again in another template.

Additional Examples
(in top-level sip.conf)

```
[defaults]
type=friend
nat=yes
qualify=on
dtmfmode=rfc2833
disallow=all
allow=alaw
#include accounts/*/sip.conf
```

(in accounts/customer1/sip.conf)

```
[def-customer1](!,defaults)
secret=this_is_not_secret
context=from-customer1
callerid=Customer 1 <300>
accountcode=0001

[phone1](def-customer1)
mailbox=phone1@customer1

[phone2](def-customer1)
mailbox=phone2@customer1
```

This example defines two phones - phone1 and phone2 with settings inherited from "def-customer1". The "def-customer1" is a template that inherits from "defaults", which also is a template.

The asterisk.conf file

Asterisk Main Configuration File

Below is a sample of the main Asterisk configuration file, asterisk.conf. Note that this file is not provided in sample form, because the Makefile creates it when needed and does not touch it when it already exists.

```
[directories]
; Make sure these directories have the right permissions if not
; running Asterisk as root

; Where the configuration files (except for this one) are located
astetcdir => /etc/asterisk

; Where the Asterisk loadable modules are located
astmoddir => /usr/lib/asterisk/modules

; Where additional 'library' elements (scripts, etc.) are located
astvarlibdir => /var/lib/asterisk

; Where AGI scripts/programs are located
astagidir => /var/lib/asterisk/agi-bin

; Where spool directories are located
; Voicemail, monitor, dictation and other apps will create files here
; and outgoing call files (used with pbx_spool) must be placed here
astspooldir => /var/spool/asterisk

; Where the Asterisk process ID (pid) file should be created
astrundir => /var/run/asterisk

; Where the Asterisk log files should be created
astlogdir => /var/log/asterisk

[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
```

CLI Prompt

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

The Asterisk Dialplan

The Asterisk dialplan

The Asterisk dialplan is divided into contexts. A context is simply a group of extensions. For each "line" that should be able to be called, an extension must be added to a context. Then, you configure the calling "line" to have access to this context.

If you change the dialplan, you can use the Asterisk CLI command "dialplan reload" to load the new dialplan without disrupting service in your PBX.

Extensions are routed according to priority and may be based on any set of characters (a-z), digits, #, and *. Please note that when matching a pattern, "N", "X", and "Z" are interpreted as classes of digits.

For each extension, several actions may be listed and must be given a unique priority. When each action completes, the call continues at the next priority (except for some modules which use explicitly GOTO's).

Extensions frequently have data they pass to the executing application (most frequently a string). You can see the available dialplan applications by entering the "core show applications" command in the CLI.

In this version of Asterisk, dialplan functions are added. These can be used as arguments to any application. For a list of the installed functions in your Asterisk, use the "core show functions" command.

Example dialplan

The example dial plan, in the configs/extensions.conf.sample file is installed as extensions.conf if you run "make samples" after installation of Asterisk. This file includes many more instructions and examples than this file, so it's worthwhile to read it.

Special extensions

There are some extensions with important meanings:

- s - What to do when an extension context is entered (unless overridden by the low level channel interface) This is used in macros, and some special cases. "s" is not a generic catch-all wildcard extension.
- i - What to do if an invalid extension is entered
- h - The hangup extension, executed at hangup
- t - What to do if nothing is entered in the requisite amount of time.
- T - This is the extension that is executed when the 'absolute' timeout is reached. See "core show function TIMEOUT" for more information on setting timeouts.
- e - 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.

And finally, the extension context "default" 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).

IP Quality of Service

Introduction

Asterisk supports different QoS settings at the application level for various protocols on both signaling and media. The Type of Service (TOS) byte can be set on outgoing IP packets for various protocols. The TOS byte is used by the network to provide some level of Quality of Service (QoS) even if the network is congested with other traffic.

Asterisk running on Linux can also set 802.1p CoS marks in VLAN packets for the VoIP protocols it uses. This is useful when working in a switched environment. In fact Asterisk only set priority for Linux socket. For mapping this priority and VLAN CoS mark you need to use this command:

```
vconfig set_egress_map [vlan-device] [skb-priority] [vlan-qos]
```

The table below shows all VoIP channel drivers and other Asterisk modules that support QoS settings for network traffic. It also shows the type(s) of traffic for which each module can support setting QoS settings.

Table 2.1: Channel Driver QoS Settings

	Signaling	Audio	Video	Text
chan_sip	+	+	+	+
chan_skinny	+	+	+	
chan_mgcp	+	+		
chan_unistm	+	+		
chan_h323		+		
chan_iax2	+			
chan_pjsip	+	+	+	

Table 2.2: Other ToS Settings

	Signaling	Audio	Video	Text
dundi.conf	+ (tos setting)			
iaxprov.conf	+ (tos setting)			

IP TOS values

The allowable values for any of the tos parameters are: CS0, CS1, CS2, CS3, CS4, CS5, CS6, CS7, AF11, AF12, AF13, AF21, AF22, AF23, AF31, AF32, AF33, AF41, AF42, AF43 and ef (expedited forwarding),*

The tos parameters also take numeric values.*

Note that on a Linux system, Asterisk must be compiled with libcap in order to use the ef tos setting if Asterisk is not run as root.

The lowdelay, throughput, reliability, mincost, and none values have been removed in current releases.

802.1p CoS values

Because 802.1p uses 3 bits of the VLAN header, this parameter can take integer values from 0 to 7.

Recommended values

The recommended values shown below are also included in sample configuration files:

Table 2.3: Recommended QoS Settings

	tos	cos
Signaling	cs3	3

Audio	ef	5
Video	af41	4
Text	af41	3
Other	ef	

IAX2

In iax.conf, there is a "tos" parameter that sets the global default TOS for IAX packets generated by chan_iax2. Since IAX connections combine signalling, audio, and video into one UDP stream, it is not possible to set the TOS separately for the different types of traffic.

In iaxprov.conf, there is a "tos" parameter that tells the IAXy what TOS to set on packets it generates. As with the parameter in iax.conf, IAX packets generated by an IAXy cannot have different TOS settings based upon the type of packet. However different IAXy devices can have different TOS settings.

SIP

In sip.conf, there are four parameters that control the TOS settings: "tos_sip", "tos_audio", "tos_video" and "tos_text". tos_sip controls what TOS SIP call signaling packets are set to. tos_audio, tos_video and tos_text control what TOS values are used for RTP audio, video, and text packets, respectively.
There are four parameters to control 802.1p CoS: "cos_sip", "cos_audio", "cos_video" and "cos_text". The behavior of these parameters is the same as for the SIP TOS settings described above.

Other RTP channels

chan_mgcp, chan_h323, chan_skinny and chan_unistim also support TOS and CoS via setting tos and cos parameters in their corresponding configuration files. Naming style and behavior are the same as for chan_sip.

Reference

IEEE 802.1Q Standard: http://standards.ieee.org/getieee802/download/802.1Q-1998.pdfRelated protocols: IEEE 802.3, 802.2, 802.1D, 802.1Q

RFC 2474 - "Definition of the Differentiated Services Field (DS field) in the IPv4 and IPv6 Headers", Nichols, K., et al, December 1998.

IANA Assignments, DSCP registry Differentiated Services Field Codepoints http://www.iana.org/assignments/dscp-registry
To get the most out of setting the TOS on packets generated by Asterisk, you will need to ensure that your network handles packets with a TOS properly. For Cisco devices, see the previously mentioned "Enterprise QoS Solution Reference Network Design Guide". For Linux systems see the "Linux Advanced Routing & Traffic Control HOWTO" at http://www.lartc.org/.
For more information on Quality of Service for VoIP networks see the "Enterprise QoS Solution Reference Network Design Guide" version 3.3 from Cisco at: http://www.cisco.com/application/pdf/en/us/guest/netsol/ns432/c649/ccmigration_09186a008049b062.pdf

MP3 Support

MP3 Music On Hold

Use of the mpg123 for your music on hold is no longer recommended and is now officially deprecated. You should now use one of the native formats for your music on hold selections.

However, if you still need to use mp3 as your music on hold format, a format driver for reading MP3 audio files is available in the asterisk-addons SVN repository on svn.digium.com or in the asterisk-addons release at http://downloads.asterisk.org/pub/telephony/asterisk/.

ICES

The advent of icecast into Asterisk allows you to do neat things like have a caller stream right into an ice-cast stream as well as using chan_local to place things like conferences, music on hold, etc. into the stream.

You'll need to specify a config file for the ices encoder. An example is included in contrib/asterisk-ices.xml.

Database Support Configuration

Top-level page for information about Database support.

Realtime Database Configuration

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 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 is 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.

FreeTDS

The cdr_tds module now works with most modern release versions of FreeTDS (from at least 0.60 through 0.82). Although versions of FreeTDS prior to 0.82 will work, we recommend using the latest available version for performance and stability reasons.

The latest release of FreeTDS is available from http://www.freetds.org/

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,
      `trustrpid` 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,
      `defaultip` varchar(40) DEFAULT NULL,
      `rtptimeout` int(11) DEFAULT NULL,
      `rtpholdtimeout` int(11) DEFAULT NULL,
      `sendrpid` 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,
      `usereqphone` 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,
      `ignoresdpversion` 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;
```

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,tmP(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.

Asterisk Extension Language (AEL)

Top-level page for all things AEL

Introduction to AEL

AEL is a specialized language intended purely for describing Asterisk dial plans.

The current version was written by Steve Murphy, and is a rewrite of the original version.

This new version further extends AEL, and provides more flexible syntax, better error messages, and some missing functionality.

AEL is really the merger of 4 different 'languages', or syntaxes:

1. The first and most obvious is the AEL syntax itself. A BNF is provided near the end of this document.
2. The second syntax is the Expression Syntax, which is normally handled by Asterisk extension engine, as expressions enclosed in $[...]. The right hand side of assignments are wrapped in $[...] by AEL, and so are the if and while expressions, among others.
3. The third syntax is the Variable Reference Syntax, the stuff enclosed in ${..} curly braces. It's a bit more involved than just putting a variable name in there. You can include one of dozens of 'functions', and their arguments, and there are even some string manipulation notation in there.
4. The last syntax that underlies AEL, and is not used directly in AEL, is the Extension Language Syntax. The extension language is what you see in extensions.conf, and AEL compiles the higher level AEL language into extensions and priorities, and passes them via function calls into Asterisk.
   Embedded in this language is the Application/AGI commands, of which one application call per step, or priority can be made. You can think of this as a "macro assembler" language, that AEL will compile into.

Any programmer of AEL should be familiar with its syntax, of course, as well as the Expression syntax, and the Variable syntax.

AEL and Asterisk in a Nutshell

Asterisk acts as a server. Devices involved in telephony, like DAHDI cards, or Voip phones, all indicate some context that should be activated in their behalf. See the config file formats for IAX, SIP, dahdi.conf, etc. They all help describe a device, and they all specify a context to activate when somebody picks up a phone, or a call comes in from the phone company, or a voip phone, etc.

AEL about Contexts

Contexts are a grouping of extensions.

Contexts can also include other contexts. Think of it as a sort of merge operation at runtime, whereby the included context's extensions are added to the contexts making the inclusion.

AEL about Extensions and priorities

A Context contains zero or more Extensions. There are several predefined extensions. The "s" extension is the "start" extension, and when a device activates a context the "s" extension is the one that is going to be run. Other extensions are the timeout "t" extension, the invalid response, or "i" extension, and there's a "fax" extension. For instance, a normal call will activate the "s" extension, but an incoming FAX call will come into the "fax" extension, if it exists. (BTW, asterisk can tell it's a fax call by the little "beep" that the calling fax machine emits every so many seconds.).

Extensions contain several priorities, which are individual instructions to perform. Some are as simple as setting a variable to a value. Others are as complex as initiating the Voicemail application, for instance. Priorities are executed in order.

When the 's" extension completes, asterisk waits until the timeout for a response. If the response matches an extension's pattern in the context, then control is transferred to that extension. Usually the responses are tones emitted when a user presses a button on their phone. For instance, a context associated with a desk phone might not have any "s" extension. It just plays a dialtone until someone starts hitting numbers on the keypad, gather the number, find a matching extension, and begin executing it. That extension might Dial out over a connected telephone line for the user, and then connect the two lines together.

The extensions can also contain "goto" or "jump" commands to skip to extensions in other contexts. Conditionals provide the ability to react to different stimuli, and there you have it.

AEL about Macros

Think of a macro as a combination of a context with one nameless extension, and a subroutine. It has arguments like a subroutine might. A macro call can be made within an extension, and the individual statements there are executed until it ends. At this point, execution returns to the next statement after the macro call. Macros can call other macros. And they work just like function calls.

AEL about Applications

Application calls, like "Dial()", or "Hangup()", or "Answer()", are available for users to use to accomplish the work of the dialplan. There are over 145 of them at the moment this was written, and the list grows as new needs and wants are uncovered. Some applications do fairly simple things, some provide amazingly complex services.

Hopefully, the above objects will allow you do anything you need to in the Asterisk environment!

Getting Started with AEL

The AEL parser (res_ael.so) is completely separate from the module that parses extensions.conf (pbx_config.so). To use AEL, the only thing that has to be done is the module res_ael.so must be loaded by Asterisk. This will be done automatically if using 'autoload=yes' in /etc/asterisk/modules.conf. When the module is loaded, it will look for 'extensions.ael' in /etc/asterisk/. extensions.conf and extensions.ael can be used in conjunction with each other if that is what is desired. Some users may want to keep extensions.conf for the features that are configured in the 'general' section of extensions.conf.

To reload extensions.ael, the following command can be issued at the CLI:

```
*CLI ael reload
```

AEL Debugging

Right at this moment, the following commands are available, but do nothing:

- Enable AEL contexts debug

```
*CLI> ael debug contexts
```

- Enable AEL macros debug

```
*CLI> ael debug macros
```

- Enable AEL read debug

```
*CLI> ael debug read
```

- Enable AEL tokens debug

```
*CLI> ael debug tokens
```

- Disable AEL debug messages

```
*CLI> ael no debug
```

If things are going wrong in your dialplan, you can use the following facilities to debug your file:

1. The messages log in /var/log/asterisk. (from the checks done at load time).
2. The "show dialplan" command in asterisk
3. The standalone executable, "aelparse" built in the utils/ dir in the source.

About "aelparse"

You can use the "aelparse" program to check your extensions.ael file before feeding it to asterisk. Wouldn't it be nice to eliminate most errors before giving the file to asterisk?

aelparse is compiled in the utils directory of the asterisk release. It isn't installed anywhere (yet). You can copy it to your favorite spot in your PATH.

aelparse has two optional arguments:

1. -d - Override the normal location of the config file dir, (usually /etc/asterisk), and use the current directory instead as the config file dir. Aelparse will then expect to find the file "./extensions.ael" in the current directory, and any included files in the current directory as well.
2. -n - Don't show all the function calls to set priorities and contexts within asterisk. It will just show the errors and warnings from the parsing and semantic checking phases.

General Notes about AEL Syntax

Note that the syntax and style are now a little more free-form. The opening '' (curly-braces) do not have to be on the same line as the keyword that precedes them. Statements can be split across lines, as long as tokens are not broken by doing so. More than one statement can be included on a single line. Whatever you think is best!

You can just as easily say,

```
if(${x}=1) { NoOp(hello!); goto s,3; } else { NoOp(Goodbye!); goto s,12; }
```

as you can say:

```
if(${x}=1) { NoOp(hello!); goto s,3; } else { NoOp(Goodbye!); goto s,12; }
```

or:

```
if(${x}=1) { NoOp(hello!); goto s,3; } else { NoOp(Goodbye!); goto s,12; }
```

or:

```
if (${x}=1) { NoOp(hello!); goto s,3; } else { NoOp(Goodbye!); goto s,12; }
```

AEL Keywords

The AEL keywords are case-sensitive. If an application name and a keyword overlap, there is probably good reason, and you should consider replacing the application call with an AEL statement. If you do not wish to do so, you can still use the application, by using a capitalized letter somewhere in its name. In the Asterisk extension language, application names are NOT case-sensitive.

The following are keywords in the AEL language:

- abstract
- context
- macro
- globals
- ignorepat
- switch
- if
- ifTime
- else
- random
- goto
- jump
- local
- return
- break
- continue
- regexten
- hint
- for
- while
- case
- pattern
- default NOTE: the "default" keyword can be used as a context name, for those who would like to do so.
- catch
- switches
- eswitches
- includes

AEL Procedural Interface and Internals

AEL first parses the extensions.ael file into a memory structure representing the file. The entire file is represented by a tree of "pval" structures linked together.

This tree is then handed to the semantic check routine.

Then the tree is handed to the compiler.

After that, it is freed from memory.

A program could be written that could build a tree of pval structures, and a pretty printing function is provided, that would dump the data to a file, or the tree could be handed to the compiler to merge the data into the asterisk dialplan. The modularity of the design offers several opportunities for developers to simplify apps to generate dialplan data.

AEL version 2 BNF

(hopefully, something close to bnf).

First, some basic objects

```
------------------------
<word> a lexical token consisting of characters matching this pattern:
[-a-zA-Z0-9"_/.\<\>\*\+!$#\[\]][-a-zA-Z0-9"_/.!\*\+\<\>\{\}$#\[\]]*
   <word3-list> a concatenation of up to 3 <word>s.
   <collected-word> all characters encountered until the character that follows the <collected-word> in the grammar.
   -------------------------
   <file> :== <objects>

   <objects> :== <object>
              | <objects> <object>
   <object> :== <context>
            | <macro>
            | <globals>
            | ';'

   <context> :== 'context' <word> '{' <elements> '}'
               | 'context' <word> '{' '}'
               | 'context' 'default' '{' <elements> '}'
               | 'context' 'default' '{' '}'
               | 'abstract' 'context' <word> '{' <elements> '}'
               | 'abstract' 'context' <word> '{' '}'
               | 'abstract' 'context' 'default' '{' <elements> '}'
               | 'abstract' 'context' 'default' '{' '}'

<macro> :== 'macro' <word> '(' <arglist> ')' '{' <macro_statements> '}'
       | 'macro' <word> '(' <arglist> ')' '{' '}'
       | 'macro' <word> '(' ')' '{' <macro_statements> '}'
       | 'macro' <word> '(' ')' '{' '}'

<globals> :== 'globals' '{' <global_statements> '}'
         | 'globals' '{' '}'

<global_statements> :== <global_statement>
                   | <global_statements> <global_statement>


<global_statement> :== <word> '=' <collected-word> ';'

<arglist> :== <word>
         | <arglist> ',' <word>

<elements> :== <element>
             | <elements> <element>

<element> :== <extension>
         | <includes>
         | <switches>
         | <eswitches>
         | <ignorepat>
         | <word> '=' <collected-word> ';'
         | 'local' <word> '=' <collected-word> ';'
         | ';'


<ignorepat> :== 'ignorepat' '=>' <word> ';'

<extension> :== <word> '=>' <statement>
           | 'regexten' <word> '=>' <statement>
           | 'hint' '(' <word3-list> ')' <word> '=>' <statement>
           | 'regexten' 'hint' '(' <word3-list> ')' <word> '=>' <statement>

<statements> :== <statement>
            | <statements> <statement>

<if_head> :== 'if' '(' <collected-word> ')'
<random_head> :== 'random' '(' <collected-word> ')'

<ifTime_head> :== 'ifTime' '(' <word3-list> ':' <word3-list> ':' <word3-list> '|' <word3-list> '|' <word3-list> '|' <word3-list>
')'
                       | 'ifTime' '(' <word> '|' <word3-list> '|' <word3-list> '|' <word3-list> ')'

<word3-list> :== <word>
       | <word> <word>
       | <word> <word> <word>
<switch_head> :== 'switch' '(' <collected-word> ')' '{'

<statement> :== '{' <statements> '}'
       | <word> '=' <collected-word> ';'
       | 'local' <word> '=' <collected-word> ';'
       | 'goto' <target> ';'
       | 'jump' <jumptarget> ';'
       | <word> ':'
       | 'for' '(' <collected-word> ';' <collected-word> ';' <collected-word> ')' <statement>
       | 'while' '(' <collected-word> ')' <statement>
       | <switch_head> '}'
       | <switch_head> <case_statements> '}'
       | '&' macro_call ';'
       | <application_call> ';'
       | <application_call> '=' <collected-word> ';'
       | 'break' ';'
       | 'return' ';'
       | 'continue' ';'
       | <random_head> <statement>
       | <random_head> <statement> 'else' <statement>
       | <if_head> <statement>
       | <if_head> <statement> 'else' <statement>
       | <ifTime_head> <statement>
       | <ifTime_head> <statement> 'else' <statement>
       | ';'
<target> :== <word>
       | <word> '|' <word>
       | <word> '|' <word> '|' <word>
       | 'default' '|' <word> '|' <word>
       | <word> ',' <word>
       | <word> ',' <word> ',' <word>
       | 'default' ',' <word> ',' <word>

<jumptarget> :== <word>
               | <word> ',' <word>
               | <word> ',' <word> '@' <word>
               | <word> '@' <word>
               | <word> ',' <word> '@' 'default'
               | <word> '@' 'default'
<macro_call> :== <word> '(' <eval_arglist> ')'
       | <word> '(' ')'

<application_call_head> :== <word> '('
<application_call> :== <application_call_head> <eval_arglist> ')'
       | <application_call_head> ')'

<eval_arglist> :== <collected-word>
       | <eval_arglist> ',' <collected-word>
       | /* nothing */
       | <eval_arglist> ',' /* nothing */

  <case_statements> :== <case_statement>
         | <case_statements> <case_statement>

  <case_statement> :== 'case' <word> ':' <statements>
         | 'default' ':' <statements>
         | 'pattern' <word> ':' <statements>
         | 'case' <word> ':'
         | 'default' ':'
         | 'pattern' <word> ':'
  <macro_statements> :== <macro_statement>
         | <macro_statements> <macro_statement>

  <macro_statement> :== <statement>
         | 'catch' <word> '{' <statements> '}'
  <switches> :== 'switches' '{' <switchlist> '}'
         | 'switches' '{' '}'
  <eswitches> :== 'eswitches' '{' <switchlist> '}'
         | 'eswitches' '{' '}'

  <switchlist> :== <word> ';'
         | <switchlist> <word> ';'
  <includeslist> :== <includedname> ';'
         | <includedname> '|' <word3-list> ':' <word3-list> ':' <word3-list> '|' <word3-list> '|' <word3-list> '|' <word3-list>
';'
         | <includedname> '|' <word> '|' <word3-list> '|' <word3-list> '|' <word3-list> ';'
         | <includeslist> <includedname> ';'
         | <includeslist> <includedname> '|' <word3-list> ':' <word3-list> ':' <word3-list> '|' <word3-list> '|' <word3-list> '|'
<word3-list> ';'
         | <includeslist> <includedname> '|' <word> '|' <word3-list> '|' <word3-list> '|' <word3-list> ';'
  <includedname> :== <word>
          | 'default'
```

```
<includes> :== 'includes' '{' <includeslist> '}'
       | 'includes' '{' '}'
```

AEL Example Usages

Example usages of AEL

AEL Comments

Comments begin with // and end with the end of the line.

Comments are removed by the lexical scanner, and will not be recognized in places where it is busy gathering expressions to wrap in $[] , or inside application call argument lists. The safest place to put comments is after terminating semicolons, or on otherwise empty lines.

AEL Context

Contexts in AEL represent a set of extensions in the same way that they do in extensions.conf.

```
context default {
}
```

A context can be declared to be "abstract", in which case, this declaration expresses the intent of the writer, that this context will only be included by another context, and not "stand on its own". The current effect of this keyword is to prevent "goto " statements from being checked.

```
abstract context longdist {
    _1NXXNXXXXXX => NoOp(generic long distance dialing actions in the US);
}
```

AEL Extensions

To specify an extension in a context, the following syntax is used. If more than one application is be called in an extension, they can be listed in order inside of a block.

```
context default {
    1234 => Playback(tt-monkeys);
    8000 => {
         NoOp(one);
         NoOp(two);
         NoOp(three);
    };
     _5XXX => NoOp(it's a pattern!);
}
```

Two optional items have been added to the AEL syntax, that allow the specification of hints, and a keyword, regexten, that will force the numbering of priorities to start at 2.

The ability to make extensions match by CID is preserved in AEL; just use '/' and the CID number in the specification. See below.

```
context default {
    regexten _5XXX => NoOp(it's a pattern!);
}

context default {
    hint(Sip/1) _5XXX => NoOp(it's a pattern!);
}

context default {
    regexten hint(Sip/1) _5XXX => NoOp(it's a pattern!);
}
```

The regexten must come before the hint if they are both present.

CID matching is done as with the extensions.conf file. Follow the extension name/number with a slash and the number to match against the Caller ID:

```
context zoombo {
    819/7079953345 => { NoOp(hello, 3345); }
}
```

In the above, the 819/7079953345 extension will only be matched if the CallerID is 7079953345, and the dialed number is 819. Hopefully you have another 819 extension defined for all those who wish 819, that are not so lucky as to have 7079953345 as their CallerID!

AEL Includes

Contexts can be included in other contexts. All included contexts are listed within a single block.

```
context default {
    includes {
        local;
        longdistance;
        international;
    }
}
```

Time-limited inclusions can be specified, as in extensions.conf format, with the fields described in the wiki page Asterisk cmd GotoIfTime.

```
context default {
    includes {
        local;
        longdistance|16:00-23:59|mon-fri||;
        international;
    }
}
```

AEL including other files

You can include other files with the #include "filepath" construct.

```
#include "/etc/asterisk/testfor.ael"
```

An interesting property of the #include, is that you can use it almost anywhere in the .ael file. It is possible to include the contents of a file in a macro, context, or even extension. The #include does not have to occur at the beginning of a line. Included files can include other files, up to 50 levels deep. If the path provided in quotes is a relative path, the parser looks in the config file directory for the file (usually /etc/asterisk).

AEL Dialplan Switches

Switches are listed in their own block within a context. For clues as to what these are used for, see Asterisk - dual servers, and Asterisk config extensions.conf.

```
context default {
    switches {
        DUNDi/e164;
        IAX2/box5;
        };
    eswitches {
        IAX2/context@${CURSERVER};
    }
}
```

AEL Ignorepat

ignorepat can be used to instruct channel drivers to not cancel dialtone upon receipt of a particular pattern. The most commonly used example is '9'.

```
context outgoing {
    ignorepat => 9;
}
```

AEL Variables

Variables in Asterisk do not have a type, so to define a variable, it just has to be specified with a value.

Global variables are set in their own block.

```
globals {
    CONSOLE=Console/dsp;
    TRUNK=DAHDI/g2;
}
```

Variables can be set within extensions as well.

```
context foo {
    555 => {
        x=5;
        y=blah;
        divexample=10/2
        NoOp(x is ${x} and y is ${y} !);
    }
}
```

NOTE: AEL wraps the right hand side of an assignment with $[ ] to allow expressions to be used If this is unwanted, you can protect the right hand side from being wrapped by using the Set() application. Read the README.variables about the requirements and behavior of $[ ] expressions.
NOTE: These things are wrapped up in a $[ ] expression: The while() test; the if() test; the middle expression in the for(x; y; z) statement (the y expression); Assignments - the right hand side, so a = b - Set(a=$[b])

Writing to a dialplan function is treated the same as writing to a variable.

```
context blah {
    s => {
        CALLERID(name)=ChickenMan;
        NoOp(My name is ${CALLERID(name)} !);
    }
}
```

You can declare variables in Macros, as so:

```
Macro myroutine(firstarg, secondarg) {
    Myvar=1;
    NoOp(Myvar is set to ${myvar});
}
```

AEL Local Variables

In 1.2, and 1.4, ALL VARIABLES are CHANNEL variables, including the function arguments and associated ARG1, ARG2, etc variables. Sorry.
In trunk (1.6 and higher), we have made all arguments local variables to a macro call. They will not affect channel variables of the same name. This includes the ARG1, ARG2, etc variables.

Users can declare their own local variables by using the keyword 'local' before setting them to a value;

```
Macro myroutine(firstarg, secondarg) {
    local Myvar=1;
    NoOp(Myvar is set to ${Myvar}, and firstarg is ${firstarg}, and secondarg is
${secondarg});
}
```

In the above example, Myvar, firstarg, and secondarg are all local variables, and will not be visible to the calling code, be it an extension, or another Macro.

If you need to make a local variable within the Set() application, you can do it this way:

```
Macro myroutine(firstarg, secondarg) {
    Set(LOCAL(Myvar)=1);
    NoOp(Myvar is set to ${Myvar}, and firstarg is ${firstarg}, and secondarg is
${secondarg});
}
```

AEL Conditionals

AEL supports if and switch statements, like AEL, but adds ifTime, and random. Unlike the original AEL, though, you do NOT need to put curly braces around a single statement in the "true" branch of an if(), the random(), or an ifTime() statement. The if(), ifTime(), and random() statements allow optional else clause.

```
context conditional {
    _8XXX => {
        Dial(SIP/${EXTEN});
        if ("${DIALSTATUS}" = "BUSY")
        {
            NoOp(yessir);
            Voicemail(${EXTEN},b);
        }
        else
            Voicemail(${EXTEN},u);
        ifTime (14:00-25:00,sat-sun,,)
            Voicemail(${EXTEN},b);
        else
        {
            Voicemail(${EXTEN},u);
            NoOp(hi, there!);
        }
        random(51) NoOp(This should appear 51% of the time);
        random( 60 )
        {
                NoOp( This should appear 60% of the time );
        }
        else
        {
                random(75)
                {
                    NoOp( This should appear 30% of the time! );
                }
                else
                {
                    NoOp( This should appear 10% of the time! );
                }
        }
    }
    _777X => {
        switch (${EXTEN}) {
            case 7771:
                NoOp(You called 7771!);
                break;
            case 7772:
                NoOp(You called 7772!);
                break;
            case 7773:
                NoOp(You called 7773!);
                // fall thru-
            pattern 777[4-9]:
                NoOp(You called 777 something!);
            default: NoOp(In the default clause!);
        }
    }
}
```

The conditional expression in if() statements (the "${DIALSTATUS}" = "BUSY" above) is wrapped by the compiler in $[] for evaluation.

Neither the switch nor case values are wrapped in $[ ]; they can be constants, or ${var} type references only.

AEL generates each case as a separate extension. case clauses with no terminating 'break', or 'goto', have a goto inserted, to the next clause, which creates a 'fall thru' effect.

AEL introduces the ifTime keyword/statement, which works just like the if() statement, but the expression is a time value, exactly like that used by the application GotoIfTime(). See Asterisk cmd GotoIfTime

The pattern statement makes sure the new extension that is created has an '_' preceding it to make sure asterisk recognizes the extension name as a pattern.

Every character enclosed by the switch expression's parenthesis are included verbatim in the labels generated. So watch out for spaces!

NEW: Previous to version 0.13, the random statement used the "Random()" application, which has been deprecated. It now uses the RAND() function instead, in the GotoIf application.

AEL goto, jump, and labels

This is an example of how to do a goto in AEL.

```
context gotoexample {
    s => {
        begin:
            NoOp(Infinite Loop! yay!);
            Wait(1);
            goto begin; // go to label in same extension
    }
    3 => {
        goto s,
        begin; // go to label in different extension
    }
    4 => {
        goto gotoexample,s,begin; // overkill go to label in same context
    }
}

context gotoexample2 {
    s => {
        end:
            goto gotoexample,s,begin; // go to label in different context
    }
}
```

You can use the special label of "1" in the goto and jump statements. It means the "first" statement in the extension. I would not advise trying to use numeric labels other than "1" in goto's or jumps, nor would I advise declaring a "1" label anywhere! As a matter of fact, it would be bad form to declare a numeric label, and it might conflict with the priority numbers used internally by asterisk.

The syntax of the jump statement is: jump extension[,priority][@context] If priority is absent, it defaults to "1". If context is not present, it is assumed to be the same as that which contains the "jump".

```
context gotoexample {
    s => {
        begin:
            NoOp(Infinite Loop! yay!);
            Wait(1);
            jump s; // go to first extension in same extension
    }
    3 => {
        jump s,begin; // go to label in different extension
    }
    4 => {
        jump s,begin@gotoexample; // overkill go to label in same context }
    }

context gotoexample2 {
    s => {
        end:
            jump s@gotoexample; // go to label in different context }
}
```

Goto labels follow the same requirements as the Goto() application, except the last value has to be a label. If the label does not exist, you will have run-time errors. If the label exists, but in a different extension, you have to specify both the extension name and label in the goto, as in: goto s,z; if the label is in a different context, you specify context,extension,label. There is a note about using goto's in a switch statement below...

AEL introduces the special label "1", which is the beginning context number for most extensions.

AEL Macros

A macro is defined in its own block like this. The arguments to the macro are specified with the name of the macro. They are then referred to by that same name. A catch block can be specified to catch special extensions.

```
macro std-exten( ext , dev ) {
    Dial(${dev}/${ext},20);
    switch(${DIALSTATUS}) {
        case BUSY:
            Voicemail(${ext},b);
            break;
        default:
            Voicemail(${ext},u);
    }
    catch a {
        VoiceMailMain(${ext});
        return;
    }
}
```

A macro is then called by preceding the macro name with an ampersand. Empty arguments can be passed simply with nothing between commas.

```
context example {
    _5XXX => &std-exten(${EXTEN}, "IAX2");
    _6XXX => &std-exten(, "IAX2");
    _7XXX => &std-exten(${EXTEN},);
    _8XXX => &std-exten(,);
}
```

AEL Loops

AEL has implementations of 'for' and 'while' loops.

```
context loops {
    1 => {
        for (x=0; ${x} < 3; x=${x} + 1) {
            Verbose(x is ${x} !);
        }
    }
    2 => {
        y=10;
        while (${y} >= 0) {
            Verbose(y is ${y} !);
            y=${y}-1;
        }
    }
}
```

NOTE: The conditional expression (the "${y} = 0" above) is wrapped in $[ ] so it can be evaluated. NOTE: The for loop test expression (the "$x 3" above) is wrapped in $[ ] so it can be evaluated.

AEL Break, Continue, and Return

Three keywords:

1. break
2. continue
3. return

are included in the syntax to provide flow of control to loops, and switches.

The break can be used in switches and loops, to jump to the end of the loop or switch.

The continue can be used in loops (while and for) to immediately jump to the end of the loop. In the case of a for loop, the increment and test will then be performed. In the case of the while loop, the continue will jump to the test at the top of the loop.

The return keyword will cause an immediate jump to the end of the context, or macro, and can be used anywhere.

AEL Examples

```
context demo {
    s => {
        Wait(1);
        Answer();
        TIMEOUT(digit)=5;
        TIMEOUT(response)=10;
    restart:
        Background(demo-congrats);
    instructions:
        for (x=0; ${x} < 3; x=${x} + 1) {
            Background(demo-instruct);
            WaitExten();
        }
    }
    2 => {
        Background(demo-moreinfo);
        goto s,instructions;
    }
    3 => {
        LANGUAGE()=fr;
        goto s,restart;
    }
    500 => {
        Playback(demo-abouttotry);
        Dial(IAX2/guest@misery.digium.com);
        Playback(demo-nogo);
        goto s,instructions;
    }
    600 => {
        Playback(demo-echotest);
        Echo();
        Playback(demo-echodone);
        goto s,instructions;
    }
    # => {
        hangup:
            Playback(demo-thanks);
            Hangup();
    }
    t => goto #,hangup;
    i => Playback(invalid);
}
```

AEL Semantic Checks

AEL, after parsing, but before compiling, traverses the dialplan tree, and makes several checks:

- Macro calls to non-existent macros.
- Macro calls to contexts.
- Macro calls with argument count not matching the definition.
- application call to macro. (missing the '&')
- application calls to "GotoIf", "GotoIfTime", "while", "endwhile", "Random", and "execIf", will generate a message to consider converting the call to AEL goto, while, etc. constructs.
- goto a label in an empty extension.
- goto a non-existent label, either a within-extension, within-context, or in a different context, or in any included contexts. Will even check "sister" context references.
- All the checks done on the time values in the dial plan, are done on the time values in the ifTime() and includes times: o the time range has to have two times separated by a dash; o the times have to be in range of 0 to 24 hours. o The weekdays have to match the internal list, if they are provided; o the day of the month, if provided, must be in range of 1 to 31; o the month name or names have to match those in the internal list.
- (0.5) If an expression is wrapped in $[...], and the compiler will wrap it again, a warning is issued.
- (0.5) If an expression had operators (you know, +,-,,/,issued. Maybe someone forgot to wrap a variable name?*
- (0.12) check for duplicate context names.
- (0.12) check for abstract contexts that are not included by any context.
- (0.13) Issue a warning if a label is a numeric value.

There are a subset of checks that have been removed until the proposed AAL (Asterisk Argument Language) is developed and incorporated into Asterisk. These checks will be:

- (if the application argument analyzer is working: the presence of the 'j' option is reported as error.
- if options are specified, that are not available in an application.
- if you specify too many arguments to an application.
- a required argument is not present in an application call.
- Switch-case using "known" variables that applications set, that does not cover all the possible values. (a "default" case will solve this problem. Each "unhandled" value is listed.
- a Switch construct is used, which is uses a known variable, and the application that would set that variable is not called in the same extension. This is a warning only...
- Calls to applications not in the "applist" database (installed in /var/lib/asterisk/applist" on most systems).
- In an assignment statement, if the assignment is to a function, the function name used is checked to see if it one of the currently known functions. A warning is issued if it is not.

Differences with the original version of AEL

1. The $[...] expressions have been enhanced to include the ==, , and && operators. These operators are exactly equivalent to the =, , and & operators, respectively. Why? So the C, Java, C++ hackers feel at home here.
2. It is more free-form. The newline character means very little, and is pulled out of the white-space only for line numbers in error messages.
3. It generates more error messages - by this I mean that any difference between the input and the grammar are reported, by file, line number, and column.
4. It checks the contents of $[ ] expressions (or what will end up being $[ ] expressions!) for syntax errors. It also does matching paren/bracket counts.
5. It runs several semantic checks after the parsing is over, but before the compiling begins, see the list above.
6. It handles #include "filepath" directives. - ALMOST anywhere, in fact. You could easily include a file in a context, in an extension, or at the root level. Files can be included in files that are included in files, down to 50 levels of hierarchy...
7. Local Goto's inside Switch statements automatically have the extension of the location of the switch statement appended to them.
8. A pretty printer function is available within pbx_ael.so.
9. In the utils directory, two standalone programs are supplied for debugging AEL files. One is called "aelparse", and it reads in the /etc/asterisk/extensions.ael file, and shows the results of syntax and semantic checking on stdout, and also shows the results of compilation to stdout. The other is "aelparse1", which uses the original ael compiler to do the same work, reading in "/etc/asterisk/extensions.ael", using the original 'pbx_ael.so' instead.
10. AEL supports the "jump" statement, and the "pattern" statement in switch constructs. Hopefully these will be documented in the AEL README.
11. Added the "return" keyword, which will jump to the end of an extension/Macro.
12. Added the ifTime (time rangedays of weekdays of monthmonths) else construct, which executes much like an if () statement, but the decision is based on the current time, and the time spec provided in the ifTime. See the example above. (Note: all the other time-dependent Applications can be used via ifTime)
13. Added the optional time spec to the contexts in the includes construct. See examples above.
14. You don't have to wrap a single "true" statement in curly braces, as in the original AEL. An "else" is attached to the closest if. As usual, be careful about nested if statements! When in doubt, use curlies!
15. Added the syntax regexten hint(channel) to precede an extension declaration. See examples above, under "Extension". The regexten keyword will cause the priorities in the extension to begin with 2 instead of 1. The hint keyword will cause its arguments to be inserted in the extension under the hint priority. They are both optional, of course, but the order is fixed at the moment- the regexten must come before the hint, if they are both present.
16. Empty case/default/pattern statements will "fall thru" as expected. (0.6)
17. A trailing label in an extension, will automatically have a NoOp() added, to make sure the label exists in the extension on Asterisk. (0.6)
18. (0.9) the semicolon is no longer required after a closing brace! (i.e. "];" === "}". You can have them there if you like, but they are not necessary. Someday they may be rejected as a syntax error, maybe.
19. (0.9) the // comments are not recognized and removed in the spots where expressions are gathered, nor in application call arguments. You may have to move a comment if you get errors in existing files.
20. (0.10) the random statement has been added. Syntax: random (expr) lucky-statement [else unlucky-statement]. The probability of the lucky-statement getting executed is expr, which should evaluate to an integer between 0 and 100. If the lucky-statement isn't so lucky this time around, then the unlucky-statement gets executed, if it is present.

AEL Hints and Bugs

The safest way to check for a null strings is to say $["${x}" = ""] The old way would do as shell scripts often do, and append something on both sides, like this: $[${x}foo = foo]. The trouble with the old way, is that, if x contains any spaces, then problems occur, usually syntax errors. It is better practice and safer wrap all such tests with double quotes! Also, there are now some functions that can be used in a variable reference, ISNULL(), and LEN(), that can be used to test for an empty string: ${ISNULL(${x})} or $[${LEN(${x})} = 0].

Assignment vs. Set(). Keep in mind that setting a variable to value can be done two different ways. If you choose say 'x=y;', keep in mind that AEL will wrap the right-hand-side with $[]. So, when compiled into extension language format, the end result will be 'Set(x=$[y])'. If you don't want this effect, then say "Set(x=y);" instead.

The Full Power of AEL

A newcomer to Asterisk will look at the above constructs and descriptions, and ask, "Where's the string manipulation functions?", "Where's all the cool operators that other languages have to offer?", etc.

The answer is that the rich capabilities of Asterisk are made available through AEL, via:

- Applications: See Asterisk - documentation of application commands
- Functions: Functions were implemented inside ${ .. } variable references, and supply many useful capabilities.
- Expressions: An expression evaluation engine handles items wrapped inside $[...]. This includes some string manipulation facilities, arithmetic expressions, etc.
- Application Gateway Interface: Asterisk can fork external processes that communicate via pipe. AGI applications can be written in any language. Very powerful applications can be added this way.
- Variables: Channels of communication have variables associated with them, and asterisk provides some global variables. These can be manipulated and/or consulted by the above mechanisms.

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 command" 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
```

Device Status Reports with AMI

blank

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 Manger (AJAM)

AJAM is a new technology which allows web browsers or other HTTP enabled applications and web pages to directly access the Asterisk Manger Interface (AMI) via HTTP. Setting up your server to process AJAM involves a few steps:

Setting up the Asterisk 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 "asterisk" and the rest of these instructions assume that value.

Allow Manager Access via HTTP

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

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:

- http://localhost:8088/asterisk/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/asterisk/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/asterisk/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/asterisk/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 Queues

Pardon, but the dialplan in this tutorial will be expressed in AEL, the new Asterisk Extension Language. If you are not used to its syntax, we hope you will find it to some degree intuitive. If not, there are documents explaining its syntax and constructs.

Configuring Call Queues

Top-level for configuring call queues

Using queues.conf

First of all, set up call queues in queue.conf
Here is an example:

queues.conf

```
; Cool Digium Queues
[general]
persistentmembers = yes

; General sales queue
[sales-general]
music=default
context=sales
strategy=ringall
joinempty=strict
leavewhenempty=strict

; Customer service queue
[customerservice]
music=default
context=customerservice
strategy=ringall
joinempty=strict
leavewhenempty=strict

; Support dispatch queue
[dispatch]
music=default
context=dispatch
strategy=ringall
joinempty=strict
leavewhenempty=strict
```

In the above, we have defined 3 separate calling queues: sales-general, customerservice, and dispatch.

Please note that the sales-general queue specifies a context of "sales", and that customerservice specifies the context of "customerservice", and the dispatch queue specifies the context "dispatch". These three contexts must be defined somewhere in your dialplan. We will show them after the main menu below.

In the [general] section, specifying the persistentmembers=yes, will cause the agent lists to be stored in astdb, and recalled on startup.

The strategy=ringall will cause all agents to be dialed together, the first to answer is then assigned the incoming call.

"joinempty" set to "strict" will keep incoming callers from being placed in queues where there are no agents to take calls. The Queue() application will return, and the dial plan can determine what to do next.

If there are calls queued, and the last agent logs out, the remaining incoming callers will immediately be removed from the queue, and the Queue() call will return, IF the "leavewhenempty" is set to "strict".

Routing Incoming Calls to Queues

Then in extensions.ael, you can do these things:

The Main Menu
At Digium, incoming callers are sent to the "mainmenu" context, where they are greeted, and directed to the numbers they choose...

```
context mainmenu {
    includes {
        digium;
        queues-loginout;
    }
    0 => goto dispatch,s,1;
    2 => goto sales,s,1;
    3 => goto customerservice,s,1;
    4 => goto dispatch,s,1;
    s => {
        Ringing();
        Wait(1);
        Set(attempts=0);
        Answer();
        Wait(1);
        Background(digium/ThankYouForCallingDigium);
        Background(digium/YourOpenSourceTelecommunicationsSupplier);
        WaitExten(0.3);
    repeat:
        Set(attempts=$[${attempts} + 1]);
        Background(digium/IfYouKnowYourPartysExtensionYouMayDialItAtAnyTime);
        WaitExten(0.1);
        Background(digium/Otherwise);
        WaitExten(0.1);
        Background(digium/ForSalesPleasePress2);
        WaitExten(0.2);
        Background(digium/ForCustomerServicePleasePress3);
        WaitExten(0.2);
        Background(digium/ForAllOtherDepartmentsPleasePress4);
        WaitExten(0.2);
        Background(digium/ToSpeakWithAnOperatorPleasePress0AtAnyTime);
        if( ${attempts} < 2 ) {
            WaitExten(0.3);
            Background(digium/ToHearTheseOptionsRepeatedPleaseHold);
        }
        WaitExten(5);
        if( ${attempts} < 2 ) goto repeat;
        Background(digium/YouHaveMadeNoSelection);
        Background(digium/ThisCallWillBeEnded);
        Background(goodbye);
        Hangup();
    }
}
```

The Contexts referenced from the queues.conf file

```
context sales {
    0 => goto dispatch,s,1;
    8 => Voicemail(${SALESVM});
    s => {
        Ringing();
        Wait(2);
        Background(digium/ThankYouForContactingTheDigiumSalesDepartment);
        WaitExten(0.3);

Background(digium/PleaseHoldAndYourCallWillBeAnsweredByOurNextAvailableSalesRepresentativ
e);
        WaitExten(0.3);
        Background(digium/AtAnyTimeYouMayPress0ToSpeakWithAnOperatorOr8ToLeaveAMessage);
        Set(CALLERID(name)=Sales);
        Queue(sales-general,t);
        Set(CALLERID(name)=EmptySalQ);
        goto dispatch,s,1;
        Playback(goodbye);
        Hangup();
    }
}
```

Please note that there is only one attempt to queue a call in the sales queue. All sales agents that are logged in will be rung.

```
context customerservice {
    0 => {
        SetCIDName(CSVTrans);
        goto dispatch|s|1;
    }
    8 => Voicemail(${CUSTSERVVM});
    s => {
        Ringing();
        Wait(2);
        Background(digium/ThankYouForCallingDigiumCustomerService);
        WaitExten(0.3);
        notracking:
Background(digium/PleaseWaitForTheNextAvailableCustomerServiceRepresentative);
        WaitExten(0.3);
        Background(digium/AtAnyTimeYouMayPress0ToSpeakWithAnOperatorOr8ToLeaveAMessage);
        Set(CALLERID(name)=Cust Svc);
        Set(QUEUE_MAX_PENALTY=10);
        Queue(customerservice,t);
        Set(QUEUE_MAX_PENALTY=0);
        Queue(customerservice,t);
        Set(CALLERID(name)=EmptyCSVQ);
        goto dispatch,s,1;
        Background(digium/NoCustomerServiceRepresentativesAreAvailableAtThisTime);
        Background(digium/PleaseLeaveAMessageInTheCustomerServiceVoiceMailBox);
        Voicemail(${CUSTSERVVM});
        Playback(goodbye);
        Hangup();
    }
}
```

Note that calls coming into customerservice will first be try to queue calls to those agents with a QUEUE_MAX_PENALTY of 10, and if none are available, then all agents are rung.

```
context dispatch {
    s => {
        Ringing();
        Wait(2);
        Background(digium/ThankYouForCallingDigium);
        WaitExten(0.3);
        Background(digium/YourCallWillBeAnsweredByOurNextAvailableOperator);
        Background(digium/PleaseHold);
        Set(QUEUE_MAX_PENALTY=10);
        Queue(dispatch|t);
        Set(QUEUE_MAX_PENALTY=20);
        Queue(dispatch|t);
        Set(QUEUE_MAX_PENALTY=0);
        Queue(dispatch|t);
        Background(digium/NoOneIsAvailableToTakeYourCall);
        Background(digium/PleaseLeaveAMessageInOurGeneralVoiceMailBox);
        Voicemail(${DISPATCHVM});
        Playback(goodbye);
        Hangup();
    }
}
```

And in the dispatch context, first agents of priority 10 are tried, then 20, and if none are available, all agents are tried.

Notice that a common pattern is followed in each of the three queue contexts:

First, you set QUEUE_MAX_PENALTY to a value, then you call Queue(queue-name,option,...) (see the Queue application documetation for details)

In the above, note that the "t" option is specified, and this allows the agent picking up the incoming call the luxury of transferring the call to other parties.

The purpose of specifying the QUEUE_MAX_PENALTY is to develop a set of priorities amongst agents. By the above usage, agents with lower number priorities will be given the calls first, and then, if no-one picks up the call, the QUEUE_MAX_PENALTY will be incremented, and the queue tried again. Hopefully, along the line, someone will pick up the call, and the Queue application will end with a hangup.

The final attempt to queue in most of our examples sets the QUEUE_MAX_PENALTY to zero, which means to try all available agents.

Assigning Agents to Queues

In this example dialplan, we want to be able to add and remove agents to handle incoming calls, as they feel they are available. As they log in, they are added to the queue's agent list, and as they log out, they are removed. If no agents are available, the queue command will terminate, and it is the duty of the dialplan to do something appropriate, be it sending the incoming caller to voicemail, or trying the queue again with a higher QUEUE_MAX_PENALTY.

Because a single agent can make themselves available to more than one queue, the process of joining multiple queues can be handled automatically by the dialplan.

Agents Log In and Out

```
context queues-loginout {
    6092 => {
        Answer();
        Read(AGENT_NUMBER,agent-enternum);
        VMAuthenticate(${AGENT_NUMBER}@default,s);
        Set(queue-announce-success=1);
        goto queues-manip,I${AGENT_NUMBER},1;
    }
    6093 => {
        Answer();
        Read(AGENT_NUMBER,agent-enternum);
        Set(queue-announce-success=1);
        goto queues-manip,O${AGENT_NUMBER},1;
    }
}
```

In the above contexts, the agents dial 6092 to log into their queues, and they dial 6093 to log out of their queues. The agent is prompted for their agent number, and if they are logging in, their passcode, and then they are transferred to the proper extension in the queues-manip context. The queues-manip context does all the actual work:

```
context queues-manip {
    // Raquel Squelch
    _[IO]6121 => {
        &queue-addremove(dispatch,10,${EXTEN});
        &queue-success(${EXTEN});
    }
    // Brittanica Spears
    _[IO]6165 => {
        &queue-addremove(dispatch,20,${EXTEN});
        &queue-success(${EXTEN});
    }
    // Rock Hudson
    _[IO]6170 => {
        &queue-addremove(sales-general,10,${EXTEN});
        &queue-addremove(customerservice,20,${EXTEN});
        &queue-addremove(dispatch,30,${EXTEN});
        &queue-success(${EXTEN});
    }
    // Saline Dye-on
    _[IO]6070 => {
        &queue-addremove(sales-general,20,${EXTEN});
        &queue-addremove(customerservice,30,${EXTEN});
        &queue-addremove(dispatch,30,${EXTEN});
        &queue-success(${EXTEN});
    }
}
```

In the above extensions, note that the queue-addremove macro is used to actually add or remove the agent from the applicable queue, with the applicable priority level. Note that agents with a priority level of 10 will be called before agents with levels of 20 or 30.

In the above example, Raquel will be dialed first in the dispatch queue, if she has logged in. If she is not, then the second call of Queue() with priority of 20 will dial Brittanica if she is present, otherwise the third call of Queue() with MAX_PENALTY of 0 will dial Rock and Saline simultaneously.

Also note that Rock will be among the first to be called in the sales-general queue, and among the last in the dispatch queue. As you can see in main menu, the callerID is set in the main menu so they can tell which queue incoming calls are coming from.

The call to queue-success() gives some feedback to the agent as they log in and out, that the process has completed.

```
macro queue-success(exten) {
    if( ${queue-announce-success} > 0 ) {
        switch(${exten:0:1}) {
            case I:
                Playback(agent-loginok);
                Hangup();
                break;
            case O:
                Playback(agent-loggedoff);
                Hangup();
                break;
        }
    }
}
```

The queue-addremove macro is defined in this manner:

```
macro queue-addremove(queuename,penalty,exten) {
    switch(${exten:0:1}) {
        case I: // Login
            AddQueueMember(${queuename},Local/${exten:1}@agents,${penalty});
            break;
        case O: // Logout
            RemoveQueueMember(${queuename},Local/${exten:1}@agents);
            break;
        case P: // Pause
            PauseQueueMember(${queuename},Local/${exten:1}@agents);
            break;
        case U: // Unpause
            UnpauseQueueMember(${queuename},Local/${exten:1}@agents);
            break;
        default: // Invalid
            Playback(invalid);
            break;
    }
}
```

Basically, it uses the first character of the exten variable, to determine the proper actions to take. In the above dial plan code, only the cases I or O are used, which correspond to the Login and Logout actions.

Controlling the way Queues Call Agents

Notice in the above, that the commands to manipulate agents in queues have "@agents" in their arguments. This is a reference to the agents context:

```
context agents {
    // General sales queue
    8010 => {
        Set(QUEUE_MAX_PENALTY=10);
        Queue(sales-general,t);
        Set(QUEUE_MAX_PENALTY=0);
        Queue(sales-general,t);
        Set(CALLERID(name)=EmptySalQ);
       goto dispatch,s,1;
    }
    // Customer Service queue
    8011 => {
        Set(QUEUE_MAX_PENALTY=10);
        Queue(customerservice,t);
        Set(QUEUE_MAX_PENALTY=0);
        Queue(customerservice,t);
        Set(CALLERID(name)=EMptyCSVQ);
        goto dispatch,s,1;
    }
    8013 => {
        Dial(iax2/sweatshop/9456@from-ecstacy);
        Set(CALLERID(name)=EmptySupQ);
        Set(QUEUE_MAX_PENALTY=10);
        Queue(support-dispatch,t);
        Set(QUEUE_MAX_PENALTY=20);
        Queue(support-dispatch,t);
        Set(QUEUE_MAX_PENALTY=0); // means no max
        Queue(support-dispatch,t);
        goto dispatch,s,1;
    }
    6121 => &callagent(${RAQUEL},${EXTEN});
    6165 => &callagent(${SPEARS},${EXTEN});
    6170 => &callagent(${ROCK},${EXTEN});
    6070 => &callagent(${SALINE},${EXTEN});
}
```

In the above, the variables ${RAQUEL}, etc stand for actual devices to ring that person's phone (like DAHDI/37).

The 8010, 8011, and 8013 extensions are purely for transferring incoming callers to queues. For instance, a customer service agent might want to transfer the caller to talk to sales. The agent only has to transfer to extension 8010, in this case.

Here is the callagent macro, note that if a person in the queue is called, but does not answer, then they are automatically removed from the queue.

```
macro callagent(device,exten) {
    if( ${GROUP_COUNT(${exten}@agents)}=0 ) {
        Set(OUTBOUND_GROUP_ONCE=${exten}@agents);
        Dial(${device},300,t);
        switch(${DIALSTATUS}) {
            case BUSY:
                Busy();
                break;
            case NOANSWER:
                Set(queue-announce-success=0);
                goto queues-manip,O${exten},1;
            default:
                Hangup();
                break;
        }
    }
    else {
        Busy();
    }
}
```

In the callagent macro above, the ${exten} will be 6121, or 6165, etc, which is the extension of the agent.

The use of the GROUP_COUNT, and OUTBOUND_GROUP follow this line of thinking. Incoming calls can be queued to ring all agents in the current priority. If some of those agents are already talking, they would get bothersome call-waiting tones. To avoid this inconvenience, when an agent gets a call, the OUTBOUND_GROUP assigns that conversation to the group specified, for instance 6171@agents. The ${GROUP_COUNT()} variable on a subsequent call should return "1" for that group. If GROUP_COUNT returns 1, then the busy() is returned without actually trying to dial the agent.

Queue Pre-Acknowledgement Messages

If you would like to have a pre acknowledge message with option to reject the message you can use the following dialplan Macro as a base with the 'M' dial argument.

```
[macro-screen]
exten=>s,1,Wait(.25)
exten=>s,2,Read(ACCEPT,screen-callee-options,1)
exten=>s,3,Gotoif($[${ACCEPT} = 1] ?50)
exten=>s,4,Gotoif($[${ACCEPT} = 2] ?30)
exten=>s,5,Gotoif($[${ACCEPT} = 3] ?40)
exten=>s,6,Gotoif($[${ACCEPT} = 4] ?30:30)
exten=>s,30,Set(MACRO_RESULT=CONTINUE)
exten=>s,40,Read(TEXTEN,custom/screen-exten,)
exten=>s,41,Gotoif($[${LEN(${TEXTEN})} = 3]?42:45)
exten=>s,42,Set(MACRO_RESULT=GOTO:from-internal^${TEXTEN}^1)
exten=>s,45,Gotoif($[${TEXTEN} = 0] ?46:4)
exten=>s,46,Set(MACRO_RESULT=CONTINUE)
exten=>s,50,Playback(after-the-tone)
exten=>s,51,Playback(connected)
exten=>s,52,Playback(beep)
```

Queue Caveats

In the above examples, some of the possible error checking has been omitted, to reduce clutter and make the examples clearer.

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)
- 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 amoutn 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.

Asterisk Security Framework

Attacks on Voice over IP networks are becoming increasingly more common. It has become clear that we must do something within Asterisk to help mitigate these attacks.

Through a number of discussions with groups of developers in the Asterisk community, the general consensus is that the best thing that we can do within Asterisk is to build a framework which recognizes and reports events that could potentially have security implications. Each channel driver has a different concept of what is an "event", and then each administrator has different thresholds of what is a "bad" event and what is a restorative event. The process of acting upon this information is left to an external program to correlate and then take action - block traffic, modify dialing rules, etc. It was decided that embedding actions inside of Asterisk was inappropriate, as the complexity of construction of such rule sets is difficult and there was no agreement on where rules should be enabled or how they should be processed. The addition of a major section of code to handle rule expiration and severity interpretation was significant. As a final determining factor, there are external programs and services which already parse log files and act in concert with packet filters or external devices to protect or alter network security models for IP connected hosts.

Security Framework Overview

This section discusses the architecture of the Asterisk modifications being proposed.
There are two main components that we propose for the initial implementation of the security framework:

- Security Event Generation
- Security Event Logger

Security Event Generation

The ast_event API is used for the generation of security events. That way, the events are in an easily interpretable format within Asterisk to make it easy to write modules that do things with them. There are also some helper data structures and functions to aid Asterisk modules in reporting these security events with the proper contents.

The next section of this document contains the current list of security events being proposed. Each security event type has some required pieces of information and some other optional pieces of information.

Subscribing to security events from within Asterisk can be done by subscribing to events of type AST_EVENT_SECURITY. These events have an information element, AST_EVENT_IE_SECURITY_EVENT, which identifies the security event sub-type (from the list described in the next section). The result of the information elements in the events contain the required and optional meta data associated with the event sub-type.

Asterisk Security Event Logger

In addition to the infrastructure for generating the events, one module that is a consumer of these events has been implemented.

Asterisk trunk was recently updated to include support for dynamic logger levels. This module takes advantage of this functionality to create a custom "security" logger level. Then, when this module is in use, logger.conf can be configured to put security events into a file

security_log => security

The content of this file is a well defined and easily interpretable format for external scripts to read and act upon. The definition for the format of the log file is described later in this chapter.

Security Events to Log

```
(-) required
(+) optional

Invalid Account ID
  (-) Local address family/IP address/port/transport
  (-) Remote address family/IP address/port/transport
  (-) Service (SIP, AMI, IAX2, ...)
  (-) System Name
  (+) Module
  (+) Account ID (username, etc)
  (+) Session ID (CallID, etc)
  (+) Session timestamp (required if Session ID present)
  (-) Event timestamp (sub-second precision)
Failed ACL match
  -> everything from invalid account ID
  (+) Name of ACL (when we have named ACLs)

Invalid Challenge/Response
  -> everything from invalid account ID
  (-) Challenge
  (-) Response
  (-) Expected Response
Invalid Password
  -> everything from invalid account ID

Successful Authentication
  -> informational event
  -> everything from invalid account ID

Invalid formatting of Request
  -> everything from invalid account ID
  -> account ID optional
  (-) Request Type
  (+) Request parameters
Session Limit Reached (such as a call limit)
  -> everything from invalid account ID

Memory Limit Reached
  -> everything from invalid account ID
Maximum Load Average Reached
  -> everything from invalid account ID

Request Not Allowed
  -> everything from invalid account ID
  (-) Request Type
  (+) Request parameters
Request Not Supported
  -> everything from invalid account ID
  (-) Request Type

Authentication Method Not Allowed
  -> everything from invalid account ID
  (-) Authentication Method attempted
In dialog message from unexpected host
  -> everything from invalid account ID
  (-) expected host
```

Security Log File Format

The beginning of each line in the log file is the same as it is for other logger levels within Asterisk.

```
[Feb 11 07:57:03] SECURITY[23736] res_security_log.c: <...>
```

The part of the log entry identified by \<...\> is where the security event content resides. The security event content is a comma separated list of key value pairs. The key is the information element type, and the value is a quoted string that contains the associated meta data for that information element. Any embedded quotes within the content are escaped with a backslash.

INFORMATION_ELEMENT_1="IE1 content",INFORMATION_ELEMENT_2="IE2 content"

The following table includes potential information elements and what the associated content looks like:

- IE: SecurityEvent
  Content: This is the security event sub-type.
  Values: FailedACL, InvalidAccountID, SessionLimit, MemoryLimit, LoadAverageLimit, RequestNotSupported, RequestNotAllowed, AuthMethodNotAllowed, ReqBadFormat, UnexpectedAddress, ChallengeResponseFailed, InvalidPassword

- IE: EventVersion
  Content: This is a numeric value that indicates when updates are made to the content of the event.
  Values: Monotonically increasing integer, starting at 1

- IE: Service
  Content: This is the Asterisk service that generated the event.
  Values: TEST, SIP, AMI

- IE: Module
  Content: This is the Asterisk module that generated the event.
  Values: chan_sip

- IE: AccountID
  Content: This is a string used to identify the account associated with the event. In most cases, this would be a username.

- IE: SessionID
  Content: This is a string used to identify the session associated with the event. The format of the session identifier is specific to the service. In the case of SIP, this would be the Call-ID.

- IE: SessionTV
  Content: The time that the session associated with the SessionID started.
  Values: <seconds><microseconds> since epoch

- IE: ACLName
  Content: This is a string that identifies which named ACL is associated with this event.

- IE: LocalAddress
  Content: This is the local address that was contacted for the related event.
  Values: <Address Family>/<Transport>/<Address>/<Port>
  Examples: -> IPV4/UDP/192.168.1.1/5060 -> IPV4/TCP/192.168.1.1/5038

- IE: RemoteAddress
  Content: This is the remote address associated with the event.
  Examples: -> IPV4/UDP/192.168.1.2/5060 -> IPV4/TCP/192.168.1.2/5038

- IE: ExpectedAddress
  Content: This is the address that was expected to be the remote address.
  Examples: -> IPV4/UDP/192.168.1.2/5060 -> IPV4/TCP/192.168.1.2/5038

- IE: EventTV
  Content: This is the timestamp of when the event occurred.
  Values: <seconds><microseconds> since epoch

- IE: RequestType
  Content: This is a service specific string that represents the invalid request

- IE: RequestParams

Content: This is a service specific string that represents relevant parameters given with a request that was considered invalid.

- IE: AuthMethod
  Content: This is a service specific string that represents an authentication method that was used or requested.
- IE: Challenge
  Content: This is a service specific string that represents the challenge provided to a user attempting challenge/response authentication.
- IE: Response
  Content: This is a service specific string that represents the response received from a user attempting challenge/response authentication.
- IE: ExpectedResponse
  Content: This is a service specific string that represents the response that was expected to be received from a user attempting challenge/response authentication.

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 additional 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

Call Completion Supplementary Services (CCSS)

Introduction

A new feature for Asterisk 1.8 is Call Completion Supplementary Services. This document aims to explain the system and how to use it. In addition, this document examines some potential troublesome points which administrators may come across during their deployment of the feature.

What is CCSS?

Call Completion Supplementary Services (often abbreviated "CCSS" or simply "CC") allow for a caller to let Asterisk automatically alert him when a called party has become available, given that a previous call to that party failed for some reason. The two services offered are Call Completion on Busy Subscriber (CCBS) and Call Completion on No Response (CCNR). To illustrate, let's say that Alice attempts to call Bob. Bob is currently on a phone call with Carol, though, so Alice hears a busy signal. In this situation, assuming that Asterisk has been configured to allow for such activity, Alice would be able to request CCBS. Once Bob has finished his phone call, Alice will be alerted. Alice can then attempt to call Bob again.

CCSS Glossary

In this document, we will use some terms which may require clarification. Most of these terms are specific to Asterisk, and are by no means standard.

- CCBS: Call Completion on Busy Subscriber. When a call fails because the recipient's phone is busy, the caller will have the opportunity to request CCBS. When the recipient's phone is no longer busy, the caller will be alerted. The means by which the caller is alerted is dependent upon the type of agent used by the caller.
- CCNR: Call Completion on No Response. When a call fails because the recipient does not answer the phone, the caller will have the opportun- ity to request CCNR. When the recipient's phone becomes busy and then is no longer busy, the caller will be alerted. The means by which the caller is alerted is dependent upon the type of the agent used by the caller.
- Agent: The agent is the entity within Asterisk that communicates with and acts on behalf of the calling party.
- Monitor: The monitor is the entity within Asterisk that communicates with and monitors the status of the called party.
- Generic Agent: A generic agent is an agent that uses protocol-agnostic methods to communicate with the caller. Generic agents should only be used for phones, and never should be used for "trunks."
- Generic Monitor: A generic monitor is a monitor that uses protocol- agnostic methods to monitor the status of the called party. Like with generic agents, generic monitors should only be used for phones.
- Native Agent: The opposite of a generic agent. A native agent uses protocol-specific messages to communicate with the calling party. Native agents may be used for both phones and trunks, but it must be known ahead of time that the device with which Asterisk is communica- ting supports the necessary signaling.
- Native Monitor: The opposite of a generic monitor. A native monitor uses protocol-specific messages to subscribe to and receive notifica- tion of the status of the called party. Native monitors may be used for both phones and trunks, but it must be known ahead of time that the device with which Asterisk is communicating supports the necessary signaling.
- Offer: An offer of CC refers to the notification received by the caller that he may request CC.
- Request: When the caller decides that he would like to subscribe to CC, he will make a request for CC. Furthermore, the term may refer to any outstanding requests made by callers.
- Recall: When the caller attempts to call the recipient after being alerted that the recipient is available, this action is referred to as a "recall."

The Call Completion Process

The Initial Call

The only requirement for the use of CC is to configure an agent for the caller and a monitor for at least one recipient of the call. This is controlled using the cc_agent_policy for the caller and the cc_monitor_policy for the recipient. For more information about these configuration settings, see configs/samples/ccss.conf.sample. If the agent for the caller is set to something other than "never" and at least one recipient has his monitor set to something other than "never," then CC will be offered to the caller at the end of the call.

Once the initial call has been hung up, the configured cc_offer_timer for the caller will be started. If the caller wishes to request CC for the previous call, he must do so before the timer expires.

Requesting CC

Requesting CC is done differently depending on the type of agent the caller is using.

With generic agents, the CallCompletionRequest application must be called in order to request CC. There are two different ways in which this may be called. It may either be called before the caller hangs up during the initial call, or the caller may hang up from the initial call and dial an extension which calls the CallCompletionRequest application. If the second method is used, then the caller will have until the cc_offer_timer expires to request CC.

With native agents, the method for requesting CC is dependent upon the technology being used, coupled with the make of equipment. It may be possible to request CC using a programmable key on a phone or by clicking a button on a console. If you are using equipment which can natively support CC but do not know the means by which to request it, then contact the equipment manufacturer for more information.

Cancelling CC

CC may be canceled after it has been requested. The method by which this is accomplished differs based on the type of agent the calling party uses.

When using a generic agent, the dialplan application CallRequestCancel is used to cancel CC. When using a native monitor, the method by which CC is cancelled depends on the protocol used. Likely, this will be done using a button on a phone.

Keep in mind that if CC is cancelled, it cannot be un-cancelled.

Monitoring the Called Party

Once the caller has requested CC, then Asterisk's job is to monitor the progress of the called parties. It is at this point that Asterisk allocates the necessary resources to monitor the called parties.

A generic monitor uses Asterisk's device state subsystem in order to determine when the called party has become available. For both CCBS and CCNR, Asterisk simply waits for the phone's state to change to a "not in use" state from a different state. Once this happens, then Asterisk will consider the called party to be available and will alert the caller.

A native monitor relies on the network to send a protocol-specific message when the called party has become available. When Asterisk receives such a message, it will consider the called party to be available and will alert the caller.

Note that since a single caller may dial multiple parties, a monitor is used for each called party. It is within reason that different called parties will use different types of monitors for the same CC request.

Alerting the Caller

Once Asterisk has determined that the called party has become available the time comes for Asterisk to alert the caller that the called party has become available. The method by which this is done differs based on the type of agent in use.

If a generic agent is used, then Asterisk will originate a call to the calling party. Upon answering the call, if a callback macro has been configured, then that macro will be executed on the calling party's channel. After the macro has completed, an outbound call will be issued to the parties involved in the original call.

If a native agent is used, then Asterisk will send an appropriate notification message to the calling party to alert it that it may now attempt its recall. How this is presented to the caller is dependent upon the protocol and equipment that the caller is using. It is possible that the calling party's phone will ring and a recall will be triggered upon answering the phone, or it may be that the user has a specific button that he may press to initiate a recall.

If the Caller is unavailable

When the called party has become available, it is possible that when Asterisk attempts to alert the calling party of the called party's availability, the calling party itself will have become unavailable. If this is the case, then Asterisk will suspend monitoring of the called party and will instead monitor the availability of the calling party. The monitoring procedure for the calling party is the same as is used in the section "Monitoring the Called Party." In other words, the method by which the calling party is monitored is dependent upon the type of agent used by the caller.

Once Asterisk has determined that the calling party has become available again, Asterisk will then move back to the process used in the section "Monitoring the Called Party."

The CC recall

The calling party will make its recall to the same extension that was dialed. Asterisk will provide a channel variable, CC_INTERFACES, to be used as an argument to the Dial application for CC recalls. It is strongly recommended that you use this channel variable during a CC recall. Listed are two reasons:

1. The dialplan may be written in such a way that the dialed destintations are dynamically generated. With such a dialplan, it cannot be guaranteed that the same interfaces will be recalled.
2. For calling destinations with native CC monitors, it may be necessary to dial a special string in order to notify the channel driver that the number being dialed is actually part of a CC recall.

Even if your call gets routed through local channels, the CC_INTERFACES variable will be populated with the appropriate values for that specific extension.

When the called parties are dialed, it is expected that a called party will answer, since Asterisk had previously determined that the party was available. However, it is possible that the called party may choose not to respond to the call, or he could have become busy again. In such a situation, the calling party must re-request CC if he wishes to still be alerted when the calling party has become available.

Call Completion Info and Tips

- Be aware when using a generic agent that the max_cc_agents configuration parameter is ignored. The main driving reason for this is that the mechanism for cancelling CC when using a generic agent would become much more potentially confusing to execute. By limiting a calling party to having a single request, there is only ever a single request to be cancelled, making the process simple.
- Keep in mind that no matter what CC agent type is being used, a CC request can only be made for the latest call issued.
- If available timers are running on multiple called parties, it is possible that one of the timers may expire before the others do. If such a situation occurs, then the interface on which the timer expired will cease to be monitored. If, though, one of the other called parties becomes available before his available timer expires, the called party whose available timer had previously expired will still be included in the CC_INTERFACES channel variable on the recall.
- It is strongly recommended that lots of thought is placed into the settings of the CC timers. Our general recommendation is that timers for phones should be set shorter than those for trunks. The reason for this is that it makes it less likely for a link in the middle of a network to cause CC to fail.
- CC can potentially be a memory hog if used irresponsibly. The following are recommendations to help curb the amount of resources required by the CC engine. First, limit the maximum number of CC requests in the system using the cc_max_requests option in ccss.conf. Second, set the cc_offer_timer low for your callers. Since it is likely that most calls will not result in a CC request, it is a good idea to set this value to something low so that information for calls does not stick around in memory for long. The final thing that can be done is to conditionally set the cc_agent_policy to "never" using the CALLCOMPLETION dialplan function. By doing this, no CC information will be kept around after the call completes.
- It is possible to request CCNR on answered calls. The reason for this is that it is impossible to know whether a call that is answered has actually been answered by a person or by something such as voicemail or some other IVR.
- Not all channel drivers have had the ability to set CC config parameters in their configuration files added yet. At the time of this writing (2009 Oct), only chan_sip has had this ability added, with short-term plans to add this to chan_dahdi as well. It is possible to set CC configuration parameters for other channel types, though. For these channel types, the setting of the parameters can only be accomplished using the CALLCOMPLETION dialplan function.
- It is documented in many places that generic agents and monitors can only be used for phones. In most cases, however, Asterisk has no way of distinguishing between a phone and a trunk itself. The result is that Asterisk will happily let you violate the advice given and allow you to set up a trunk with a generic monitor or agent. While this will not cause anything catastrophic to occur, the behavior will most definitely not be what you want.
- At the time of this writing (2009 Oct), Asterisk is the only known SIP stack to write an implementation of draft-ietf-bliss-call-completion-04. As a result, it is recommended that for your SIP phones, use a generic agent and monitor. For SIP trunks, you will only be able to use CC if the other end is terminated by another Asterisk server running version 1.8 or later.
- Native SIP CC will only work if the xml2 development library is installed. This is because we use libxml2 in order to parse PIDF bodies of PUBLISH messages received. If, at configure time, Asterisk cannot detect that the necessary library is installed, then native CC in SIP will be disabled. Attempts to set a channel or SIP peer to use native CC will be changed to having CC being disabled instead.
- If the Dial application is called multiple times by a single extension, CC will only be offered to the caller for the parties called by the first instantiation of Dial.
- If a phone forwards a call, then CC may only be requested for the phone that executed the call forward. CC may not be requested for the phone to which the call was forwarded.
- CC is currently only supported by the Dial application. Queue, Followme, and Page do not support CC because it is not particularly useful for those applications.

> - Generic CC relies heavily on accurate device state reporting. In particular, when using SIP phones it is vital to be sure that device state is updated properly when using them. In order to facilitate proper device state handling, be sure to set callcounter=yes for all peers and to set limitonpeers=yes in the general section of sip.conf

- When using SIP CC (i.e. native CC over SIP), it is important that your minexpiry and maxexpiry values allow for available timers to run as little or as long as they are configured. When an Asterisk server requests call completion over SIP, it sends a SUBSCRIBE message with an Expires header set to the number of seconds that the available timer should run. If the Asterisk server that receives this SUBSCRIBE has a maxexpiry set lower than what is in the received Expires header, then the available timer will only run for maxexpiry seconds.
- CC support for ETSI PTP and Q.SIG requires CallerID information to match CC requests with CC offers. For Q.SIG, depending upon the options negotiated when CC is requested, the CallerID information needs to be callable as well.

- As with all Asterisk components, CC is not perfect. If you should find a bug or wish to enhance the feature, please open an issue on https://issues.asterisk.org. If writing an enhancement, please be sure to include a patch for the enhancement, or else the issue will be closed.

Generic Call Completion Example

The following is an incredibly bare-bones example sip.conf and dialplan to show basic usage of generic call completion. It is likely that if you have a more complex setup, you will need to make use of items like the CALLCOMPLETION dialplan function or the CC_INTERFACES channel variable.
First, let's establish a very simple sip.conf to use for this

sip.conf

```
[Mark]
context=phone_calls
cc_agent_policy=generic
cc_monitor_policy=generic ;We will accept defaults for the rest of the cc parameters
;We also are not concerned with other SIP details for this
;example

[Richard]
context=phone_calls
cc_agent_policy=generic
cc_monitor_policy=generic
```

Now, let's write a simple dialplan

extensions.conf

```
[phone_calls]
exten => 1000,1,Dial(SIP/Mark,20)
exten => 1000,n,Hangup
exten => 2000,1,Dial(SIP/Richard,20)
exten => 2000,n,Hangup
exten => 30,1,CallCompletionRequest
exten => 30,n,Hangup
exten => 31,1,CallCompletionCancel
exten => 31,n,Hangup
```

Scenario 1: Mark picks up his phone and dials Richard by dialing 2000. Richard is currently on a call, so Mark hears a busy signal. Mark then hangs up, picks up the phone and dials 30 to call the CallCompletionRequest application. After some time, Richard finishes his call and hangs up. Mark is automatically called back by Asterisk. When Mark picks up his phone, Asterisk will dial extension 2000 for him.

Scenario 2: Richard picks up his phone and dials Mark by dialing 1000. Mark has stepped away from his desk, and so he is unable to answer the phone within the 20 second dial timeout. Richard hangs up, picks the phone back up and then dials 30 to request call completion. Mark gets back to his desk and dials somebody's number. When Mark finishes the call, Asterisk detects that Mark's phone has had some activity and has become available again and rings Richard's phone. Once Richard picks up, Asterisk automatically dials exteision 1000 for him.

Scenario 3: Much like scenario 1, Mark calls Richard and Richard is busy. Mark hangs up, picks the phone back up and then dials 30 to request call completion. After a little while, Mark realizes he doesn't actually need to talk to Richard, so he dials 31 to cancel call completion. When Richard becomes free, Mark will not automatically be redialed by Asterisk.

Scenario 4: Richard calls Mark, but Mark is busy. About thirty seconds later, Richard decides that he should perhaps request call completion. However, since Richard's phone has the default cc_offer_timer of 20 seconds, he has run out of time to request call completion. He instead must attempt to dial Mark again manually. If Mark is still busy, Richard can attempt to request call completion on this second call instead.

Scenario 5: Mark calls Richard, and Richard is busy. Mark requests call completion. Richard does not finish his current call for another 2 hours (7200 seconds). Since Mark has the default ccbs_available_timer of 4800 seconds set, Mark will not be automatically recalled by Asterisk when Richard finishes his call.

Scenario 6: Mark calls Richard, and Richard does not respond within the 20 second dial timeout. Mark requests call completion. Richard does not use his phone again for another 4 hours (144000 seconds). Since Mark has the default ccnr_available_timer of 7200 seconds set, Mark will not be automatically recalled by Asterisk when Richard finishes his call.

Call Detail Records (CDR)

Top-level page for all things CDR

CDR Applications

- SetAccount - Set account code for billing
- SetAMAFlags - Sets AMA flags
- NoCDR - Make sure no CDR is saved for a specific call
- ResetCDR - Reset CDR
- ForkCDR - Save current CDR and start a new CDR for this call
- Authenticate - Authenticates and sets the account code
- SetCDRUserField - Set CDR user field
- AppendCDRUserField - Append data to CDR User field

For more information, use the "core show application application" command. You can set default account codes and AMA flags for devices in channel configuration files, like sip.conf, iax.conf etc.

CDR Fields

- accountcode: What account number to use, (string, 20 characters)
- src: Caller*ID number (string, 80 characters)
- dst: Destination extension (string, 80 characters)
- dcontext: Destination context (string, 80 characters)
- clid: Caller*ID with text (80 characters)
- channel: Channel used (80 characters)
- dstchannel: Destination channel if appropriate (80 characters)
- lastapp: Last application if appropriate (80 characters)
- lastdata: Last application data (arguments) (80 characters)
- start: Start of call (date/time)
- answer: Answer of call (date/time)
- end: End of call (date/time)
- duration: Total time in system, in seconds (integer), from dial to hangup
- billsec: Total time call is up, in seconds (integer), from answer to hangup
- disposition: What happened to the call: ANSWERED, NO ANSWER, BUSY
- amaflags: What flags to use: DOCUMENTATION, BILL, IGNORE etc, specified on a per channel basis like accountcode.
- user field: A user-defined field, maximum 255 characters

In some cases, uniqueid is appended:

- uniqueid: Unique Channel Identifier (32 characters) This needs to be enabled in the source code at compile time

If you use IAX2 channels for your calls, and allow 'full' transfers (not media-only transfers), then when the calls is transferred the server in the middle will no longer be involved in the signaling path, and thus will not generate accurate CDRs for that call. If you can, use media-only transfers with IAX2 to avoid this problem, or turn off transfers completely (although this can result in a media latency increase since the media packets have to traverse the middle server(s) in the call).

In 1.8 and later

In some CDR backends, the following fields may also be supported:

- linkedid: a unique identifier based on uniqueid. Unlike uniqueid, but spreads to other channels as transfers, dials, etc are performed
- peeraccount: the account code of the bridged channel
- sequence: a field that can be combined with uniqueid and linkedid to uniquely identify a 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=voipdpass 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 PostgresSQL
[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

Calling using Google

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

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.

Channel Event Logging (CEL)

Top-level page for all things CEL

CEL Design Goals

CEL, or Channel Event Logging, has been 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, conferenced, forwarded, etc. In general, those doing billing applications in 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.

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.

The design philosophy of CEL is to generate event data that can grouped together to form a billing record. This may not be a simple task, but we hope to provide a few different examples that could be used as a basis for those involved in this effort.

There are definite parallels between Manager 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.

To 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).

This sequence of actions will generate the following annotated list of 42 CEL events:

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","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","","Z
ap/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","","Z
ap/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","","Z
ap/51-1" ;;; 1 second later, bridge formed betw. 151-2 and 151 (152 answers, 151 and 152 convering; 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:

```
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: label1
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 Events and Fields

While CDRs and the Manager are basically both event tracking mechanisms, CEL tries to track only those events that might pertain to billing issues.

Table of CEL Events

Event	Description
CHAN_START	The time a channel was created
CHAN_END	The time a channel was terminated
ANSWER	The time a channel was answered (ie, phone taken off-hook)
HANGUP	The time at which a hangup occurred
CONF_ENTER	The time a channel was connected into a conference room
CONF_EXIT	The time a channel was removed from a conference room
CONF_START	The time the first person enters a conference room
CONF_END	The time the last person left a conference room (and turned out the lights?)
APP_START	The time a tracked application was started
APP_END	the time a tracked application ended
PARK_START	The time a call was parked
PARK_END	Unpark event
BRIDGE_START	The time a bridge is started
BRIDGE_END	The time a bridge is ended
BRIDGE_UPDATE	This is a replacement channel (Masquerade)
3WAY_START	When a 3-way conference starts (usually via attended transfer)
3WAY_END	When one or all exit a 3-way conference
BLINDTRANSFER	When a blind transfer is initiated
ATTENDEDTRANSFER	When an attended transfer is initiated
TRANSFER	Generic transfer initiated; not used yet...?
PICKUP	This channel picked up the peer channel
FORWARD	This channel is being forwarded somewhere else
HOOKFLASH	So far, when a hookflash event occurs on a DAHDI interface
LINKEDID_END	The last channel with the given linkedid is retired
USER_DEFINED	Triggered from the dialplan, and has a name given by the user

Table of CEL Event Fields

Table 11.2: List of CEL Event Fields

Field	Description
eventtype	The name of the event; see the above list.
eventtime	The time the event happened
cid_name	CID name field

cid_num	CID number field
cid_ani	CID ANI field
cid_rdnis	CID RDNIS field
cid_dnid	CID DNID field
exten	The extension in the dialplan
context	The context in the dialplan
channame	The name assigned to the channel in which the event took place
appname	The name of the current application
appdata	The arguments that will be handed to that application
amaflags	The AMA flags associated with the event; user assignable.
accountcode	A user assigned datum (string)
peeraccount	A user assigned datum (string) on the peer.
uniqueid	Each Channel instance gets a unique ID associated with it.
linkedid	the per-call id, spans several events, possibly.
userfield	A user assigned datum (string)
peer	For bridge or other 2-channel events, this would be the other channel name
userdeftype	User defined event name
extra	Extra information associated with the event.

CEL Applications and Functions

Top-level page for information on CEL Applications and Functions

CEL Function

-
  -
    -
      -
        - THIS IS NO LONGER TRUE REWRITE *****

The CEL function parallels the CDR function, for fetching values from the channel or event. It has some notable notable differences, though! For instance, CEL data is not stored on the channel. Well, not much of it, anyway! You can use the CEL function to set the amaflags, accountcode, and userfield, which are stored on the channel.

Channel variables are not available for reading from the CEL function, nor can any variable name other than what's in the list, be set. CDRs have a structure attached to the channel, where the CDR function could access the values stored there, or set the values there. CDRs could store their own variable lists, but CEL has no such storage. There is no reason to store any event information, as they are immediately output to the various backends at the time they are generated.

See the description for the CEL function from the CLI: core show function CEL

Here is a list of all the available channel field names:

- cidname
- userfield
- cidnum
- amaflags
- cidani
- cidrdnis
- ciddnid
- appdata
- exten
- accountcode
- context
- uniqueid
- channame
- appname
- peer
- eventtime
- eventtype

CELGenUserEvent Application

This application allows the dialplan to insert custom events into the event stream.
For more information, in the CLI, type: **core show application CELGenUserEvent**
Its arguments take this format:

```
CELGenUserEvent(eventname)
```

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!

CEL Configuration Files

cel.conf

Generating Billing Information from CEL

-
    -
        -
            - This is the Next Big Task ****

CEL Storage Backends

Right now, the CEL package will support CSV, Customized CSV, ODBC, PGSQL, TDS, Sqlite3, and Radius back ends. See the doc/celdriver.tex file for how to use these back ends.

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 ,
        [channame] [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=voipdpass
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 ,
        [channame] [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.

MySQL CEL Backend

Using MySQL for Channel Event records is supported by using ODBC and the cel_odbc module.

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 PostgresSQL
[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
);
```

SQLite 3 CEL Backend

SQLite version 3 is supported in cel_sqlite3_custom.

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-Cidrdnis",
- "Asterisk-Ciddnid",
- "Asterisk-Exten",
- "Asterisk-Context", The destination context
- "Asterisk-Channame", 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)

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.

About 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}. For example, to stringwise append $lala to $blabla and store result in $koko, do:

```
exten => 1,2,Set(koko=${blabla}${lala})
```

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(koko=lala) exten => 1,3,Set(${koko}=blabla)
```

stores to the variable "koko" the value "lala" and to variable "lala" the value "blabla".
In fact, everything contained ${here} is just replaced with the value of the variable "here".

Variable Inheritance

Variable names which are prefixed by "" 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. If the name is prefixed by "" in the channel, then the variable is inherited and the "_" will remain intact in the new channel.

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.

Variable Inheritance Examples

```
Set(__FOO=bar)
```

Sets an inherited version of "FOO" variable Set(FOO=bar), Removes the inherited version and sets a local variable.

However, NoOp(${__FOO}) is identical to NoOp(${FOO})

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})
```

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 corresponing 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 invtervening 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.

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.

Conditionals

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)
```

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 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 ôperator 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, ${VAR1} : (WhoWhat)+ may have have worked before, but unless you wrap the pattern in double quotes now, look out for trouble! This is better: "${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 be 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.

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 ${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} * unpausequeuemember()
- ${VMSTATUS} * voicmail()
- ${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_PRIO}` - 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)
- ${SIPFROMDOMAIN} - Set SIP domain 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_CODEC} - Set the SIP codec for an inbound call
- ${SIP_CODEC_INBOUND} - Set the SIP codec for an inbound call
- ${SIP_CODEC_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.
- ${OSPINTOKEN} - The inbound OSP token.
- ${OSPINTIMELIMIT} - 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.
- ${OSPOUTTOKEN} - The outbound OSP token.
- ${OSPOUTTIMELIMIT} - 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.

Distributed Universal Number Discovery (DUNDi)

Top-level page for all things DUNDi

Introduction to DUNDi

http://www.dundi.com

Mark Spencer, Digium, Inc.

DUNDi is essentially a trusted, peer-to-peer system for being able to call any phone number from the Internet. DUNDi works by creating a network of nodes called the "DUNDi E.164 Trust Group" which are bound by a common peering agreement known as the General Peering Agreement or GPA. The GPA legally binds the members of the Trust Group to provide good-faith accurate information to the other nodes on the network, and provides standards by which the community can insure the integrity of the information on the nodes themselves. Unlike ENUM or similar systems, DUNDi is explicitly designed to preclude any necessity for a single centralized system which could be a source of fees, regulation, etc.

Much less dramatically, DUNDi can also be used within a private enterprise to share a dialplan efficiently between multiple nodes, without incurring a risk of a single point of failure. In this way, administrators can locally add extensions which become immediately available to the other nodes in the system.

For more information visit http://www.dundi.com

DUNDIQUERY and DUNDIRESULT

The DUNDIQUERY and DUNDIRESULT dialplan functions will let you initiate a DUNDi query from the dialplan, see how many results there are, and access each one. Here is some example usage:

```
exten => 1,1,Set(ID=${DUNDIQUERY(1,dundi_test,b)})
exten => 1,n,Set(NUM=${DUNDIRESULT(${ID},getnum)})
exten => 1,n,NoOp(There are ${NUM} results)
exten => 1,n,Set(X=1)
exten => 1,n,While($[${X} <= ${NUM}])
exten => 1,n,NoOp(Result ${X} is ${DUNDIRESULT(${ID},${X})})
exten => 1,n,Set(X=$[${X} + 1])
exten => 1,n,EndWhile
```

DUNDi Peering Agreement

```
DIGIUM GENERAL PEERING AGREEMENT (TM)
Version 1.0.0, September 2004
Copyright (C) 2004 Digium, Inc.
150 West Park Loop Suite 100, Huntsville, AL 35806 USA

Everyone is permitted to copy and distribute complete verbatim copies of this General Peering Agreement provided it is not
modified in any manner.

------------------------------------------------------

DIGIUM GENERAL PEERING AGREEMENT

PREAMBLE

For most of the history of telecommunications, the power of being able to locate and communicate with another person in a system,
be it across a hall or around the world, has always centered around a centralized authority -- from a local PBX administrator to
regional and national RBOCs, generally requiring fees, taxes or regulation. By contrast, DUNDi is a technology developed to
provide users the freedom to communicate with each other without the necessity of any centralized authority. This General Peering
Agreement ("GPA") is used by individual parties (each, a "Participant") to allow them to build the E164 trust group for the DUNDi
protocol.

To protect the usefulness of the E164 trust group for those who use it, while keeping the system wholly decentralized, it is
necessary to replace many of the responsibilities generally afforded to a company or government agency, with a set of
responsibilities implemented by the parties who use the system, themselves. It is the goal of this document to provide all the
protections necessary to keep the DUNDi E164 trust group useful and reliable.

The Participants wish to protect competition, promote innovation and value added services and make this service valuable both
commercially and non-commercially. To that end, this GPA provides special terms and conditions outlining some permissible and
non-permissible revenue sources.

This GPA is independent of any software license or other license agreement for a program or technology employing the DUNDi
protocol. For example, the implementation of DUNDi used by Asterisk is covered under a separate license. Each Participant is
responsible for compliance with any licenses or other agreements governing use of such program or technology that they use to
peer.

You do not have to execute this GPA to use a program or technology employing the DUNDi protocol, however if you do not execute
this GPA, you will not be able to peer using DUNDi and the E164 context with anyone who is a member of the trust group by virtue
of their having executed this GPA with another member.

The parties to this GPA agree as follows:

0. DEFINITIONS. As used herein, certain terms shall be defined as
follows:

(a) The term "DUNDi" means the DUNDi protocol as published by Digium, Inc. or its successor in interest with respect to the DUNDi
protocol specification.

(b) The terms "E.164" and "E164" mean ITU-T specification E.164 as published by the International Telecommunications Union (ITU)
in May, 1997.

(c) The term "Service" refers to any communication facility (e.g., telephone, fax, modem, etc.), identified by an
E.164-compatible number, and assigned by the appropriate authority in that jurisdiction.

(d) The term "Egress Gateway" refers an Internet facility that provides a communications path to a Service or Services that may
not be directly addressable via the Internet.

(e) The term "Route" refers to an Internet address, policies, and other characteristics defined by the DUNDi protocol and
associated with the Service, or the Egress Gateway which provides access to the specified Service.

(f) The term "Propagate" means to accept or transmit Service and/or Egress Gateway Routes only using the DUNDi protocol and the
DUNDi context "e164" without regard to case, and does not apply to the exchange of information using any other protocol or
context.

(g) The term "Peering System" means the network of systems that Propagate Routes.

(h) The term "Subscriber" means the owner of, or someone who contracts to receive, the services identified by an E.164 number.

(i) The term "Authorizing Individual" means the Subscriber to a number who has authorized a Participant to provide Routes
regarding their services via this Peering System.

(j) The term "Route Authority" refers to a Participant that provides an original source of said Route within the Peering System.
Routes are propagated from the Route Authorities through the Peering System and may be cached at intermediate points. There may
be multiple Route Authorities for any Service.

(k) The term "Participant" (introduced above) refers to any member of the Peering System.

(l) The term "Service Provider" refers to the carrier (e.g., exchange carrier, Internet Telephony Service Provider, or other
reseller) that provides communication facilities for a particular Service to a Subscriber, Customer or other End User.

(m) The term "Weight" refers to a numeric quality assigned to a Route as per the DUNDi protocol specification. The current Weight
definitions are shown in Exhibit A.

1. PEERING. The undersigned Participants agree to Propagate Routes with each other and any other member of the Peering System and
further agree not to Propagate DUNDi Routes with a third party unless they have first have executed this GPA (in its unmodified
form) with such third party. The Participants further agree only to Propagate Routes with Participants whom they reasonably
```

believe to be honoring the terms of the GPA. Participants may not insert, remove, amend, or otherwise modify any of the terms of the GPA.

2. ACCEPTABLE USE POLICY. The DUNDi protocol contains information that reflect a Subscriber's or Egress Gateway's decisions to receive calls. In addition to the terms and conditions set forth in this GPA, the Participants agree to honor the intent of restrictions encoded in the DUNDi protocol. To that end, Participants agree to the following:

(a) A Participant may not utilize or permit the utilization of Routes for which the Subscriber or Egress Gateway provider has indicated that they do not wish to receive "Unsolicited Calls" for the purpose of making an unsolicited phone call on behalf of any party or organization.

(b) A Participant may not utilize or permit the utilization of Routes which have indicated that they do not wish to receive "Unsolicited Commercial Calls" for the purpose of making an unsolicited phone call on behalf of a commercial organization.

(c) A Participant may never utilize or permit the utilization of any DUNDi route for the purpose of making harassing phone calls.

(d) A Party may not utilize or permit the utilization of DUNDi provided Routes for any systematic or random calling of numbers (e.g., for the purpose of locating facsimile, modem services, or systematic telemarketing).

(e) Initial control signaling for all communication sessions that utilize Routes obtained from the Peering System must be sent from a member of the Peering System to the Service or Egress Gateway identified in the selected Route. For example, 'SIP INVITES' and IAX2 "NEW" commands must be sent from the requesting DUNDi node to the terminating Service.

(f) A Participant may not disclose any specific Route, Service or Participant contact information obtained from the Peering System to any party outside of the Peering System except as a by-product of facilitating communication in accordance with section 2e (e.g., phone books or other databases may not be published, but the Internet addresses of the Egress Gateway or Service does not need to be obfuscated.)

(g) The DUNDi Protocol requires that each Participant include valid contact information about itself (including information about nodes connected to each Participant). Participants may use or disclose the contact information only to ensure enforcement of legal furtherance of this Agreement.

3. ROUTES. The Participants shall only propagate valid Routes, as defined herein, through the Peering System, regardless of the original source. The Participants may only provide Routes as set forth below, and then only if such Participant has no good faith reason to believe such Route to be invalid or unauthorized.

(a) A Participant may provide Routes if each Route has as its original source another member of the Peering System who has duly executed the GPA and such Routes are provided in accordance with this Agreement; provided that the Routes are not modified (e.g., with regards to existence, destination, technology or Weight); or

(b) A Participant may provide Routes for Services with any Weight for which it is the Subscriber; or

(c) A Participant may provide Routes for those Services whose Subscriber has authorized the Participant to do so, provided that the Participant is able to confirm that the Authorizing Individual is the Subscriber through:

i. a written statement of ownership from the Authorizing Individual, which the Participant believes in good faith to be accurate (e.g., a phone bill with the name of the Authorizing Individual and the number in question); or

ii. the Participant's own direct personal knowledge that the Authorizing Individual is the Subscriber.

(d) A Participant may provide Routes for Services, with Weight in accordance with the Current DUNDi Specification, if it can in good faith provide an Egress Gateway to that Service on the traditional telephone network without cost to the calling party.

4. REVOCATION. A Participant must provide a free, easily accessible mechanism by which a Subscriber may revoke permission to act as a Route Authority for his Service. A Participant must stop acting as a Route Authority for that Service within 7 days after:

(a) receipt of a revocation request;

(b) receiving other notice that the Service is no longer valid; or

(c) determination that the Subscriber's information is no longer accurate (including that the Subscriber is no longer the service owner or the service owner's authorized delegate).

5. SERVICE FEES. A Participant may charge a fee to act as a Route Authority for a Service, with any Weight, provided that no Participant may charge a fee to propagate the Route received through the Peering System.

6. TOLL SERVICES. No Participant may provide Routes for any Services that require payment from the calling party or their customer for communication with the Service. Nothing in this section shall prohibit a Participant from providing routes for Services where the calling party may later enter into a financial transaction with the called party (e.g., a Participant may provide Routes for calling cards services).

7. QUALITY. A Participant may not intentionally impair communication using a Route provided to the Peering System (e.g. by adding delay, advertisements, reduced quality). If for any reason a Participant is unable to deliver a call via a Route provided to the Peering System, that Participant shall return out-of-band Network Congestion notification (e.g. "503 Service Unavailable" with SIP protocol or "CONGESTION" with IAX protocol).

8. PROTOCOL COMPLIANCE. Participants agree to Propagate Routes in strict compliance with current DUNDi protocol specifications.

9. ADMINISTRATIVE FEES. A Participant may charge (but is not required to charge) another Participant a reasonable fee to cover administrative expenses incurred in the execution of this Agreement. A Participant may not charge any fee to continue the relationship or to provide Routes to another Participant in the Peering System.

10. CALLER IDENTIFICATION. A Participant will make a good faith effort to ensure the accuracy and appropriate nature of any caller identification that it transmits via any Route obtained from the Peering System. Caller identification shall at least be provided as a valid E.164 number.

11. COMPLIANCE WITH LAWS. The Participants are solely responsible for determining to what extent, if any, the obligations set forth in this GPA conflict with any laws or regulations their region. A Participant may not provide any service or otherwise use DUNDi under this GPA if doing so is prohibited by law or regulation, or if any law or regulation imposes requirements on the

Participant that are inconsistent with the terms of this GPA or the Acceptable Use Policy.

12. WARRANTY. EACH PARTICIPANT WARRANTS TO THE OTHER PARTICIPANTS THAT IT MADE, AND WILL CONTINUE TO MAKE, A GOOD FAITH EFFORT TO AUTHENTICATE OTHERS IN THE PEERING SYSTEM AND TO PROVIDE ACCURATE INFORMATION IN ACCORDANCE WITH THE TERMS OF THIS GPA. THIS WARRANTY IS MADE BETWEEN THE PARTICIPANTS, AND THE PARTICIPANTS MAY NOT EXTEND THIS WARRANTY TO ANY NON-PARTICIPANT INCLUDING END-USERS.

13. DISCLAIMER OF WARRANTIES. THE PARTICIPANTS UNDERSTAND AND AGREE THAT ANY SERVICE PROVIDED AS A RESULT OF THIS GPA IS "AS IS." EXCEPT FOR THOSE WARRANTIES OTHERWISE EXPRESSLY SET FORTH HEREIN, THE PARTICIPANTS DISCLAIM ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND OR NATURE, EXPRESS OR IMPLIED, AS TO THE CONDITION, VALUE OR QUALITIES OF THE SERVICES PROVIDED HEREUNDER, AND SPECIFICALLY DISCLAIM ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY, SUITABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR AS TO THE CONDITION OR WORKMANSHIP THEREOF, OR THE ABSENCE OF ANY DEFECTS THEREIN, WHETHER LATENT OR PATENT, INCLUDING ANY WARRANTIES ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE. EXCEPT AS EXPRESSLY PROVIDED HEREIN, THE PARTICIPANTS EXPRESSLY DISCLAIM ANY REPRESENTATIONS OR WARRANTIES THAT THE PEERING SERVICE WILL BE CONTINUOUS, UNINTERRUPTED OR ERROR-FREE, THAT ANY DATA SHARED OR OTHERWISE MADE AVAILABLE WILL BE ACCURATE OR COMPLETE OR OTHERWISE COMPLETELY SECURE FROM UNAUTHORIZED ACCESS.

14. LIMITATION OF LIABILITIES. NO PARTICIPANT SHALL BE LIABLE TO ANY OTHER PARTICIPANT FOR INCIDENTAL, INDIRECT, CONSEQUENTIAL, SPECIAL, PUNITIVE OR EXEMPLARY DAMAGES OF ANY KIND (INCLUDING LOST REVENUES OR PROFITS, LOSS OF BUSINESS OR LOSS OF DATA) IN ANY WAY RELATED TO THIS GPA, WHETHER IN CONTRACT OR IN TORT, REGARDLESS OF WHETHER SUCH PARTICIPANT WAS ADVISED OF THE POSSIBILITY THEREOF.

15. END-USER AGREEMENTS. The Participants may independently enter into agreements with end-users to provide certain services (e.g., fees to a Subscriber to originate Routes for that Service). To the extent that provision of these services employs the Peering System, the Parties will include in their agreements with their end-users terms and conditions consistent with the terms of this GPA with respect to the exclusion of warranties, limitation of liability and Acceptable Use Policy. In no event may a Participant extend the warranty described in Section 12 in this GPA to any end-users.

16. INDEMNIFICATION. Each Participant agrees to defend, indemnify and hold harmless the other Participant or third-party beneficiaries to this GPA (including their affiliates, successors, assigns, agents and representatives and their respective officers, directors and employees) from and against any and all actions, suits, proceedings, investigations, demands, claims, judgments, liabilities, obligations, liens, losses, damages, expenses (including, without limitation, attorneys' fees) and any other fees arising out of or relating to (i) personal injury or property damage caused by that Participant, its employees, agents, servants, or other representatives; (ii) any act or omission by the Participant, its employees, agents, servants or other representatives, including, but not limited to, unauthorized representations or warranties made by the Participant; or (iii) any breach by the Participant of any of the terms or conditions of this GPA.

17. THIRD PARTY BENEFICIARIES. This GPA is intended to benefit those Participants who have executed the GPA and who are in the Peering System. It is the intent of the Parties to this GPA to give to those Participants who are in the Peering System standing to bring any necessary legal action to enforce the terms of this GPA.

18. TERMINATION. Any Participant may terminate this GPA at any time, with or without cause. A Participant that terminates must immediately cease to Propagate.

19. CHOICE OF LAW. This GPA and the rights and duties of the Parties hereto shall be construed and determined in accordance with the internal laws of the State of New York, United States of America, without regard to its conflict of laws principles and without application of the United Nations Convention on Contracts for the International Sale of Goods.

20. DISPUTE RESOLUTION. Unless otherwise agreed in writing, the exclusive procedure for handling disputes shall be as set forth herein. Notwithstanding such procedures, any Participant may, at any time, seek injunctive relief in addition to the process described below.

(a) Prior to mediation or arbitration the disputing Participants shall seek informal resolution of disputes. The process shall be initiated with written notice of one Participant to the other describing the dispute with reasonable particularity followed with a written response within ten (10) days of receipt of notice. Each Participant shall promptly designate an executive with requisite authority to resolve the dispute. The informal procedure shall commence within ten (10) days of the date of response. All reasonable requests for non-privileged information reasonably related to the dispute shall be honored. If the dispute is not resolved within thirty (30) days of commencement of the procedure either Participant may proceed to mediation or arbitration pursuant to the rules set forth in (b) or (c) below.

(b) If the dispute has not been resolved pursuant to (a) above or, if the disputing Participants fail to commence informal dispute resolution pursuant to (a) above, either Participant may, in writing and within twenty (20) days of the response date noted in (a) above, ask the other Participant to participate in a one (1) day mediation with an impartial mediator, and the other Participant shall do so. Each Participant will bear its own expenses and an equal share of the fees of the mediator. If the mediation is not successful the Participants may proceed with arbitration pursuant to (c) below.

(c) If the dispute has not been resolved pursuant to (a) or (b) above, the dispute shall be promptly referred, no later than one (1) year from the date of original notice and subject to applicable statute of limitations, to binding arbitration in accordance with the UNCITRAL Arbitration Rules in effect on the date of this contract. The appointing authority shall be the International Centre for Dispute Resolution. The case shall be administered by the International Centre for Dispute Resolution under its Procedures for Cases under the UNCITRAL Arbitration Rules. Each Participant shall bear its own expenses and shall share equally in fees of the arbitrator. All arbitrators shall have substantial experience in information technology and/or in the telecommunications business and shall be selected by the disputing participants in accordance with UNCITRAL Arbitration Rules. If any arbitrator, once selected is unable or unwilling to continue for any reason, replacement shall be filled via the process described above and a re-hearing shall be conducted. The disputing Participants will provide each other with all requested documents and records reasonably related to the dispute in a manner that will minimize the expense and inconvenience of both parties. Discovery will not include depositions or interrogatories except as the arbitrators expressly allow upon a showing of need. If disputes arise concerning discovery requests, the arbitrators shall have sole and complete discretion to resolve the disputes. The parties and arbitrator shall be guided in resolving discovery disputes by the Federal Rules of Civil Procedure. The Participants agree that time of the essence principles shall guide the hearing and that the arbitrator shall have the right and authority to issue monetary sanctions in the event of unreasonable delay. The arbitrator shall deliver a written opinion setting forth findings of fact and the rationale for the award within thirty (30) days following conclusion of the hearing. The award of the arbitrator, which may include legal and equitable relief, but which may not include punitive damages, will be final and binding upon the disputing Participants, and judgment may be entered upon it in accordance with applicable law in any court having jurisdiction thereof. In addition to award the arbitrator shall have the discretion to award the prevailing Participant all or part of its attorneys' fees and costs, including fees associated with arbitrator, if the arbitrator determines that the positions taken by the other Participant on material issues of the dispute were without substantial foundation. Any conflict between the UNCITRAL Arbitration Rules and the provisions of this GPA shall be controlled by this GPA.

21. INTEGRATED AGREEMENT. This GPA, constitutes the complete integrated agreement between the parties concerning the subject

matter hereof. All prior and contemporaneous agreements, understandings, negotiations or representations, whether oral or in writing, relating to the subject matter of this GPA are superseded and canceled in their entirety.

22. WAIVER. No waiver of any of the provisions of this GPA shall be deemed or shall constitute a waiver of any other provision of this GPA, whether or not similar, nor shall such waiver constitute a continuing waiver unless otherwise expressly so provided in writing. The failure of either party to enforce at any time any of the provisions of this GPA, or the failure to require at any time performance by either party of any of the provisions of this GPA, shall in no way be construed to be a present or future waiver of such provisions, nor in any way affect the ability of a Participant to enforce each and every such provision thereafter.

23. INDEPENDENT CONTRACTORS. Nothing in this GPA shall make the Parties partners, joint venturers, or otherwise associated in or with the business of the other. Parties are, and shall always remain, independent contractors. No Participant shall be liable for any debts, accounts, obligations, or other liabilities of the other Participant, its agents or employees. No party is authorized to incur debts or other obligations of any kind on the part of or as agent for the other. This GPA is not a franchise agreement and does not create a franchise relationship between the parties, and if any provision of this GPA is deemed to create a franchise between the parties, then this GPA shall automatically terminate.

24. CAPTIONS AND HEADINGS. The captions and headings used in this GPA are used for convenience only and are not to be given any legal effect.

25. EXECUTION. This GPA may be executed in counterparts, each of which so executed will be deemed to be an original and such counterparts together will constitute one and the same Agreement. The Parties shall transmit to each other a signed copy of the GPA by any means that faithfully reproduces the GPA along with the Signature. For purposes of this GPA, the term "signature" shall include digital signatures as defined by the jurisdiction of the Participant signing the GPA.

Exhibit A

Weight Range Requirements

0-99 May only be used under authorization of Owner

100-199 May only be used by the Owner's service
provider, regardless of authorization.

200-299 Reserved -- do not use for e164 context.

300-399 May only be used by the owner of the code under
which the Owner's number is a part of.

400-499 May be used by any entity providing access via
direct connectivity to the Public Switched
Telephone Network.

500-599 May be used by any entity providing access via
indirect connectivity to the Public Switched
Telephone Network (e.g. Via another VoIP
provider)

600- Reserved-- do not use for e164 context.

Participant Participant

Company:

Address:

Email:

_________________________ _________________________
Authorized Signature Authorized Signature

Name:

END OF GENERAL PEERING AGREEMENT

How to Peer using this GPA If you wish to exchange routing information with parties using the e164 DUNDi context, all you must do is execute this GPA with any member of the Peering System and you will become a member of the Peering System and be able to make Routes available in accordance with this GPA.

```
DUNDi, IAX, Asterisk and GPA are trademarks of Digium, Inc.
```

E.164 NUmber Mapping (ENUM)

The ENUMLOOKUP Dialplan Function

The ENUMLOOKUP function is more complex than it first may appear, and this guide is to give a general overview and set of examples that may be well-suited for the advanced user to evaluate in their consideration of ENUM or ENUM-like lookup strategies. This document assumes a familiarity with ENUM (RFC3761) or ENUM-like methods, as well as familiarity with NAPTR DNS records (RFC2915, RFC3401-3404). For an overview of NAPTR records, and the use of NAPTRs in the ENUM global phone-number-to-DNS mapping scheme, please see http://www.voip-info.org/tiki-index.php?page=ENUM for more detail.

Using ENUM within Asterisk can be simple or complex, depending on how many failover methods and redundancy procedures you wish to utilize. Implementation of ENUM paths is supposedly defined by the person creating the NAPTR records, but the local administrator may choose to ignore certain NAPTR response methods (URI types) or prefer some over others, which is in contradiction to the RFC. The ENUMLOOKUP method simply provides administrators a method for determining NAPTR results in either the globally unique ENUM (e164.arpa) DNS tree, or in other ENUM-like DNS trees which are not globally unique. The methods to actually create channels ("dial") results given by the ENUMLOOKUP function is then up to the administrator to implement in a way that best suits their environment.

```
Function: ENUMLOOKUP(number[,Method-type[,options[,record#[,zone-suffix]]]])
```

Performs an ENUM tree lookup on the specified number, method type, and ordinal record offset, and returns one of four different values:

1. Post-parsed NAPTR of one method (URI) type
2. Count of elements of one method (URI) type
3. Count of all method types
4. Full URI of method at a particular point in the list of all possible methods

ENUMLOOKUP Arguments

- number - Telephone number or search string. Only numeric values within this string are parsed; all other digits are ignored for search, but are re-written during NAPTR regexp expansion.
- service_type - tel, sip, h323, iax2, mailto, ...[any other string], ALL. Default type is "sip". Special name of "ALL" will create a list of method types across all NAPTR records for the search number, and then put the results in an ordinal list starting with 1. The position number specified will then be returned, starting with 1 as the first record (lowest value) in the list. The service types are not hardcoded in Asterisk except for the default (sip) if no other service type specified; any method type string (IANA-approved or not) may be used except for the string "ALL".
- options
    - c - count. Returns the number of records of this type are returned (regardless of order or priority.) If "ALL" is the specified service_type, then a count of all methods will be returned for the DNS record.
- record# - Which record to present if multiple answers are returned integer = The record in priority/order sequence based on the total count of records passed back by the query. If a service_type is specified, all entries of that type will be sorted into an ordinal list starting with 1 (by order first, then priority). The default of options is "1"
- zone_suffix - Allows customization of the ENUM zone. Default is e164.arpa.

ENUMLOOKUP Examples

Let's use this ENUM list as an example (note that these examples exist in the DNS, and will hopefully remain in place as example destinations, but they may change or become invalid over time. The end result URIs are not guaranteed to actually work, since some of these hostnames or SIP proxies are imaginary. Of course, the tel: replies go to directory assistance for New York City and San Francisco...) Also note that the complex SIP NAPTR at weight 30 will strip off the leading "+" from the dialed string if it exists. This is probably a better NAPTR than hard-coding the number into the NAPTR, and it is included as a more complex regexp example, though other simpler NAPTRs will work just as well.

```
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 10 100 "u" "E2U+tel" "Unable to render embedded object: File (+12125551212) not
found." .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 21 100 "u" "E2U+tel" "Unable to render embedded object: File (+14155551212) not
found." .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 25 100 "u" "E2U+sip" "Unable to render embedded object: File
(2203@sip.fox-den.com) not found." .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 26 100 "u" "E2U+sip" "Unable to render embedded object: File
(1234@sip-2.fox-den.com) not found." .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 30 100 "u" "E2U+sip" "Unable to render embedded object: File
(\\1@sip-3.fox-den.com) not found." .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 55 100 "u" "E2U+mailto" "Unable to render embedded object: File
(jtodd@fox-den.com) not found." .
```

Example 1: Simplest case, using first SIP return (use all defaults except for domain name)

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,,,,loligo.com)})
```

returns: ${foo}="2203@sip.fox-den.com"

Example 2: What is the first "tel" pointer type for this number? (after sorting by order/preference; default of "1" is assumed in options field)

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,tel,,,loligo.com)})
```

returns: ${foo}="+12125551212"

Example 3: How many "sip" pointer type entries are there for this number?

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,sip,c,,loligo.com)})
```

returns: ${foo}=3

Example 4: For all the "tel" pointer type entries, what is the second one in the list? (after sorting by preference)

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,tel,,2,loligo.com)})
```

returns: ${foo}="+14155551212"

Example 5: How many NAPTRs (tel, sip, mailto, etc.) are in the list for this number?

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,ALL,c,,loligo.com)})
```

returns: ${foo}=6

Example 6: Give back the second full URI in the sorted list of all NAPTR URIs:

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,ALL,,2,loligo.com)})
```

returns: ${foo}="tel:+14155551212" [note the "tel:" prefix in the string]

Example 7: Look up first SIP entry for the number in the e164.arpa zone (all defaults)

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+437203001721)})
```

returns: ${foo}="enum-test@sip.nemox.net" [note: this result is subject to change as it is "live" DNS and not under my control]

Example 8: Look up the ISN mapping in freenum.org alpha test zone

```
exten => 100,1,Set(foo=${ENUMLOOKUP(1234*256,,,,freenum.org)})
```

returns: ${foo}="1234@204.91.156.10" [note: this result is subject to change as it is "live" DNS]

Example 9: Give back the first SIP pointer for a number in the enum.yoydynelabs.com zone (invalid lookup)

```
exten => 100,1,Set(foo=${ENUMLOOKUP(1234567890,sip,,1,enum.yoyodynelabs.com)})
```

returns: ${foo}=""

ENUMLOOKUP Usage Notes and Subtle Features

- The use of "" in lookups is confusing, and warrants further explanation. All E.164 numbers ("global phone numbers") by definition need a leading "" during ENUM lookup. If you neglect to add a leading "", you may discover that numbers that seem to exist in the DNS aren't getting matched by the system or are returned with a null string result. This is due to the NAPTR reply requiring a "" in the regular expression matching sequence. Older versions of Asterisk add a "" from within the code, which may confuse administrators converting to the new function. Please ensure that all ENUM (e164.arpa) lookups contain a leading "" before lookup, so ensure your lookup includes the leading plus sign. Other DNS trees may or may not require a leading "" - check before using those trees, as it is possible the parsed NAPTRs will not provide correct results unless you have the correct dialed string. If you get console messages like "WARNING[24907]: enum.c:222 parse_naptr: NAPTR Regex match failed." then it is very possible that the returned NAPTR expects a leading "" in the search string (or the returned NAPTR is mis-formed.)
- If a query is performed of type "c" ("count") and let's say you get back 5 records and then some seconds later a query is made against record 5 in the list, it may not be the case that the DNS resolver has the same answers as it did a second or two ago - maybe there are only 4 records in the list in the newest query. The resolver should be the canonical storage location for DNS records, since that is the intent of ENUM. However, some obscure future cases may have wildly changing NAPTR records within several seconds. This is a corner case, and probably only worth noting as a very rare circumstance. (note: I do not object to Asterisk's dnsmgr method of locally caching DNS replies, but this method needs to honor the TTL given by the remote zone master. Currently, the ENUMLOOKUP function does not use the dnsmgr method of caching local DNS replies.)
- If you want strict NAPTR value ordering, then it will be necessary to use the "ALL" method to incrementally step through the different returned NAPTR pointers. You will need to use string manipulation to strip off the returned method types, since the results will look like "sip:12125551212" in the returned value. This is a non-trivial task, though it is required in order to have strict RFC compliance and to comply with the desires of the remote party who is presenting NAPTRs in a particular order for a reason.
- Default behavior for the function (even in event of an error) is to move to the next priority, and the result is a null value. Most ENUM lookups are going to be failures, and it is the responsibility of the dialplan administrator to manage error conditions within their dialplan. This is a change from the old app_enumlookup method and it's arbitrary priority jumping based on result type or failure.
- Anything other than digits will be ignored in lookup strings. Example: a search string of "+4372030blah01721" will turn into 1.2.7.1.0.0.3.0.2.7.3.4.e164.arpa. for the lookup. The NAPTR parsing may cause unexpected results if there are strings inside your NAPTR lookups.
  If there exist multiple records with the same weight and order as a result of your query, the function will RANDOMLY select a single NAPTR from those equal results.
- Currently, the function ignores the settings in enum.conf as the search zone name is now specified within the function, and the H323 driver can be chosen by the user via the dialplan. There were no other values in this file, and so it becomes deprecated.
- The function will digest and return NAPTRs which use older (deprecated) style, reversed method strings such as "sip+E2U" instead of the more modern "E2U+sip"
- There is no provision for multi-part methods at this time. If there are multiple NAPTRs with (as an example) a method of "E2U+voice:sip" and then another NAPTR in the same DNS record with a method of ""E2U+sip", the system will treat these both as method "sip" and they will be separate records from the perspective of the function. Of course, if both records point to the same URI and have equal priority/weight (as is often the case) then this will cause no serious difficulty, but it bears mentioning.
- ISN (ITAD Subscriber Number) usage: If the search number is of the form ABC*DEF (where ABC and DEF are at least one numeric digit) then perform an ISN-style lookup where the lookup is manipulated to C.B.A.DEF.domain.tld (all other settings and options apply.) See http://www.freenum.org/ for more details on ISN lookups. In the unlikely event you wish to avoid ISN re-writes, put an "n" as the first digit of the search string - the "n" will be ignored for the search.

More ENUMLOOKUP Examples

All examples below except where noted use "e164.arpa" as the referenced domain, which is the default domain name for ENUMLOOKUP. All numbers are assumed to not have a leading "+" as dialed by the inbound channel, so that character is added where necessary during ENUMLOOKUP function calls.

extensions.conf

```
; example 1
;
; Assumes North American international dialing (011) prefix.
; Look up the first SIP result and send the call there, otherwise
; send the call out a PRI. This is the most simple possible
; ENUM example, but only uses the first SIP reply in the list of
; NAPTR(s).
;
exten => _011.,1,Set(enumresult=${ENUMLOOKUP(${EXTEN:3})})
exten => _011.,n,Dial(SIP/${enumresult})
exten => _011.,n,Dial(DAHDI/g1/${EXTEN})

;
; example 2
;
; Assumes North American international dialing (011) prefix.
; Check to see if there are multiple SIP NAPTRs returned by
; the lookup, and dial each in order. If none work (or none
; exist) then send the call out a PRI, group 1.
;
exten => _011.,1,Set(sipcount=${ENUMLOOKUP(${EXTEN:3},sip,c)}|counter=0)
exten => _011.,n,While($["${counter}"<"${sipcount}"])
exten => _011.,n,Set(counter=$[${counter}+1])
exten => _011.,n,Dial(SIP/${ENUMLOOKUP(${EXTEN:3},sip,,${counter})})
exten => _011.,n,EndWhile
exten => _011.,n,Dial(DAHDI/g1/${EXTEN})

;
; example 3
;
; This example expects an ${EXTEN} that is an e.164 number (like
; 14102241145 or 437203001721)
; Search through e164.arpa and then also search through e164.org
; to see if there are any valid SIP or IAX termination capabilities.
; If none, send call out via DAHDI channel 1.
;
; Start first with e164.arpa zone...
;
exten => _X.,1,Set(sipcount=${ENUMLOOKUP(${EXTEN},sip,c)}|counter=0)
exten => _X.,2,GotoIf($["${counter}"<"${sipcount}"]?3:6)
exten => _X.,3,Set(counter=$[${counter}+1])
exten => _X.,4,Dial(SIP/${ENUMLOOKUP(${EXTEN},sip,,${counter})})
exten => _X.,5,GotoIf($["${counter}"<"${sipcount}"]?3:6)
;
exten => _X.,6,Set(iaxcount=${ENUMLOOKUP(${EXTEN},iax2,c)}|counter=0)
exten => _X.,7,GotoIf($["${counter}"<"${iaxcount}"]?8:11)
exten => _X.,8,Set(counter=$[${counter}+1])
exten => _X.,9,Dial(IAX2/${ENUMLOOKUP(${EXTEN},iax2,,${counter})})
exten => _X.,10,GotoIf($["${counter}"<"${iaxcount}"]?8:11)
;
exten => _X.,11,NoOp("No valid entries in e164.arpa for ${EXTEN} - checking in e164.org")
```

```
;
; ...then also try e164.org, and look for SIP and IAX NAPTRs...
;
exten => _X.,12,Set(sipcount=${ENUMLOOKUP(${EXTEN},sip,c,,e164.org)}|counter=0)
exten => _X.,13,GotoIf($["${counter}"<"${sipcount}"]?14:17)
exten => _X.,14,Set(counter=$[${counter}+1])
exten => _X.,15,Dial(SIP/${ENUMLOOKUP(${EXTEN},sip,,${counter},e164.org)})
exten => _X.,16,GotoIf($["${counter}"<"${sipcount}"]?14:17)
;
exten => _X.,17,Set(iaxcount=${ENUMLOOKUP(${EXTEN},iax2,c,,e164.org)}|counter=0)
exten => _X.,18,GotoIf($["${counter}"<"${iaxcount}"]?19:22)
exten => _X.,19,Set(counter=$[${counter}+1])
exten => _X.,20,Dial(IAX2/${ENUMLOOKUP(${EXTEN},iax2,,${counter},e164.org)})
exten => _X.,21,GotoIf($["${counter}"<"${iaxcount}"]?19:22)
;
; ...then send out PRI.
;
```

```
exten => _X.,22,NoOp("No valid entries in e164.org for ${EXTEN} - sending out via DAHDI")
exten => _X.,23,Dial(DAHDI/g1/${EXTEN})
```

Features

Miscellaneous documents that talk about Asterisk functionality. All of this needs to be integrated into Configuration and Operation.

Asterisk Applications

This page is a container page for Asterisk applications, e.g. those things that appear in the apps source directory.

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(RESULT=${COUNT})
exten => s,n,SetGlobalVar(COUNT=$[${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.

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 an 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.<queuename> holding all messages from service centre to phone, and me-sc.<queuename> 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-<queuename> directory, deleting the files as it goes. Any received in this mode are placed in the me-sc-<queuename> directory.

When connected as a client, SMS() will send all messages waiting in the me-sc-<queuename> directory, deleting the files as it goes. Any received in this mode are placed in the sc-me-<queuename> 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 processed 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...

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 Command Line Interface

In addition to being the console for Asterisk, the CLI also sports several features that make it very helpful to use for obtaining information and affecting system configuration. The console can also be seen by starting a remote console, which connects to the running daemon and shows much of the same information as if using the daemon in foreground mode.

Connecting a remote console is as easy as using the -r or -R flags. The only difference between these flags is that the uppercase variation (-R) will automatically reconnect to the daemon (or at least retry) if the daemon restarts. To exit a remote console, simply type 'quit' or 'exit'. Please note that you can differentiate between a remote console and the 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, you can use the 'stop' set of commands, such as 'stop now', 'stop gracefully', or 'stop when convenient'.

Once on the console, the 'help' command may be used to see a list of commands available for use. Note that in addition to the 'help' command, the Asterisk CLI sports tab command line completion on all commands, including many arguments. To use tab command line completion, 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.

The 'help' command may also be used to obtain more detailed information on how to use a particular command. 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.

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: -> DAHDIChannel
 For active channels, the Channel: and Uniqueid: headers are added
 You can now add a "DAHDIChannel: " 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
  SpyeeChannel: 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

...

Building Queues

Building Queues

Written by: Leif Madsen
Initial version: 2010-01-14

In this article, we'll look at setting up a pair of queues in Asterisk called 'sales' and 'support'. These queues can be logged into by queue members, and those members will also have the ability to pause and unpause themselves.

All configuration will be done in flat files on the system in order to maintain simplicity in configuration.

Note that this documentation is based on Asterisk 1.6.2, and this is just one approach to creating queues and the dialplan logic. You may create a better way, and in that case, I would encourage you to submit it to the Asterisk issue tracker at http://issues.asterisk.org for inclusion in Asterisk.

Adding SIP Devices to Your Server

The first thing we want to do is register a couple of SIP devices to our server. These devices will be our agents that can login and out of the queues we'll create later. Our naming convention will be to use MAC addresses as we want to abstract the concepts of user (agent), device, and extension from each other.

In sip.conf, we add the following to the bottom of our file:

```
sip.conf
--------

[std-device](!)
type=peer
context=devices
host=dynamic
secret=s3CuR#p@s5
dtmfmode=rfc2833
disallow=all
allow=ulaw

[0004f2040001](std-device)

[0004f2040002](std-device)
```

What we're doing here is creating a [std-device] template and applying it to a pair of peers that we'll register as 0004f2040001 and 0004f2040002; our devices.

Then our devices can register to Asterisk. In my case I have a hard phone and a soft phone registered. I can verify their connectivity by running 'sip show peers'.

```
*CLI> sip show peers
Name/username              Host            Dyn Nat ACL Port     Status
0004f2040001/0004f2040001  192.168.128.145  D          5060     Unmonitored
0004f2040002/0004f2040002  192.168.128.126  D          5060     Unmonitored
2 sip peers [Monitored: 0 online, 0 offline Unmonitored: 2 online, 0 offline]
```

Configuring Device State

Next, we need to configure our system to track the state of the devices. We do this by defining a 'hint' in the dialplan which creates the ability for a device subscription to be retained in memory. By default we can see there are no hints registered in our system by running the 'core show hints' command.

```
*CLI> core show hints
There are no registered dialplan hint
```

We need to add the devices we're going to track to the extensions.conf file under the [default] context which is the default configuration in sip.conf, however we can change this to any context we want with the 'subscribecontext'
option.

Add the following lines to extensions.conf:

```
[default]
exten => 0004f2040001,hint,SIP/0004f2040001
exten => 0004f2040002,hint,SIP/0004f2040002
```

Then perform a 'dialplan reload' in order to reload the dialplan.

After reloading our dialplan, you can see the status of the devices with 'core show hints' again.

```
*CLI> core show hints

    -= Registered Asterisk Dial Plan Hints =-
           0004f2040002@default             : SIP/0004f2040002      State:Idle
Watchers  0
           0004f2040001@default             : SIP/0004f2040001      State:Idle
Watchers  0
----------------
- 2 hints registered
```

At this point, create an extension that you can dial that will play a prompt that is long enough for you to go back to the Asterisk console to check the state of your device while it is in use.

To do this, add the 555 extension to the [devices] context and make it playback the tt-monkeys file.

```
extensions.conf
---------------

[devices]
exten => 555,1,Playback(tt-monkeys)
```

Dial that extension and then check the state of your device on the console.

```
*CLI>   == Using SIP RTP CoS mark 5
    -- Executing [555@devices:1] Playback("SIP/0004f2040001-00000001", "tt-monkeys") in
new stack
    -- <SIP/0004f2040001-00000001> Playing 'tt-monkeys.slin' (language 'en')

*CLI> core show hints

    -= Registered Asterisk Dial Plan Hints =-
           0004f2040002@default             : SIP/0004f2040002      State:Idle
Watchers  0
           0004f2040001@default             : SIP/0004f2040001      State:Idle
Watchers  0
----------------
- 2 hints registered
```

Aha, we're not getting the device state correctly. There must be something else we need to configure.

In sip.conf, we need to enable 'callcounter' in order to activate the ability for Asterisk to monitor whether the device is in use or not. In versions prior to 1.6.0 we needed to use 'call-limit' for this functionality, but call-limit is now deprecated and is no longer necessary.

So, in sip.conf, in our [std-device] template, we need to add the callcounter option.

```
sip.conf
--------

[std-device](!)
type=peer
context=devices
host=dynamic
secret=s3CuR#p@s5
dtmfmode=rfc2833
disallow=all
allow=ulaw
callcounter=yes     ; <-- add this
```

Then reload chan_sip with 'sip reload' and perform our 555 test again. Dial 555 and then check the device state with 'core show hints'.

```
*CLI>   == Using SIP RTP CoS mark 5
    -- Executing [555@devices:1] Playback("SIP/0004f2040001-00000002", "tt-monkeys") in
new stack
    -- <SIP/0004f2040001-00000002> Playing 'tt-monkeys.slin' (language 'en')

*CLI> core show hints

    -= Registered Asterisk Dial Plan Hints =-
           0004f2040002@default             : SIP/0004f2040002      State:Idle
Watchers  0
           0004f2040001@default             : SIP/0004f2040001      State:InUse
Watchers  0
----------------
- 2 hints registered
```

Note that now we have the correct device state when extension 555 is dialed, showing that our device is InUse after dialing extension 555. This is important when creating queues, otherwise our queue members would get multiple calls from the queues.

Adding Queues to Asterisk

The next step is to add a couple of queues to Asterisk that we can assign queue members into. For now we'll work with two queues; sales and support. Lets create those queues now in queues.conf.

We'll leave the default settings that are shipped with queues.conf.sample in the [general] section of queues.conf. See the queues.conf.sample file for more information about each of the available options.

```
queues.conf
--------

[general]
persistantmembers=yes
autofill=yes
monitor-type=MixMonitor
shared_lastcall=no
```

We can then define a [queue_template] that we'll assign to each of the queues we create. These definitions can be overridden by each queue individually if you reassign them under the [sales] or [support] headers. So under the [general]
section of your queues.conf file, add the following.

```
queues.conf
----------

[queue_template](!)
musicclass=default      ; play [default] music
strategy=rrmemory       ; use the Round Robin Memory strategy
joinempty=yes           ; join the queue when no members available
leavewhenempty=no       ; don't leave the queue no members available
ringinuse=no            ; don't ring members when already InUse

[sales](queue_template)
; Sales queue

[support](queue_template)
; Support queue
```

After defining our queues, lets reload our app_queue.so module.

```
*CLI> module reload app_queue.so
    -- Reloading module 'app_queue.so' (True Call Queueing)

  == Parsing '/etc/asterisk/queues.conf':   == Found
```

Then verify our queues loaded with 'queue show'.

```
*CLI> queue show
support      has 0 calls (max unlimited) in 'rrmemory' strategy (0s holdtime, 0s
talktime), W:0, C:0, A:0, SL:0.0% within 0s
   No Members
   No Callers

sales        has 0 calls (max unlimited) in 'rrmemory' strategy (0s holdtime, 0s
talktime), W:0, C:0, A:0, SL:0.0% within 0s
   No Members
   No Callers
```

Adding Queue Members

You'll notice that we have no queue members available to take calls from the queues. We can add queue members from the Asterisk CLI with the 'queue add member' command.

This is the format of the 'queue add member' command:

```
Usage: queue add member <channel> to <queue> [[[penalty <penalty>] as <membername>]
state_interface <interface>]
       Add a channel to a queue with optionally:  a penalty, membername and a
state_interface
```

The penalty, membername, and state_interface are all optional values. Special attention should be brought to the 'state_interface' option for a member though. The reason for state_interface is that if you're using a channel that does not have device state itself (for example, if you were using the Local channel to deliver a call to an end point) then you could assign the device state of a SIP device to the pseudo channel. This allows the state of a SIP device to be applied to the Local channel for correct device state information.

Lets add our device located at SIP/0004f2040001

```
*CLI> queue add member SIP/0004f2040001 to sales
Added interface 'SIP/0004f2040001' to queue 'sales'
```

Then lets verify our member was indeed added.

```
*CLI> queue show sales
sales        has 0 calls (max unlimited) in 'rrmemory' strategy (0s holdtime, 0s
talktime), W:0, C:0, A:0, SL:0.0% within 0s
   Members:
      SIP/0004f2040001 (dynamic) (Not in use) has taken no calls yet
   No Callers
```

Now, if we dial our 555 extension, we should see that our member becomes InUse within the queue.

```
*CLI>   == Using SIP RTP CoS mark 5
    -- Executing [555@devices:1] Playback("SIP/0004f2040001-00000001", "tt-monkeys") in
new stack
    -- <SIP/0004f2040001-00000001> Playing 'tt-monkeys.slin' (language 'en')


*CLI> queue show sales
sales        has 0 calls (max unlimited) in 'rrmemory' strategy (0s holdtime, 0s
talktime), W:0, C:0, A:0, SL:0.0% within 0s
   Members:
      SIP/0004f2040001 (dynamic) (In use) has taken no calls yet
   No Callers
```

We can also remove our members from the queue using the 'queue remove' CLI command.

```
*CLI> queue remove member SIP/0004f2040001 from sales
Removed interface 'SIP/0004f2040001' from queue 'sales'
```

Because we don't want to have to add queue members manually from the CLI, we should create a method that allows queue members to login and out from their devices. We'll do that in the next section.

But first, lets add an extension to our dialplan in order to permit people to dial into our queues so calls can be delivered to our queue members.

```
extensions.conf
---------------

[devices]
exten => 555,1,Playback(tt-monkeys)

exten => 100,1,Queue(sales)

exten => 101,1,Queue(support)
```

Then reload the dialplan, and try calling extension 100 from SIP/0004f2040002, which is the device we have not logged into the queue.

```
*CLI> dialplan reload
```

And now we call the queue at extension 100 which will ring our device at SIP/0004f2040001.

```
*CLI>   == Using SIP RTP CoS mark 5
    -- Executing [100@devices:1] Queue("SIP/0004f2040002-00000005", "sales") in new stack
    -- Started music on hold, class 'default', on SIP/0004f2040002-00000005
  == Using SIP RTP CoS mark 5
    -- SIP/0004f2040001-00000006 is ringing
```

We can see the device state has changed to Ringing while the device is ringing.

```
*CLI> queue show sales
sales        has 1 calls (max unlimited) in 'rrmemory' strategy (2s holdtime, 3s
talktime), W:0, C:1, A:1, SL:0.0% within 0s
   Members:
      SIP/0004f2040001 (dynamic) (Ringing) has taken 1 calls (last was 14 secs ago)
   Callers:
      1. SIP/0004f2040002-00000005 (wait: 0:03, prio: 0)
```

Our queue member then answers the phone.

```
*CLI>     -- SIP/0004f2040001-00000006 answered SIP/0004f2040002-00000005
    -- Stopped music on hold on SIP/0004f2040002-00000005
    -- Native bridging SIP/0004f2040002-00000005 and SIP/0004f2040001-00000006
```

And we can see the queue member is now in use.

```
*CLI> queue show sales
sales        has 0 calls (max unlimited) in 'rrmemory' strategy (3s holdtime, 3s
talktime), W:0, C:1, A:1, SL:0.0% within 0s
   Members:
      SIP/0004f2040001 (dynamic) (In use) has taken 1 calls (last was 22 secs ago)
   No Callers
```

Then the call is hung up.

```
*CLI>   == Spawn extension (devices, 100, 1) exited non-zero on
'SIP/0004f2040002-00000005'
```

And we see that our queue member is available to take another call.

```
*CLI> queue show sales
sales        has 0 calls (max unlimited) in 'rrmemory' strategy (3s holdtime, 4s
talktime), W:0, C:2, A:1, SL:0.0% within 0s
   Members:
      SIP/0004f2040001 (dynamic) (Not in use) has taken 2 calls (last was 6 secs ago)
   No Callers
```

Logging In and Out of Queues

In this section we'll show how to use the AddQueueMember() and RemoveQueueMember() dialplan applications to login and out of queues. For more information about the available options to AddQueueMember() and RemoveQueueMember() use the 'core show application <app>' command from the CLI.

The following bit of dialplan is a bit long, but stick with it, and you'll see that it isn't really all that bad. The gist of the dialplan is that it will check to see if the active user (the device that is dialing the extension) is currently logged into the queue extension that has been requested, and if logged in, then will log them out; if not logged in, then they will be logged into the queue.

We've updated the two lines we added in the previous section that allowed us to dial the sales and support queues. We've abstracted this out a bit in order to make it easier to add new queues in the future. This is done by adding the queue
names to a global variable, then utilizing the extension number dialed to look up the queue name.

So we replace extension 100 and 101 with the following dialplan.

```
; Call any of the queues we've defined in the [globals] section.
exten => _1XX,1,Verbose(2,Call queue as configured in the QUEUE_${EXTEN} global variable)
exten => _1XX,n,Set(thisQueue=${GLOBAL(QUEUE_${EXTEN})})
exten => _1XX,n,GotoIf($["${thisQueue}" = ""]?invalid_queue,1)
exten => _1XX,n,Verbose(2, --> Entering the ${thisQueue} queue)
exten => _1XX,n,Queue(${thisQueue})
exten => _1XX,n,Hangup()

exten => invalid_queue,1,Verbose(2,Attempted to enter invalid queue)
exten => invalid_queue,n,Playback(silence/1&invalid)
exten => invalid_queue,n,Hangup()
```

The globals section contains the following two global variables.

```
[globals]
QUEUE_100=sales
QUEUE_101=support
```

So when we dial extension 100, it matches our pattern _1XX. The number we dialed (100) is then retrievable via ${EXTEN} and we can get the name of queue 100 (sales) from the global variable QUEUE_100. We then assign it to the channel variable thisQueue so it is easier to work with in our dialplan.

```
exten => _1XX,n,Set(thisQueue=${GLOBAL(QUEUE_${EXTEN})})
```

We then check to see if we've gotten a value back from the global variable which would indicate whether the queue was valid or not.

```
exten => _1XX,n,GotoIf($["${thisQueue}" = ""]?invalid_queue,1)
```

If ${thisQueue} returns nothing, then we Goto the invalid_queue extension and playback the 'invalid' file.

We could alternatively limit our pattern match to only extension 100 and 101 with the _10[0-1] pattern instead.

Lets move into the nitty-gritty section and show how we can login and logout our devices to the pair of queues we've created.

First, we create a pattern match that takes star ⭐ plus the queue number that we want to login or logout of. So to login/out of the sales queue (100) we would dial *100. We use the same extension for logging in and out.

```
; Extension *100 or *101 will login/logout a queue member from sales or support queues
respectively.
exten => _*10[0-1],1,Set(xtn=${EXTEN:1})                        ; save ${EXTEN} with *
chopped off to ${xtn}
exten => _*10[0-1],n,Goto(queueLoginLogout,member_check,1)      ; check if already logged
into a queue
```

We save the value of ${EXTEN:1} to the 'xtn' channel variable so we don't need to keep typing the complicated pattern match.

Now we move into the meat of our login/out dialplan inside the [queueLoginLogout] context.

The first section is initializing some variables that we need throughout the member_check extension such as the name of the queue, the members currently logged into the queue, and the current device peer name (i.e. SIP/0004f2040001).

```
; ### Login or Logout a Queue Member
[queueLoginLogout]
exten => member_check,1,Verbose(2,Logging queue member in or out of the request queue)
exten => member_check,n,Set(thisQueue=${GLOBAL(QUEUE_${xtn})})                  ; assign
queue name to a variable
exten => member_check,n,Set(queueMembers=${QUEUE_MEMBER_LIST(${thisQueue})})    ; assign
list of logged in members of thisQueue to
                                                                                ; a
variable (comma separated)
exten => member_check,n,Set(thisActiveMember=SIP/${CHANNEL(peername)})          ;
initialize 'thisActiveMember' as current device

exten => member_check,n,GotoIf($["${queueMembers}" = ""]?q_login,1)             ; short
circuit to logging in if we don't have
                                                                                ; any
members logged into this queue
```

At this point if there are no members currently logged into our sales queue, we then short-circuit our dialplan to go to the 'q_login' extension since there is no point in wasting cycles searching to see if we're already logged in.

The next step is to finish initializing some values we need within the While() loop that we'll use to check if we're already logged into the queue. We set our ${field} variable to 1, which will be used as the field number offset in the CUT() function.

```
; Initialize some values we'll use in the While() loop
exten => member_check,n,Set(field=1)                                            ; start
our field counter at one
exten => member_check,n,Set(logged_in=0)                                        ;
initialize 'logged_in' to "not logged in"
exten => member_check,n,Set(thisQueueMember=${CUT(queueMembers,\,,${field})})   ;
initialize 'thisQueueMember' with the value in the
                                                                                ;   first
field of the comma-separated list
```

Now we get to enter our While() loop to determine if we're already logged in.

```
; Enter our loop to check if our member is already logged into this queue
exten => member_check,n,While($[${EXISTS(${thisQueueMember})}])
; while we have a queue member...
```

This is where we check to see if the member at this position of the list is the same as the device we're calling from. If it doesn't match, then we go to the 'check_next' priority label (where we increase our ${field} counter variable). If it does match, then we continue on in the dialplan.

```
exten => member_check,n,GotoIf($["${thisQueueMember}" !=
"${thisActiveMember}"]?check_next)     ; if 'thisQueueMember' is not the

;   same as our active peer, then

;   check the next in the list of

;   logged in queue members
```

If we continued on in the dialplan, then we set the ${logged_in} channel variable to '1' which represents we're already logged into this queue. We then exit the While() loop with the ExitWhile() dialplan application.

```
exten => member_check,n,Set(logged_in=1)
; if we got here, set as logged in
exten => member_check,n,ExitWhile()
;  then exit our loop
```

If we didn't match this peer name in the list, then we increase our ${field} counter variable by one, update the ${thisQueueMember} channel variable and then move back to the top of the loop for another round of checks.

```
exten => member_check,n(check_next),Set(field=$[${field} + 1])
; if we got here, increase counter
exten => member_check,n,Set(thisQueueMember=${CUT(queueMembers,\,,${field})})
; get next member in the list
exten => member_check,n,EndWhile()
; ...end of our loop
```

And once we exit our loop, we determine whether we need to log our device in or out of the queue.

```
; if not logged in, then login to this queue, otherwise, logout
exten => member_check,n,GotoIf($[${logged_in} = 0]?q_login,1:q_logout,1)        ; if not
logged in, then login, otherwise, logout
```

The following two extensions are used to either log the device in or out of the queue. We use the AddQueueMember() and RemovQueueMember() applications to login or logout the device from the queue.

The first two arguments for AddQueueMember() and RemoveQueueMember() are 'queue' and 'device'. There are additional arguments we can pass, and you can check those out with 'core show application AddQueueMember' and 'core show application RemoveQueueMember()'.

```
; ### Login queue member ###
exten => q_login,1,Verbose(2,Logging ${thisActiveMember} into the ${thisQueue} queue)
exten => q_login,n,AddQueueMember(${thisQueue},${thisActiveMember})                     ;
login our active device to the queue
                                                                                         ;
requested
exten => q_login,n,Playback(silence/1)  ; answer the channel by playing one second of
silence

; If the member was added to the queue successfully, then playback "Agent logged in",
otherwise, state an error occurred
exten => q_login,n,ExecIf($["${AQMSTATUS}" =
"ADDED"]?Playback(agent-loginok):Playback(an-error-has-occurred))
exten => q_login,n,Hangup()


; ### Logout queue member ###
exten => q_logout,1,Verbose(2,Logging ${thisActiveMember} out of ${thisQueue} queue)
exten => q_logout,n,RemoveQueueMember(${thisQueue},${thisActiveMember})
exten => q_logout,n,Playback(silence/1)
exten => q_logout,n,ExecIf($["${RQMSTATUS}" =
"REMOVED"]?Playback(agent-loggedoff):Playback(an-error-has-occurred))
exten => q_logout,n,Hangup()
```

And that's it! Give it a shot and you should see console output similar to the following which will login and logout your queue members to the queues you've configured.

You can see there are already a couple of queue members logged into the sales queue.

```
*CLI> queue show sales
sales        has 0 calls (max unlimited) in 'rrmemory' strategy (3s holdtime, 4s
talktime), W:0, C:2, A:1, SL:0.0% within 0s
   Members:
      SIP/0004f2040001 (dynamic) (Not in use) has taken no calls yet
      SIP/0004f2040002 (dynamic) (Not in use) has taken no calls yet
   No Callers
```

Then we dial *100 to logout the active device from the sales queue.

```
*CLI>   == Using SIP RTP CoS mark 5
    -- Executing [*100@devices:1] Set("SIP/0004f2040001-00000012", "xtn=100") in new
stack
    -- Executing [*100@devices:2] Goto("SIP/0004f2040001-00000012",
"queueLoginLogout,member_check,1") in new stack
    -- Goto (queueLoginLogout,member_check,1)
    -- Executing [member_check@queueLoginLogout:1] Verbose("SIP/0004f2040001-00000012",
"2,Logging queue member in or out of the request queue") in new stack
  == Logging queue member in or out of the request queue
    -- Executing [member_check@queueLoginLogout:2] Set("SIP/0004f2040001-00000012",
"thisQueue=sales") in new stack
    -- Executing [member_check@queueLoginLogout:3] Set("SIP/0004f2040001-00000012",
"queueMembers=SIP/0004f2040001,SIP/0004f2040002") in new stack
    -- Executing [member_check@queueLoginLogout:4] Set("SIP/0004f2040001-00000012",
"thisActiveMember=SIP/0004f2040001") in new stack
    -- Executing [member_check@queueLoginLogout:5] GotoIf("SIP/0004f2040001-00000012",
"0?q_login,1") in new stack
    -- Executing [member_check@queueLoginLogout:6] Set("SIP/0004f2040001-00000012",
"field=1") in new stack
    -- Executing [member_check@queueLoginLogout:7] Set("SIP/0004f2040001-00000012",
"logged_in=0") in new stack
    -- Executing [member_check@queueLoginLogout:8] Set("SIP/0004f2040001-00000012",
"thisQueueMember=SIP/0004f2040001") in new stack
    -- Executing [member_check@queueLoginLogout:9] While("SIP/0004f2040001-00000012",
"1") in new stack
    -- Executing [member_check@queueLoginLogout:10] GotoIf("SIP/0004f2040001-00000012",
"0?check_next") in new stack
    -- Executing [member_check@queueLoginLogout:11] Set("SIP/0004f2040001-00000012",
"logged_in=1") in new stack
    -- Executing [member_check@queueLoginLogout:12]
ExitWhile("SIP/0004f2040001-00000012", "") in new stack
    -- Jumping to priority 15
    -- Executing [member_check@queueLoginLogout:16] GotoIf("SIP/0004f2040001-00000012",
"0?q_login,1:q_logout,1") in new stack
    -- Goto (queueLoginLogout,q_logout,1)
    -- Executing [q_logout@queueLoginLogout:1] Verbose("SIP/0004f2040001-00000012",
"2,Logging SIP/0004f2040001 out of sales queue") in new stack
  == Logging SIP/0004f2040001 out of sales queue
    -- Executing [q_logout@queueLoginLogout:2]
RemoveQueueMember("SIP/0004f2040001-00000012", "sales,SIP/0004f2040001") in new stack
[Nov 12 12:08:51] NOTICE[11582]: app_queue.c:4842 rqm_exec: Removed interface
'SIP/0004f2040001' from queue 'sales'
    -- Executing [q_logout@queueLoginLogout:3] Playback("SIP/0004f2040001-00000012",
"silence/1") in new stack
    -- <SIP/0004f2040001-00000012> Playing 'silence/1.slin' (language 'en')
    -- Executing [q_logout@queueLoginLogout:4] ExecIf("SIP/0004f2040001-00000012",
"1?Playback(agent-loggedoff):Playback(an-error-has-occurred)") in new stack
    -- <SIP/0004f2040001-00000012> Playing 'agent-loggedoff.slin' (language 'en')
    -- Executing [q_logout@queueLoginLogout:5] Hangup("SIP/0004f2040001-00000012", "") in
new stack
  == Spawn extension (queueLoginLogout, q_logout, 5) exited non-zero on
'SIP/0004f2040001-00000012'
```

And we can see that the device we loggd out by running 'queue show sales'.

```
*CLI> queue show sales
sales        has 0 calls (max unlimited) in 'rrmemory' strategy (3s holdtime, 4s
talktime), W:0, C:2, A:1, SL:0.0% within 0s
   Members:
      SIP/0004f2040002 (dynamic) (Not in use) has taken no calls yet
   No Callers
```

Pausing and Unpausing Members of Queues

Once we have our queue members logged in, it is inevitable that they will want to pause themselves during breaks, and other short periods of inactivity. To do this we can utilize the 'queue pause' and 'queue unpause' CLI commands.

We have two devices logged into the sales queue as we can see with the 'queue show sales' CLI command.

```
*CLI> queue show sales
sales        has 0 calls (max unlimited) in 'rrmemory' strategy (0s holdtime, 0s
talktime), W:0, C:0, A:0, SL:0.0% within 0s
   Members:
      SIP/0004f2040002 (dynamic) (Not in use) has taken no calls yet
      SIP/0004f2040001 (dynamic) (Not in use) has taken no calls yet
   No Callers
```

We can then pause our devices with 'queue pause' which has the following format.

```
Usage: queue {pause|unpause} member <member> [queue <queue> [reason <reason>]]
 Pause or unpause a queue member. Not specifying a particular queue
 will pause or unpause a member across all queues to which the member
 belongs.
```

Lets pause device 0004f2040001 in the sales queue by executing the following.

```
*CLI> queue pause member SIP/0004f2040001 queue sales
paused interface 'SIP/0004f2040001' in queue 'sales' for reason 'lunch'
```

And we can see they are paused with 'queue show sales'.

```
*CLI> queue show sales
sales        has 0 calls (max unlimited) in 'rrmemory' strategy (0s holdtime, 0s
talktime), W:0, C:0, A:0, SL:0.0% within 0s
   Members:
      SIP/0004f2040002 (dynamic) (Not in use) has taken no calls yet
      SIP/0004f2040001 (dynamic) (paused) (Not in use) has taken no calls yet
   No Callers
```

At this point the queue member will no longer receive calls from the system. We can unpause them with the CLI command 'queue unpause member'.

```
*CLI> queue unpause member SIP/0004f2040001 queue sales
unpaused interface 'SIP/0004f2040001' in queue 'sales'
```

And if you don't specify a queue, it will pause or unpause from all queues.

```
*CLI> queue pause member SIP/0004f2040001
paused interface 'SIP/0004f2040001'
```

Of course we want to allow the agents to pause and unpause themselves from their devices, so we need to create an extension and some dialplan logic for that to happen.

Below we've created the pattern patch **_0[01]! which will match on *00 and *01, and will *also** match with zero or more digits following it, such as the queue extension number.

So if we want to pause ourselves in all queues, we can dial *00; unpausing can be done with *01. But if our agents just need to pause or unpause themselves from a single queue, then we will also accept *00100 to pause in queue 100 (sales), or we can unpause ourselves from sales with *01100.

```
extensions.conf
---------------

; Allow queue members to pause and unpause themselves from all queues, or an individual
queue.
;
; _*0[01]! pattern match will match on *00 and *01 plus 0 or more digits.
exten => _*0[01]!,1,Verbose(2,Pausing or unpausing queue member from one or more queues)
exten => _*0[01]!,n,Set(xtn=${EXTEN:3})
; save the queue extension to 'xtn'
exten => _*0[01]!,n,Set(thisQueue=${GLOBAL(QUEUE_${xtn})})
; get the queue name if available
exten => _*0[01]!,n,GotoIf($[${ISNULL(${thisQueue})} &
${EXISTS(${xtn})}]?invalid_queue,1)  ; if 'thisQueue' is blank and the

;  the agent dialed a queue exten,

;  we will tell them it's invalid
```

The following line will determine if we're trying to pause or unpause. This is done by taking the value dialed (e.g. *00100) and chopping off the first 2 digits which leaves us with 0100, and then the :1 will return the next digit, which in this case is '0' that we're using to signify that the queue member wants to be paused (in queue 100).

So we're doing the following with our EXTEN variable.

```
      ${EXTEN:2:1}
offset        ^ ^  length
```

Which causes the following.

```
 *00100
 ^^      offset these characters

 *00100
   ^     then return a digit length of one, which is digit 0
```

```
exten => _*0[01]!,n,GotoIf($[${EXTEN:2:1} = 0]?pause,1:unpause,1)                      ;
determine if they wanted to pause
                                                                                        ;
or to unpause.
```

The following two extensions, pause & unpause, are used for pausing and unpausing our extension from the queue(s). We use the PauseQueueMember() and UnpauseQueueMember() dialplan applications which accept the queue name (optional) and the queue member name. If the queue name is not provided, then it is assumed we want to pause or unpause from all logged in queues.

```
; Unpause ourselves from one or more queues
exten => unpause,1,NoOp()
exten => unpause,n,UnpauseQueueMember(${thisQueue},SIP/${CHANNEL(peername)})            ;
if 'thisQueue' is populated we'll pause in
                                                                                        ;
that queue, otherwise, we'll unpause in
                                                                                        ;
in all queues
```

Once we've unpaused ourselves, we use GoSub() to perform some common dialplan logic that is used for pausing and unpausing. We pass three arguments to the subroutine:

- variable name that contains the result of our operation
- the value we're expecting to get back if successful
- the filename to play

```
exten => unpause,n,GoSub(changePauseStatus,start,1(UPQMSTATUS,UNPAUSED,available))      ;
use the changePauseStatus subroutine and
                                                                                        ;
pass the values for: variable to check,
                                                                                        ;
value to check for, and file to play
exten => unpause,n,Hangup()
```

And the same method is done for pausing.

```
; Pause ourselves in one or more queues
exten => pause,1,NoOp()
exten => pause,n,PauseQueueMember(${thisQueue},SIP/${CHANNEL(peername)})
exten => pause,n,GoSub(changePauseStatus,start,1(PQMSTATUS,PAUSED,unavailable))
exten => pause,n,Hangup()
```

Lets explore what happens in the subroutine we're using for pausing and unpausing.

```
; ### Subroutine we use to check pausing and unpausing status ###
[changePauseStatus]
; ARG1:  variable name to check, such as PQMSTATUS and UPQMSTATUS (PauseQueueMemberStatus
/ UnpauseQueueMemberStatus)
; ARG2:  value to check for, such as PAUSED or UNPAUSED
; ARG3:  file to play back if our variable value matched the value to check for
;
exten => start,1,NoOp()
exten => start,n,Playback(silence/1)
; answer line with silence
```

The following line is probably the most complex. We're using the IF() function inside the Playback() application which determines which file to playback to the user.

Those three values we passed in from the pause and unpause extensions could have been something like:

- ARG1 – PQMSTATUS
- ARG2 – PAUSED
- ARG3 – unavailable

So when expanded, we'd end up with the following inside the IF() function.

```
$["${PQMSTATUS}" = "PAUSED"]?unavailable:not-yet-connected
```

${PQMSTATUS} would then be expanded further to contain the status of our PauseQueueMember() dialplan application, which could either be PAUSED or NOTFOUND. So if ${PQMSTATUS} returned PAUSED, then it would match what we're looking to match on, and we'd then return 'unavailable' to Playback() that would tell the user they are now unavailable.

Otherwise, we'd get back a message saying "not yet connected" to indicate they are likely not logged into the queue they are attempting to change status in.

```
; Please note that ${ARG1} is wrapped in ${  } in order to expand the value of ${ARG1}
into
;   the variable we want to retrieve the value from, i.e. ${${ARG1}} turns into
${PQMSTATUS}
exten => start,n,Playback(${IF($["${${ARG1}}" = "${ARG2}"]?${ARG3}:not-yet-connected)})
; check if value of variable

;  matches the value we're looking

;  for and playback the file we want

;  to play if it does
```

If ${xtn} is null, then we just go to the end of the subroutine, but if it isn't then we will play back "in the queue" followed by the queue extension number indicating which queue they were (un)paused from.

```
exten => start,n,GotoIf($[${ISNULL(${xtn})}]?end)       ; if ${xtn} is null, then just
Return()
exten => start,n,Playback(in-the-queue)                 ;   if not null, then playback
"in the queue"
exten => start,n,SayNumber(${xtn})                      ;   and the queue number that we
(un)paused from
exten => start,n(end),Return()                          ; return from were we came
```

Conclusion

You should now have a simple system that permits you to login and out of queues you create in queues.conf, and to allow queue members to pause themselves within one or more queues. There are a lot of dialplan concepts utilized in this
article, so you are encouraged to seek out additional documentation if any of these concepts are a bit fuzzy for you.

A good start is the doc/ subdirectory of the Asterisk sources, or the various configuration samples files located in the configs/ subdirectory of your Asterisk source code.

Call Completion Supplementary Services

Placeholder page for CCSS information.

Call Queues

Template holder while we wait for input on a good README for call queues. Please open a bug report and add text to this document

General advice on the agent channel

Using dynamic queue members

SIP channel configuration

Queues depend on the channel driver reporting the proper state for each member of the queue. To get proper signalling on
queue members that use the SIP channel driver, you need to enable a call limit (could be set to a high value so it
is not put into action) and also make sure that both inbound and outbound calls are accounted for.

Example:

```
[general]
limitonpeer = yes

[peername]
type=friend
call-limit=10
```

Other references

- queuelog.txt
- queues-with-callback-members.txt

(Should we merge those documents into this?)

Channel Drivers

placeholder page to store the various channel driver subpages

Corosync

Corosync

Corosync is an open source group messaging system typically used in clusters, cloud computing, and other high availability environments.

The project, at it's 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>
===
=============================================================
```

Database Transactions

As of 1.6.2, Asterisk now supports doing database transactions from the Dialplan. A number of new applications and functions have been introduced for this purpose and this document should hopefully familiarize you with all of them.

First, the `ODBC()` function has been added which is used to set up all new database transactions. Simply write the name of the transaction to this function, along with the arguments of "transaction" and the database name, e.g. `Set(ODBC(transaction,postgres-asterisk)=foo)`. In this example, the name of the transaction is "foo". The name doesn't really matter, unless you're manipulating multiple transactions within the same dialplan, at the same time. Then, you use the transaction name to change which transaction is active for the next dialplan function.

The `ODBC()` function is also used to turn on a mode known as `forcecommit`. For most cases, you won't need to use this, but it's there. It simply causes a transaction to be committed, when the channel hangs up. The other property which may be set is the `isolation` property. Please consult with your database vendor as to which values are supported by their ODBC driver. Asterisk supports setting all standard ODBC values, but many databases do not support the entire complement.

Finally, when you have run multiple statements on your transaction and you wish to complete the transaction, use the `ODBC_Commit` and `ODBC_Rollback` applications, along with the transaction ID (in the example above, "foo") to commit or rollback the transaction. Please note that if you do not explicitly commit the transaction or if `forcecommit` is not turned on, the transaction will be automatically rolled back at channel destruction (after hangup) and all related database resources released back to the pool.

Distributed Device State

Distributed Device State with AIS

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() diaplan 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

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-b1858/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 securly 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 connec to using the 'buddy' option.

*Asterisk Server 1

jabber.conf on server1

```
[general]
debug=no                                ;;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=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() diaplan 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.

DUNDi - Distributed Universal Number Discovery

Digium General Peering Agreement

```
DIGIUM GENERAL PEERING AGREEMENT (TM)
                      Version 1.0.0, September 2004
 Copyright (C) 2004 Digium, Inc.
                445 Jan Davis Drive, Huntsville, AL 35806 USA

 Everyone is permitted to copy and distribute complete verbatim copies
 of this General Peering Agreement provided it is not modified in any
 manner.

        ------------------------------------------------------

                    DIGIUM GENERAL PEERING AGREEMENT

                                PREAMBLE

  For most of the history of telecommunications, the power of being able
to locate and communicate with another person in a system, be it across
a hall or around the world, has always centered around a centralized
authority -- from a local PBX administrator to regional and national
RBOCs, generally requiring fees, taxes or regulation.  By contrast,
DUNDi is a technology developed to provide users the freedom to
communicate with each other without the necessity of any centralized
authority.  This General Peering Agreement ("GPA") is used by individual
parties (each, a "Participant") to allow them to build the E164 trust
group for the DUNDi protocol.

  To protect the usefulness of the E164 trust group for those who use
it, while keeping the system wholly decentralized, it is necessary to
replace many of the responsibilities generally afforded to a company or
government agency, with a set of responsibilities implemented by the
parties who use the system, themselves.  It is the goal of this document
to provide all the protections necessary to keep the DUNDi E164 trust
group useful and reliable.

  The Participants wish to protect competition, promote innovation and
value added services and make this service valuable both commercially
and non-commercially.  To that end, this GPA provides special terms and
conditions outlining some permissible and non-permissible revenue
sources.

  This GPA is independent of any software license or other license
agreement for a program or technology employing the DUNDi protocol.  For
example, the implementation of DUNDi used by Asterisk is covered under a
separate license.  Each Participant is responsible for compliance with
any licenses or other agreements governing use of such program or
technology that they use to peer.

  You do not have to execute this GPA to use a program or technology
employing the DUNDi protocol, however if you do not execute this GPA,
you will not be able to peer using DUNDi and the E164 context with
anyone who is a member of the trust group by virtue of their having
executed this GPA with another member.

The parties to this GPA agree as follows:

  0. DEFINITIONS.  As used herein, certain terms shall be defined as
follows:

    (a) The term "DUNDi" means the DUNDi protocol as published by
        Digium, Inc. or its successor in interest with respect to the
        DUNDi protocol specification.

    (b) The terms "E.164" and "E164" mean ITU-T specification E.164 as
        published by the International Telecommunications Union (ITU) in
        May, 1997.

    (c) The term "Service" refers to any communication facility (e.g.,
        telephone, fax, modem, etc.), identified by an E.164-compatible
        number, and assigned by the appropriate authority in that
        jurisdiction.

    (d) The term "Egress Gateway" refers an Internet facility that
        provides a communications path to a Service or Services that may
        not be directly addressable via the Internet.

    (e) The term "Route" refers to an Internet address, policies, and
        other characteristics defined by the DUNDi protocol and
        associated with the Service, or the Egress Gateway which
        provides access to the specified Service.

    (f) The term "Propagate" means to accept or transmit Service and/or
        Egress Gateway Routes only using the DUNDi protocol and the
        DUNDi context "e164" without regard to case, and does not apply
        to the exchange of information using any other protocol or
        context.
    (g) The term "Peering System" means the network of systems that
        Propagate Routes.

    (h) The term "Subscriber" means the owner of, or someone who
        contracts to receive, the services identified by an E.164
        number.

    (i) The term "Authorizing Individual" means the Subscriber to a
        number who has authorized a Participant to provide Routes
        regarding their services via this Peering System.

    (j) The term "Route Authority" refers to a Participant that provides
        an original source of said Route within the Peering System.
        Routes are propagated from the Route Authorities through the
        Peering System and may be cached at intermediate points.  There
        may be multiple Route Authorities for any Service.

    (k) The term "Participant" (introduced above) refers to any member
        of the Peering System.

    (l) The term "Service Provider" refers to the carrier (e.g.,
        exchange carrier, Internet Telephony Service Provider, or other
        reseller) that provides communication facilities for a
        particular Service to a Subscriber, Customer or other End User.

    (m) The term "Weight" refers to a numeric quality assigned to a
        Route as per the DUNDi protocol specification.  The current
        Weight definitions are shown in Exhibit A.

  1. PEERING. The undersigned Participants agree to Propagate Routes
with each other and any other member of the Peering System and further
agree not to Propagate DUNDi Routes with a third party unless they have
first have executed this GPA (in its unmodified form) with such third
party.  The Participants further agree only to Propagate Routes with
Participants whom they reasonably believe to be honoring the terms of
the GPA.  Participants may not insert, remove, amend, or otherwise
modify any of the terms of the GPA.

  2. ACCEPTABLE USE POLICY.  The DUNDi protocol contains information
that reflect a Subscriber's or Egress Gateway's decisions to receive
calls.  In addition to the terms and conditions set forth in this GPA,
the Participants agree to honor the intent of restrictions encoded in
the DUNDi protocol.  To that end, Participants agree to the following:

    (a) A Participant may not utilize or permit the utilization of
        Routes for which the Subscriber or Egress Gateway provider has
        indicated that they do not wish to receive "Unsolicited Calls"
        for the purpose of making an unsolicited phone call on behalf of
        any party or organization.

    (b) A Participant may not utilize or permit the utilization of
        Routes which have indicated that they do not wish to receive
        "Unsolicited Commercial Calls" for the purpose of making an
        unsolicited phone call on behalf of a commercial organization.

    (c) A Participant may never utilize or permit the utilization of any
        DUNDi route for the purpose of making harassing phone calls.

    (d) A Party may not utilize or permit the utilization of DUNDi
        provided Routes for any systematic or random calling of numbers
        (e.g., for the purpose of locating facsimile, modem services, or
        systematic telemarketing).

    (e) Initial control signaling for all communication sessions that
        utilize Routes obtained from the Peering System must be sent
        from a member of the Peering System to the Service or Egress
        Gateway identified in the selected Route.  For example, 'SIP
        INVITES' and IAX2 "NEW" commands must be sent from the
        requesting DUNDi node to the terminating Service.

    (f) A Participant may not disclose any specific Route, Service or
        Participant contact information obtained from the Peering System
        to any party outside of the Peering System except as a
        by-product of facilitating communication in accordance with
        section 2e (e.g., phone books or other databases may not be
        published, but the Internet addresses of the Egress Gateway or
        Service does not need to be obfuscated.)

    (g) The DUNDi Protocol requires that each Participant include valid
        contact information about itself (including information about
        nodes connected to each Participant).  Participants may use or
        disclose the contact information only to ensure enforcement of
        legal furtherance of this Agreement.

  3. ROUTES. The Participants shall only propagate valid Routes, as
defined herein, through the Peering System, regardless of the original
source.  The Participants may only provide Routes as set forth below,
and then only if such Participant has no good faith reason to believe
```

```
such Route to be invalid or unauthorized.

    (a) A Participant may provide Routes if each Route has as its
        original source another member of the Peering System who has
        duly executed the GPA and such Routes are provided in accordance
        with this Agreement; provided that the Routes are not modified
        (e.g., with regards to existence, destination, technology or
        Weight); or

    (b) A Participant may provide Routes for Services with any Weight
        for which it is the Subscriber; or

    (c) A Participant may provide Routes for those Services whose
        Subscriber has authorized the Participant to do so, provided
        that the Participant is able to confirm that the Authorizing
        Individual is the Subscriber through:

            i. a written statement of ownership from the Authorizing
               Individual, which the Participant believes in good faith
               to be accurate (e.g., a phone bill with the name of the
               Authorizing Individual and the number in question); or

           ii. the Participant's own direct personal knowledge that the
               Authorizing Individual is the Subscriber.

    (d) A Participant may provide Routes for Services, with Weight in
        accordance with the Current DUNDi Specification, if it can in
        good faith provide an Egress Gateway to that Service on the
        traditional telephone network without cost to the calling party.

  4. REVOCATION. A Participant must provide a free, easily accessible
mechanism by which a Subscriber may revoke permission to act as a Route
Authority for his Service.  A Participant must stop acting as a Route
Authority for that Service within 7 days after:

    (a) receipt of a revocation request;

    (b) receiving other notice that the Service is no longer valid; or

    (c) determination that the Subscriber's information is no longer
        accurate (including that the Subscriber is no longer the service
        owner or the service owner's authorized delegate).

  5. SERVICE FEES. A Participant may charge a fee to act as a Route
Authority for a Service, with any Weight, provided that no Participant
may charge a fee to propagate the Route received through the Peering
System.

  6. TOLL SERVICES. No Participant may provide Routes for any Services
that require payment from the calling party or their customer for
communication with the Service.  Nothing in this section shall prohibit
a Participant from providing routes for Services where the calling party
may later enter into a financial transaction with the called party
(e.g., a Participant may provide Routes for calling cards services).

  7. QUALITY. A Participant may not intentionally impair communication
using a Route provided to the Peering System (e.g. by adding delay,
advertisements, reduced quality).  If for any reason a Participant is
unable to deliver a call via a Route provided to the Peering System,
that Participant shall return out-of-band Network Congestion
notification (e.g. "503 Service Unavailable" with SIP protocol or
"CONGESTION" with IAX protocol).

  8. PROTOCOL COMPLIANCE.  Participants agree to Propagate Routes in
strict compliance with current DUNDi protocol specifications.

  9. ADMINISTRATIVE FEES. A Participant may charge (but is not required
to charge) another Participant a reasonable fee to cover administrative
expenses incurred in the execution of this Agreement.  A Participant may
not charge any fee to continue the relationship or to provide Routes to
another Participant in the Peering System.

  10. CALLER IDENTIFICATION. A Participant will make a good faith effort
to ensure the accuracy and appropriate nature of any caller
identification that it transmits via any Route obtained from the Peering
System. Caller identification shall at least be provided as a valid
E.164 number.

  11. COMPLIANCE WITH LAWS.  The Participants are solely responsible for
determining to what extent, if any, the obligations set forth in this
GPA conflict with any laws or regulations their region.  A Participant
may not provide any service or otherwise use DUNDi under this GPA if
doing so is prohibited by law or regulation, or if any law or regulation
imposes requirements on the Participant that are inconsistent with the
terms of this GPA or the Acceptable Use Policy.

  12. WARRANTY. EACH PARTICIPANT WARRANTS TO THE OTHER PARTICIPANTS THAT
IT MADE, AND WILL CONTINUE TO MAKE, A GOOD FAITH EFFORT TO AUTHENTICATE
OTHERS IN THE PEERING SYSTEM AND TO PROVIDE ACCURATE INFORMATION IN
```

```
ACCORDANCE WITH THE TERMS OF THIS GPA.  THIS WARRANTY IS MADE BETWEEN
THE PARTICIPANTS, AND THE PARTICIPANTS MAY NOT EXTEND THIS WARRANTY TO
ANY NON-PARTICIPANT INCLUDING END-USERS.

  13. DISCLAIMER OF WARRANTIES. THE PARTICIPANTS UNDERSTAND AND AGREE
THAT ANY SERVICE PROVIDED AS A RESULT OF THIS GPA IS "AS IS." EXCEPT FOR
THOSE WARRANTIES OTHERWISE EXPRESSLY SET FORTH HEREIN, THE PARTICIPANTS
DISCLAIM ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND OR NATURE,
EXPRESS OR IMPLIED, AS TO THE CONDITION, VALUE OR QUALITIES OF THE
SERVICES PROVIDED HEREUNDER, AND SPECIFICALLY DISCLAIM ANY
REPRESENTATION OR WARRANTY OF MERCHANTABILITY, SUITABILITY OR FITNESS
FOR A PARTICULAR PURPOSE OR AS TO THE CONDITION OR WORKMANSHIP THEREOF,
OR THE ABSENCE OF ANY DEFECTS THEREIN, WHETHER LATENT OR PATENT,
INCLUDING ANY WARRANTIES ARISING FROM A COURSE OF DEALING, USAGE OR
TRADE PRACTICE.  EXCEPT AS EXPRESSLY PROVIDED HEREIN, THE PARTICIPANTS
EXPRESSLY DISCLAIM ANY REPRESENTATIONS OR WARRANTIES THAT THE PEERING
SERVICE WILL BE CONTINUOUS, UNINTERRUPTED OR ERROR-FREE, THAT ANY DATA
SHARED OR OTHERWISE MADE AVAILABLE WILL BE ACCURATE OR COMPLETE OR
OTHERWISE COMPLETELY SECURE FROM UNAUTHORIZED ACCESS.

  14. LIMITATION OF LIABILITIES.  NO PARTICIPANT SHALL BE LIABLE TO ANY
OTHER PARTICIPANT FOR INCIDENTAL, INDIRECT, CONSEQUENTIAL, SPECIAL,
PUNITIVE OR EXEMPLARY DAMAGES OF ANY KIND (INCLUDING LOST REVENUES OR
PROFITS, LOSS OF BUSINESS OR LOSS OF DATA) IN ANY WAY RELATED TO THIS
GPA, WHETHER IN CONTRACT OR IN TORT, REGARDLESS OF WHETHER SUCH
PARTICIPANT WAS ADVISED OF THE POSSIBILITY THEREOF.

  15. END-USER AGREEMENTS.  The Participants may independently enter
into agreements with end-users to provide certain services (e.g., fees
to a Subscriber to originate Routes for that Service).  To the extent
that provision of these services employs the Peering System, the Parties
will include in their agreements with their end-users terms and
conditions consistent with the terms of this GPA with respect to the
exclusion of warranties, limitation of liability and Acceptable Use
Policy.  In no event may a Participant extend the warranty described in
Section 12 in this GPA to any end-users.

  16. INDEMNIFICATION.  Each Participant agrees to defend, indemnify and
hold harmless the other Participant or third-party beneficiaries to this
GPA (including their affiliates, successors, assigns, agents and
representatives and their respective officers, directors and employees)
from and against any and all actions, suits, proceedings,
investigations, demands, claims, judgments, liabilities, obligations,
liens, losses, damages, expenses (including, without limitation,
attorneys' fees) and any other fees arising out of or relating to (i)
personal injury or property damage caused by that Participant, its
employees, agents, servants, or other representatives; (ii) any act or
omission by the Participant, its employees, agents, servants or other
representatives, including, but not limited to, unauthorized
representations or warranties made by the Participant; or (iii) any
breach by the Participant of any of the terms or conditions of this GPA.

  17. THIRD PARTY BENEFICIARIES. This GPA is intended to benefit those
Participants who have executed the GPA and who are in the Peering
System. It is the intent of the Parties to this GPA to give to those
Participants who are in the Peering System standing to bring any
necessary legal action to enforce the terms of this GPA.

  18. TERMINATION. Any Participant may terminate this GPA at any time,
with or without cause.  A Participant that terminates must immediately
cease to Propagate.

  19. CHOICE OF LAW. This GPA and the rights and duties of the Parties
hereto shall be construed and determined in accordance with the internal
laws of the State of New York, United States of America, without regard
to its conflict of laws principles and without application of the United
Nations Convention on Contracts for the International Sale of Goods.

  20. DISPUTE RESOLUTION. Unless otherwise agreed in writing, the
exclusive procedure for handling disputes shall be as set forth herein.
Notwithstanding such procedures, any Participant may, at any time, seek
injunctive relief in addition to the process described below.

    (a) Prior to mediation or arbitration the disputing Participants
        shall seek informal resolution of disputes. The process shall be
        initiated with written notice of one Participant to the other
        describing the dispute with reasonable particularity followed
        with a written response within ten (10) days of receipt of
        notice. Each Participant shall promptly designate an executive
        with requisite authority to resolve the dispute.  The informal
        procedure shall commence within ten (10) days of the date of
        response. All reasonable requests for non-privileged information
        reasonably related to the dispute shall be honored. If the
        dispute is not resolved within thirty (30) days of commencement
        of the procedure either Participant may proceed to mediation or
        arbitration pursuant to the rules set forth in (b) or (c) below.

    (b) If the dispute has not been resolved pursuant to (a) above or,
        if the disputing Participants fail to commence informal dispute
```

resolution pursuant to (a) above, either Participant may, in writing and within twenty (20) days of the response date noted in (a) above, ask the other Participant to participate in a one (1) day mediation with an impartial mediator, and the other Participant shall do so. Each Participant will bear its own expenses and an equal share of the fees of the mediator. If the mediation is not successful the Participants may proceed with arbitration pursuant to (c) below.

(c) If the dispute has not been resolved pursuant to (a) or (b) above, the dispute shall be promptly referred, no later than one (1) year from the date of original notice and subject to applicable statute of limitations, to binding arbitration in accordance with the UNCITRAL Arbitration Rules in effect on the date of this contract. The appointing authority shall be the International Centre for Dispute Resolution. The case shall be administered by the International Centre for Dispute Resolution under its Procedures for Cases under the UNCITRAL Arbitration Rules. Each Participant shall bear its own expenses and shall share equally in fees of the arbitrator. All arbitrators shall have substantial experience in information technology and/or in the telecommunications business and shall be selected by the disputing participants in accordance with UNCITRAL Arbitration Rules. If any arbitrator, once selected is unable or unwilling to continue for any reason, replacement shall be filled via the process described above and a re-hearing shall be conducted. The disputing Participants will provide each other with all requested documents and records reasonably related to the dispute in a manner that will minimize the expense and inconvenience of both parties. Discovery will not include depositions or interrogatories except as the arbitrators expressly allow upon a showing of need. If disputes arise concerning discovery requests, the arbitrators shall have sole and complete discretion to resolve the disputes. The parties and arbitrator shall be guided in resolving discovery disputes by the Federal Rules of Civil Procedure. The Participants agree that time of the essence principles shall guide the hearing and that the arbitrator shall have the right and authority to issue monetary sanctions in the event of unreasonable delay. The arbitrator shall deliver a written opinion setting forth findings of fact and the rationale for the award within thirty (30) days following conclusion of the hearing. The award of the arbitrator, which may include legal and equitable relief, but which may not include punitive damages, will be final and binding upon the disputing Participants, and judgment may be entered upon it in accordance with applicable law in any court having jurisdiction thereof. In addition to award the arbitrator shall have the discretion to award the prevailing Participant all or part of its attorneys' fees and costs, including fees associated with arbitrator, if the arbitrator determines that the positions taken by the other Participant on material issues of the dispute were without substantial foundation. Any conflict between the UNCITRAL Arbitration Rules and the provisions of this GPA shall be controlled by this GPA.

21. INTEGRATED AGREEMENT. This GPA, constitutes the complete integrated agreement between the parties concerning the subject matter hereof. All prior and contemporaneous agreements, understandings, negotiations or representations, whether oral or in writing, relating to the subject matter of this GPA are superseded and canceled in their entirety.

22. WAIVER. No waiver of any of the provisions of this GPA shall be deemed or shall constitute a waiver of any other provision of this GPA, whether or not similar, nor shall such waiver constitute a continuing waiver unless otherwise expressly so provided in writing. The failure of either party to enforce at any time any of the provisions of this GPA, or the failure to require at any time performance by either party of any of the provisions of this GPA, shall in no way be construed to be a present or future waiver of such provisions, nor in any way affect the ability of a Participant to enforce each and every such provision thereafter.

23. INDEPENDENT CONTRACTORS. Nothing in this GPA shall make the Parties partners, joint venturers, or otherwise associated in or with the business of the other. Parties are, and shall always remain, independent contractors. No Participant shall be liable for any debts, accounts, obligations, or other liabilities of the other Participant, its agents or employees. No party is authorized to incur debts or other obligations of any kind on the part of or as agent for the other. This GPA is not a franchise agreement and does not create a franchise relationship between the parties, and if any provision of this GPA is deemed to create a franchise between the parties, then this GPA shall automatically terminate.

24. CAPTIONS AND HEADINGS. The captions and headings used in this GPA are used for convenience only and are not to be given any legal effect.

25. EXECUTION. This GPA may be executed in counterparts, each of which

```
so executed will be deemed to be an original and such counterparts
together will constitute one and the same Agreement.  The Parties shall
transmit to each other a signed copy of the GPA by any means that
faithfully reproduces the GPA along with the Signature.  For purposes of
this GPA, the term "signature" shall include digital signatures as
defined by the jurisdiction of the Participant signing the GPA.

                         Exhibit A

Weight Range            Requirements

0-99                    May only be used under authorization of Owner

100-199                 May only be used by the Owner's service
                        provider, regardless of authorization.

200-299                 Reserved -- do not use for e164 context.

300-399                 May only be used by the owner of the code under
                        which the Owner's number is a part of.

400-499                 May be used by any entity providing access via
                        direct connectivity to the Public Switched
                        Telephone Network.

500-599                 May be used by any entity providing access via
                        indirect connectivity to the Public Switched
                        Telephone Network (e.g. Via another VoIP
                        provider)

600-                    Reserved-- do not use for e164 context.

                 Participant                         Participant

Company:

Address:

Email:


          _________________________           _________________________
          Authorized Signature                Authorized Signature

Name:


END OF GENERAL PEERING AGREEMENT

------------------------------------------------

How to Peer using this GPA If you wish to exchange routing information
with parties using the e164 DUNDi context, all you must do is execute
this GPA with any member of the Peering System and you will become a
member of the Peering System and be able to make Routes available in
```

```
accordance with this GPA.

DUNDi, IAX, Asterisk and GPA are trademarks of Digium, Inc.
```

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/applcation[(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,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` opt ion is specified.

Errors

Any newline-terminated (`\n`) output generated by the child process on its `stderr` handle will be copied into the Asterisk log.

Followme - Realtime

Followme is now realtime-enabled.

To use, you must define two backend data structures, with the following fields:

```
followme:
 name                Name of this followme entry.  Specified when invoking the FollowMe
                     application in the dialplan.  This field is the only one which is
                     mandatory.  All of the other fields will inherit the default from
                     followme.conf, if not specified in this data resource.
 musicclass OR       The musiconhold class used for the caller while waiting to be
 musiconhold OR      connected.
    music
 context             Dialplan context from which to dial numbers
 takecall            DTMF used to accept the call and be connected.  For obvious reasons,
                     this needs to be a single digit, '*', or '#'.
 declinecall         DTMF used to refuse the call, sending it onto the next step, if any.
 call_from_prompt    Prompt to play to the callee, announcing the call.
 norecording_prompt  The alternate prompt to play to the callee, when the caller
                     refuses to leave a name (or the option isn't set to allow them).
 options_prompt      Normally, "press 1 to accept, 2 to decline".
 hold_prompt         Message played to the caller while dialing the followme steps.
 status_prompt       Normally, "Party is not at their desk".
 sorry_prompt        Normally, "Unable to locate party".
```

```
followme_numbers:
 name                Name of this followme entry.  Must match the name above.
 ordinal             An integer, specifying the order in which these numbers will be
                     followed.
 phonenumber         The telephone number(s) you would like to call, separated by '&'.
 timeout             Timeout associated with this step.  See the followme documentation
                     for more information on how this value is handled.
```

IAX2 Security

IAX2 Security

Document Version 1.0
09/03/09
Asterisk Development Team <asteriskteam@digium.com>

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.

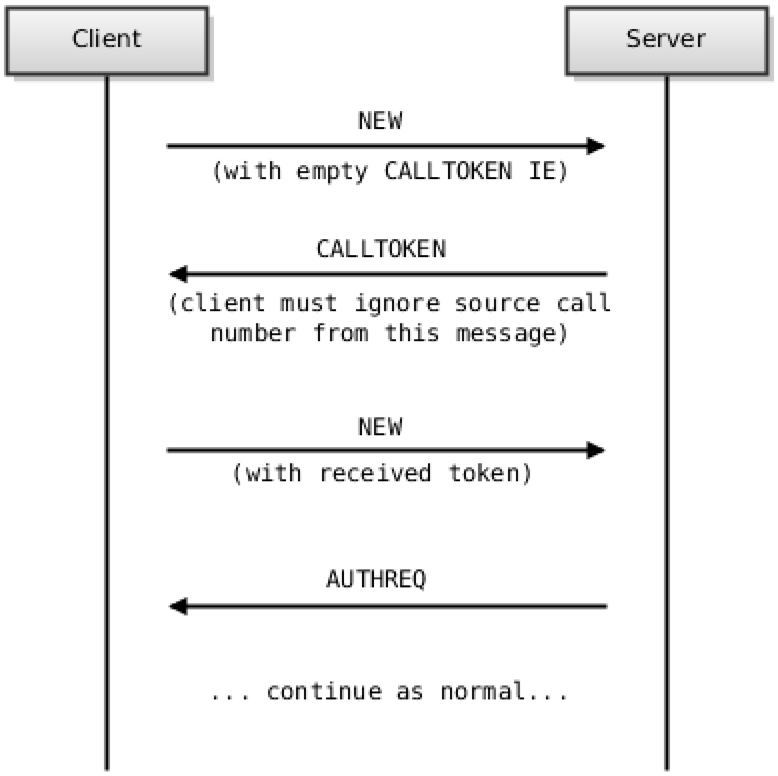
When an IAX2 client receives a CALLTOKEN response, it MUST send its initial request again. This request MUST include the CALLTOKEN IE with a copy of the value of the CALLTOKEN IE received in the CALLTOKEN response. The IE value is an opaque value. Clients MUST be able to accept a CALLTOKEN payload of 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.

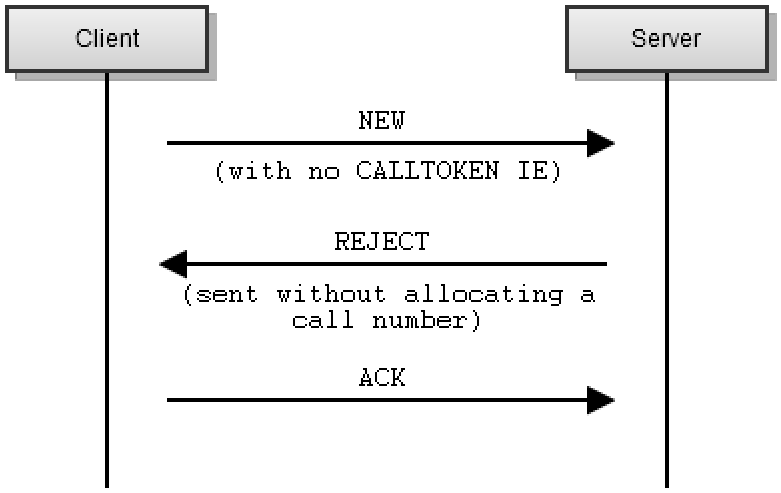
If a server receives a request with an invalid CALLTOKEN IE value, then it MUST drop it and not respond.

Example Message Exchanges

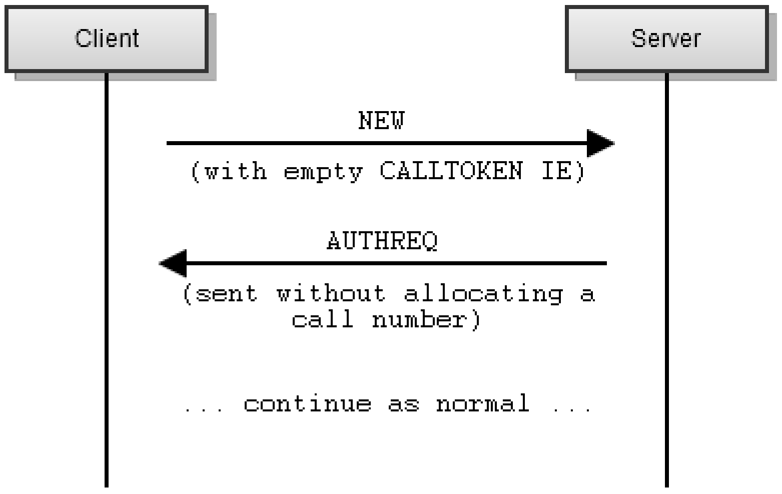
Call Setup


Client
Server
NEW
(with empty CALLTOKEN IE)
CALLTOKEN
(client must ignore source call number from this message)
NEW
(with received token)
AUTHREQ
... continue as normal...

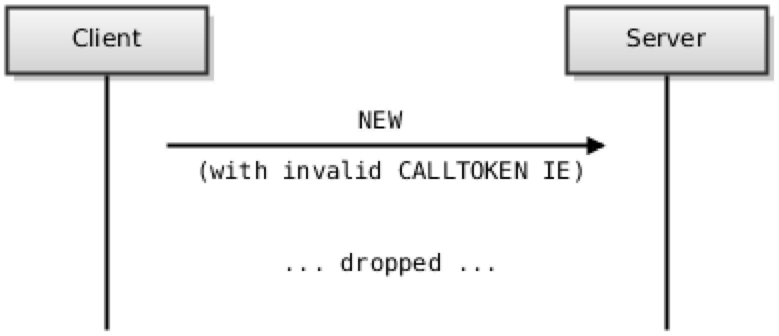
Call Setup, client does not support CALLTOKEN



Call Setup, client supports CALLTOKEN, server does not


Client
Server
NEW
(with empty CALLTOKEN IE)
AUTHREQ
(sent without allocating a call number)
... continue as normal ...

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.

LDAP Realtime Driver

Asterisk Realtime Lightweight Directory Access Protocol (LDAP) Driver

With this driver Asterisk, using the Realtime Database Configuration, can access and update information in an LDAP directory. Asterisk can configure SIP/IAX2 users, extensions, queues, queue members, and entire configuration files. This guide assumes you have a working knowledge of LDAP and have an LDAP server with authentication already setup. Asterisk requires read and write permissions to update the directory.

See configs/res_ldap.conf.sample for a configuration file sample.
See contrib/scripts for the LDAP schema and ldif files needed for the LDAP server.

To use static realtime with certain core configuration files the realtime backend you wish to use must be preloaded in `modules.conf`.

From within your Asterisk source directory:

```
cd contrib/scripts
sudo cp asterisk.ldap-schema /etc/ldap/schema/
sudo service slapd restart
sudo ldapadd -Y EXTERNAL -H ldapi:/// -f ./asterisk.ldif
```

Let's edit the extconfig.conf file to specify LDAP as our realtime storage engine and where Asterisk will look for data.

```
sippeers = ldap,"ou=sip,dc=example,dc=domain",sip
sipusers = ldap,"ou=sip,dc=example,dc=domain",sip
extensions = ldap,"ou=extensions,dc=example,dc=domain",extensions
```

You'll want to reference the Asterisk res_ldap.conf file which holds the LDAP mapping configuration when building your own record schema.

Basic sip users record layout which will need to be saved to a file (we'll use 'createduser.ldif' here as an example). This example record is for sip user '1000'. This example record is for sip user '1000'.

```
dn: cn=1000,ou=sip,dc=digium,dc=internal
objectClass: AsteriskAccount
objectClass: AsteriskExtension
objectClass: AsteriskSIPUser
objectClass: top
AstAccountName: sip user
cn: 1000
AstAccountDefaultUser: 0
AstAccountExpirationTimestamp: 0
AstAccountFullContact: 0
AstAccountHost: dynamic
AstAccountIPAddress: 0
AstAccountLastQualifyMilliseconds: 0
AstAccountPort: 0
AstAccountRegistrationServer: 0
AstAccountType: 0
AstAccountUserAgent: 0
AstExtension: 1000
```

Let's add the record to the LDAP server:

```
sudo ldapadd -D "cn=admin,dc=example,dc=domain" -x -W -f createduser.ldif
```

When creating your own record schema, you'll obviously want to incorporate authentication. Asterisk + LDAP requires that the user secrets be stored as an MD5 hash. MD5 hashes can be created using 'md5sum'.

For AstAccountRealmedPassword authentication use this.

```
printf "<secret composed of username, realm, and password goes here>" | md5sum
```

For AstMD5secret authentication use this.

```
printf "password" | md5sum
```

Open Settlement Protocol (OSP) User Guide

OSP User Guide for Asterisk V1.6

9 February 2007

Table of Contents

Asterisk is a trademark of Digium, Inc.
TransNexus and OSP Secures are trademarks of TransNexus, Inc.

Revision History
Revision Date of Issue Description

```
1       26 Jul 2005   OSP Module User Guide for Asterisk V1.2
1.4     16 Jun 2006   OSP Module User Guide for Asterisk V1.4
1.6.0   13 Dec 2006   OSP Module User Guide for Asterisk V1.6
1.6.1   4 Jan 2007    Clarifying edits, add revision history, add general
                      purpose extensions.conf example
1.6.2   9 Feb 2007    Replace OSP Toolkit site from SIPfoundry with
                      SourceForge
```

1 Introduction
This document provides instructions on how to build and configure Asterisk V1.6 with the OSP Toolkit to enable secure, multi-lateral peering. This document is also available in the Asterisk source package as doc/osp.txt. The OSP Toolkit is an open source implementation of the OSP peering protocol and is freely available from https://sourceforge.net/projects/osp-toolkit. The OSP standard defined by the European Telecommunications Standards Institute (ETSI TS 101 321) www.etsi.org. If you have questions or need help, building Asterisk with the OSP Toolkit, please post your question on the OSP mailing list at https://lists.sourceforge.net/lists/listinfo/osp-toolkit-client.

2 OSP Toolkit
Please reference the OSP Toolkit document "How to Build and Test the OSP Toolkit" available from https://sourceforge.net/projects/osp-toolkit.

2.1 Build OSP Toolkit
The software listed below is required to build and use the OSP Toolkit:

- OpenSSL (required for building) - Open Source SSL protocol and Cryptographic Algorithms (version 0.9.7g recommended) from www.openssl.org. Pre-compiled OpenSSL binary packages are not recommended because of the binary compatibility issue.

- Perl (required for building) - A programming language used by OpenSSL for compilation. Any version of Perl should work. One version of Perl is available from www.activestate.com/Products/ActivePer. If pre-compiled OpenSSL packages are used, Perl package is not required.
- C compiler (required for building) - Any C compiler should work. The GNU Compiler Collection from www.gnu.org is routinely used for building the OSP Toolkit for testing.
- OSP Server (required for testing) - Access to any OSP server should work. An open source reference OSP server developed by Cisco System is available at http://www.vovida.org/applications/downloads/openosp/. RAMS, a java based open source OSP server is available at https://sourceforge.net/projects/rams. A free version of the TransNexus commercial OSP server may be downloaded from http://www.transnexus.com/OSP%20Toolkit/Peering_Server/VoIP_Peering_Server.htm.

2.1.1 Unpacking the Toolkit
After downloading the OSP Toolkit (version 3.3.6 or later release) from www.sourceforge.net, perform the following steps in order:

- Copy the OSP Toolkit distribution into the directory where it will reside. The default directory for the OSP Toolkit is /usr/src.

*Un-package the distribution file by executing the following command:

```
gunzip –c OSPToolkit-###.tar.gz | tar xvf –
```

Where ### is the version number separated by underlines. For example, if the version is 3.3.6, then the above command would be:

```
gunzip –c OSPToolkit-3_3_6.tar.gz | tar xvf –
```

A new directory (TK-3_3_6-20060303) will be created within the same directory as the tar file.

- Go to the TK-3_3_6-20060303 directory by running this command:

```
cd TK-3_3_6-20060303
```

Within this directory, you will find directories and files similar to what is listed below if the command "ls -F" is executed):

```
 ls -F
enroll/
RelNotes.txt    lib/
README.txt    license.txt
bin/     src/
crypto/     test/
include/
```

2.1.2 Preparing to build the OSP Toolkit

- Compile OpenSSL according to the instructions provided with the OpenSSL distribution (You would need to do this only if you don’t have openssl already).
- Copy the OpenSSL header files (the *.h files) into the crypto/openssl directory within the osptoolkit directory. The OpenSSL header files are located under the openssl/include/openssl directory.
- Copy the OpenSSL library files (libcrypto.a and libssl.a) into the lib directory within the osptoolkit directory. The OpenSSL library files are located under the openssl directory.
  Note: Since the Asterisk requires the OpenSSL package. If the OpenSSL package has been installed, steps 4 through 6 are not necessary.
- Optionally, change the install directory of the OSP Toolkit. Open the Makefile in the /usr/src/TK-3_3_6-20060303/src directory, look for the install path variable – INSTALL_PATH, and edit it to be anywhere you want (defaults /usr/local).
  Note: Please change the install path variable only if you are familiar with both the OSP Toolkit and the Asterisk.

2.1.3 Building the OSP Toolkit

- From within the OSP Toolkit directory (/usr/src/TK-3_3_6-20060303), start the compilation script by executing the following commands:

```
cd src
make clean; make build
```

2.1.4 Installing the OSP Toolkit

The header files and the library of the OSP Toolkit should be installed. Otherwise, you must specify the OSP Toolkit path for the Asterisk.

- Use the make script to install the Toolkit.

```
make install
```

The make script is also used to install the OSP Toolkit header files and the library into the INSTALL_PATH specified in the Makefile.

> ⚠ Please make sure you have the rights to access the INSTALL_PATH directory. For example, in order to access /usr/local directory, root privileges are required.

2.1.5 Building the Enrollment Utility

Device enrollment is the process of establishing a trusted cryptographic relationship between the VoIP device and the OSP Server. The Enroll program is a utility application for establishing a trusted relationship between an OSP client and an OSP server. Please see the document "Device Enrollment" at http://www.transnexus.com/OSP%20Toolkit/OSP%20Toolkit%20Documents/Device_Enrollment.pdf for more information about the enroll application.

- From within the OSP Toolkit directory (example: /usr/src/TK-3_3_6-20060303), execute the following commands at the command prompt:

```
cd enroll
make clean; make linux
```

Compilation is successful if there are no errors in the compiler output. The enroll program is now located in the OSP Toolkit/bin directory (example: /usr/src/TK-3_3_6-20060303/bin).

2.2 Obtain Crypto Files

The OSP module in Asterisk requires three crypto files containing a local certificate (localcert.pem), private key (pkey.pem), and CA certificate (cacert_0.pem). Asterisk will try to load the files from the Asterisk public/private key directory - /var/lib/asterisk/keys. If the files are not present, the OSP module will not start and the Asterisk will not support the OSP protocol. Use the enroll.sh script from the toolkit distribution to enroll Asterisk with an OSP server and obtain the crypto files. Documentation explaining how to use the enroll.sh script (Device Enrollment) to enroll with an OSP server is available at http://www.transnexus.com/OSP%20Toolkit/OSP%20Toolkit%20Documents/Device_Enrollment.pdf. Copy the files generated by the enrollment process to the Asterisk /var/lib/asterisk/keys directory.

> ⚠ The osptestserver.transnexus.com is configured only for sending and receiving non-SSL messages, and issuing signed tokens. If you need help, post a message on the OSP mailing list at https://lists.sourceforge.net/lists/listinfo/osp-toolkit-client

The enroll.sh script takes the domain name or IP addresses of the OSP servers that the OSP Toolkit needs to enroll with as arguments, and then generates pem files – cacert_#.pem, certreq.pem, localcert.pem, and pkey.pem. The '#' in the cacert file name is used to differentiate the ca certificate file names for the various SP's (OSP servers). If only one address is provided at the command line, cacert_0.pem will be generated. If 2 addresses are provided at the command line, 2 files will be generated – cacert_0.pem and cacert_1.pem, one for each SP (OSP server). The example below shows the usage when the client is registering with osptestserver.transnexus.com.

```
./enroll.sh osptestserver.transnexus.com
Generating a 512 bit RSA private key
.......................++++++++++++
.........++++++++++++
writing new private key to 'pkey.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: _______
State or Province Name (full name) [Some-State]: _______
Locality Name (eg, city) []:_______
Organization Name (eg, company) [Internet Widgits Pty Ltd]: _______
Organizational Unit Name (eg, section) []:_______
Common Name (eg, YOUR name) []:_______
Email Address []:_______

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:_______
An optional company name []:_______

Error Code returned from openssl command : 0

CA certificate received
[SP: osptestserver.transnexus.com]Error Code returned from getcacert command : 0

output buffer after operation: operation=request
output buffer after nonce: operation=request&nonce=1655976791184458
X509 CertInfo context is null pointer
Unable to get Local Certificate
depth=0 /CN=osptestserver.transnexus.com/O=OSPServer
verify error:num=18:self signed certificate
verify return:1
depth=0 /CN=osptestserver.transnexus.com/O=OSPServer
verify return:1
The certificate request was successful.
Error Code returned from localcert command : 0
```

The files generated should be copied to the /var/lib/asterisk/keys directory.

The script enroll.sh requires AT&T korn shell (ksh) or any of its compatible variants. The /usr/src/TK-3_3_6-20060303/bin directory should be in the PATH variable. Otherwise, enroll.sh cannot find the enroll file.

3 Asterisk

In Asterisk, all OSP support is implemented as dial plan functions. In Asterisk V1.6, all combinations of routing between OSP and non-OSP enabled networks using any combination of SIP, H.323 and IAX protocols are fully supported. Section
3.1 describes the three easy steps to add OSP support to Asterisk:

1. Build Asterisk with OSP Toolkit
2. Configure osp.conf file
3. Cut and paste to extensions.conf

Sections 3.2 and 3.3 provide a detailed explanation of OSP dial plan functions and configuration examples. The detailed information provided in Sections 3.2 and 3.3 is not required for operating Asterisk with OSP, but may be helpful to developers who want to customize their Asterisk OSP implementation.

3.1 Configure for OSP Support

3.1.1 Build Asterisk with OSP Toolkit

The first step is to build Asterisk with the OSP Toolkit. If the OSP Toolkit is installed in the default install directory, /usr/local, no additional configuration is required. Compile Asterisk according to the instructions provided with the Asterisk distribution.

If the OSP Toolkit is installed in another directory, such as /myosp, Asterisk must be configured with the location of the OSP Toolkit. See the example below.

```
--with-osptk=/myosp
```

Please change the install path only if you familiar with both the OSP Toolkit and the Asterisk. Otherwise, the change may result in Asterisk not supporting the OSP protocol.

3.1.2 osp.conf

The /etc/asterisk/osp.conf file, shown below, contains configuration parameters for using OSP. Two parameters, servicepoint and source must be configured. The default values for all other parameters will work well for standard OSP implementations.

```
;
; Open Settlement Protocol Sample Configuration File
;
; This file contains configuration of OSP server providers that
; are used by the Asterisk OSP module.  The section "general" is
; reserved for global options.  All other sections describe specific
; OSP Providers.  The provider "default" is used when no provider is
; otherwise specified.
:
: The "servicepoint" and "source" parameters must be configured.  For
; most implementations the other parameters in this file can be left
; unchanged.
;
[general]
;
; Enable cryptographic acceleration hardware.
;
accelerate=no
;
; Defines the status of tokens that Asterisk will validate.
; 0 - signed tokens only
; 1 - unsigned tokens only
; 2 - both signed and unsigned
; The default value is 0, i.e. the Asterisk will only validate signed
; tokens.
;
tokenformat=0
;
[default]
;
; List all service points (OSP servers) for this provider.  Use
; either domain name or IP address.  Most OSP servers use port 5045.
;
```

```
;servicepoint=http://osptestserver.transnexus.com:5045/osp
servicepoint=http://OSP server IP:5045/osp
;
; Define the "source" device for requesting OSP authorization.
: This value is usually the domain name or IP address of the
: the Asterisk server.
;
;source=domain name or [IP address in brackets]
source=[host IP]
;
; Define path and file name of crypto files.
; The default path for crypto file is /var/lib/asterisk/keys.  If no
; path is defined, crypto files should be in
; /var/lib/asterisk/keys directory.
;
; Specify the private key file name.
; If this parameter is unspecified or not present, the default name
; will be the osp.conf section name followed by "-privatekey.pem"
; (for example: default-privatekey.pem)
;
privatekey=pkey.pem
;
; Specify the local certificate file.
; If this parameter is unspecified or not present, the default name
; will be the osp.conf section name followed by "- localcert.pem "
; (for example: default-localcert.pem)
;
localcert=localcert.pem
;
; Specify one or more Certificate Authority key file names.  If none
; are listed, a single Certificate Authority key file name is added
; with the default name of the osp.conf section name followed by
; "-cacert_0.pem " (for example: default-cacert_0.pem)
;
cacert=cacert_0.pem
;
; Configure parameters for OSP communication between Asterisk OSP
; client and OSP servers.
;
; maxconnections: Max number of simultaneous connections to the
;                 provider OSP server (default=20)
; retrydelay:     Extra delay between retries (default=0)
; retrylimit:     Max number of retries before giving up (default=2)
; timeout:        Timeout for response in milliseconds (default=500)
;
maxconnections=20
retrydelay=0
retrylimit=2
timeout=500
;
; Set the authentication policy.
; 0 - NO        - Accept all calls.
; 1 – YES       - Accept calls with valid token or no token.
;                  Block calls with invalid token.
; 2 – EXCLUSIVE – Accept calls with valid token.
;                  Block calls with invalid token or no token.
; Default is 1,
;
authpolicy=1
```

```
;
; Set the default destination protocol. The OSP module supports
; SIP, H323, and IAX protocols.  The default protocol is set to SIP.
```

```
;
defaultprotocol=SIP
```

3.1.3 extensions.conf

OSP functions are implemented as dial plan functions in the extensions.conf file. To add OSP support to your Asterisk server, simply copy and paste the text box below to your extensions.conf file. These functions will enable your Asterisk server to support all OSP call scenarios. Configuration of your Asterisk server for OSP is now complete.

```
[globals]
DIALOUT=DAHDI/1

[SrcGW] ; OSP Source Gateway
exten => _XXXX.,1,NoOp(OSPSrcGW)
; Set calling number if necessary
exten => _XXXX.,n,Set(CALLERID(numner)=1234567890)
; OSP lookup using default provider, if fail/error jump to lookup+101
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||j)
; Deal with outbound call according to protocol
exten => _XXXX.,n,Macro(outbound)
; Dial to destination, 60 timeout, with call duration limit
exten => _XXXX.,n,Dial(${OSPDIALSTR},60,oL($[${OSPOUTTIMELIMIT}*1000]))
; Wait 1 second
exten => _XXXX.,n,Wait,1
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})

[DstGW] ; OSP Destination Gateway
exten => _XXXX.,1,NoOp(OSPDstGW)
; Deal with inbound call according to protocol
exten => _XXXX.,n,Macro(inbound)
; Validate token using default provider, if fail/error jump to auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; Ringing
exten => _XXXX.,n,Ringing
; Wait 1 second
exten => _XXXX.,n,Wait,1
; Check inbound call duration limit
exten => _XXXX.,n,GoToIf($[${OSPINTIMELIMIT}=0]?100:200)
; Without duration limit
exten => _XXXX.,100,Dial(${DIALOUT},15,o)
exten => _XXXX.,n,Goto(1000)
; With duration limit
exten => _XXXX.,200,Dial(${DIALOUT},15,oL($[${OSPINTIMELIMIT}*1000]))
exten => _XXXX.,n,Goto(1000)
; Wait 1 second
exten => _XXXX.,1000,Wait,1
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
exten => h,1,NoOp()
```

```
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})

[GeneralProxy] ; Proxy
exten => _XXXX.,1,NoOp(OSP-GeneralProxy)
; Deal with inbound call according to protocol
exten => _XXXX.,n,Macro(inbound)
; Validate token using default provider, if fail/error jump to auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; OSP lookup using default provider, if fail/error jump to lookup+101
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||j)
; Deal with outbound call according to protocol
exten => _XXXX.,n,Macro(outbound)
; Dial to destination, 14 timeout, with call duration limit
exten => _XXXX.,n,Dial(${OSPDIALSTR},14,oL($[${OSPOUTTIMELIMIT}*1000]))
; OSP lookup next destination using default provider, if fail/error jump to next1+101
exten => _XXXX.,n(next1),OSPNext(${HANGUPCAUSE}||j)
; Deal with outbound call according to protocol
exten => _XXXX.,n,Macro(outbound)
; Dial to destination, 15 timeout, with call duration limit
exten => _XXXX.,n,Dial(${OSPDIALSTR},15,oL($[${OSPOUTTIMELIMIT}*1000]))
; OSP lookup next destination using default provider, if fail/error jump to next2+101
exten => _XXXX.,n(next2),OSPNext(${HANGUPCAUSE}||j)
; Deal with outbound call according to protocol
exten => _XXXX.,n,Macro(outbound)
; Dial to destination, 16 timeout, with call duration limit
exten => _XXXX.,n,Dial(${OSPDIALSTR},16,oL($[${OSPOUTTIMELIMIT}*1000]))
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
; Deal with OSPNext fail/error
exten => _XXXX.,next1+101,Hangup
; Deal with OSPNext fail/error
exten => _XXXX.,next2+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})

[macro-inbound]
exten => s,1,NoOp(inbound)
; Get inbound protocol
exten => s,n,Set(CHANTECH=${CUT(CHANNEL,/,1)})
exten => s,n,GoToIf($["${CHANTECH}"="H323"]?100)
exten => s,n,GoToIf($["${CHANTECH}"="IAX2"]?200)
exten => s,n,GoToIf($["${CHANTECH}"="SIP"]?300)
exten => s,n,GoTo(1000)
; H323 -----------------------------------------------------
; Get peer IP
exten => s,100,Set(OSPPEERIP=${H323CHANINFO(peerip)})
; Get OSP token
exten => s,n,Set(OSPINTOKEN=${H323CHANINFO(osptoken)})
exten => s,n,GoTo(1000)
; IAX ------------------------------------------------------
; Get peer IP
exten => s,200,Set(OSPPEERIP=${IAXPEER(CURRENTCHANNEL)})
; Get OSP token
```

```
exten => s,n,Set(OSPINTOKEN=${IAXCHANINFO(osptoken)})
exten => s,n,GoTo(1000)
; SIP ---------------------------------------------------------
; Get peer IP
exten => s,300,Set(OSPPEERIP=${SIPCHANINFO(peerip)})
; Get OSP token
exten => s,n,Set(OSPINTOKEN=${SIP_HEADER(P-OSP-Auth-Token)})
exten => s,n,GoTo(1000)
; -------------------------------------------------------------
exten => s,1000,MacroExit

[macro-outbound]
exten => s,1,NoOp(outbound)
; Set calling number which may be translated
exten => s,n,Set(CALLERID(num)=${OSPCALLING})
; Check destinatio protocol
exten => s,n,GoToIf($["${OSPTECH}"="H323"]?100)
exten => s,n,GoToIf($["${OSPTECH}"="IAX2"]?200)
exten => s,n,GoToIf($["${OSPTECH}"="SIP"]?300)
; Something wrong
exten => s,n,Hangup
exten => s,n,GoTo(1000)
; H323 -------------------------------------------------------
; Set call id
exten => s,100,Set(H323CHANINFO(callid)=${OSPOUTCALLID})
; Set OSP token
exten => s,n,Set(H323CHANINFO(osptoken)=${OSPOUTTOKEN})
exten => s,n,GoTo(1000)
; IAX ---------------------------------------------------------
; Set OSP token
exten => s,200,Set(IAXCHANINFO(osptoken)=${OSPOUTTOKEN})
exten => s,n,GoTo(1000)
; SIP ---------------------------------------------------------
exten => s,300,GoTo(1000)
```

```
; --------------------------------------------------------------
exten => s,1000,MacroExit
```

3.1.4 dahdi/sip/iax/h323/ooh323.conf

There is no configuration required for OSP.

3.2 OSP Dial Plan Functions

This section provides a description of each OSP dial plan function.

3.2.1 OSPAuth

OSP token validation function.
Input:

- OSPPEERIP: last hop IP address
- OSPINTOKEN: inbound OSP token
- provider: OSP service provider configured in osp.conf. If it is empty, default provider is used.
- priority jump

Output:

- OSPINHANDLE: inbound OSP transaction handle
- OSPINTIMELIMIT: inbound call duration limit
- OSPAUTHSTATUS: OSPAuth return value. SUCCESS/FAILED/ERROR

3.2.2 OSPLookup

OSP lookup function.

Input:

- OSPPEERIP: last hop IP address
- OSPINHANDLE: inbound OSP transaction handle
- OSPINTIMELIMIT: inbound call duration limit
- exten: called number
- provider: OSP service provider configured in osp.conf. If it is empty, default provider is used.
- priority jump
- callidtypes: Generate call ID for the outbound call. h: H.323; s: SIP; i: IAX. Only h, H.323, has been implemented.

Output:

- OSPOUTHANDLE: outbound transaction handle
- OSPTECH: outbound protocol
- OSPDEST: outbound destination IP address
- OSPCALLED: outbound called nummber
- OSPCALLING: outbound calling number
- OSPOUTTOKEN: outbound OSP token
- OSPRESULTS: number of remaining destinations
- OSPOUTTIMELIMIT: outbound call duration limit
- OSPOUTCALLIDTYPES: same as input callidtypes
- OSPOUTCALLID: outbound call ID. Only for H.323
- OSPDIALSTR: outbound dial string
- OSPLOOKUPSTATUS: OSPLookup return value. SUCCESS/FAILED/ERROR

3.2.3 OSPNext

OSP lookup next function.

Input:

- OSPINHANDLE: inbound transaction handle
- OSPOUTHANDLE: outbound transaction handle
- OSPINTIMELIMIT: inbound call duration limit
- OSPOUTCALLIDTYPES: types of call ID generated by Asterisk.
- OSPRESULTS: number of remain destinations
- cause: last destination disconnect cause
- priority jump

Output:

- OSPTECH: outbound protocol
- OSPDEST: outbound destination IP address
- OSPCALLED: outbound called number
- OSPCALLING: outbound calling number
- OSPOUTTOKEN: outbound OSP token
- OSPRESULTS: number of remain destinations
- OSPOUTTIMELIMIT: outbound call duration limit
- OSPOUTCALLID: outbound call ID. Only for H.323
- OSPDIALSTR: outbound dial string
- OSPNEXTSTATUS: OSPLookup return value. SUCCESS/FAILED/ERROR

3.2.4 OSPFinish

OSP report usage function.

Input:

- OSPINHANDLE: inbound transaction handle
- OSPOUTHANDLE: outbound transaction handle
- OSPAUTHSTATUS: OSPAuth return value
- OSPLOOKUPTSTATUS: OSPLookup return value
- OSPNEXTSTATUS: OSPNext return value
- cause: last destination disconnect cause
- priority jump

Output:

- OSPFINISHSTATUS: OSPLookup return value. SUCCESS/FAILED/ERROR

3.3 extensions.conf Examples

The extensions.conf file example provided in Section 3.1 is designed to handle all OSP call scenarios when Asterisk is used as a source or destination gateway to the PSTN or as a proxy between VoIP networks. The extenstion.conf examples in this section are designed for specific use cases only.

3.3.1 Source Gateway

The examples in this section apply when the Asterisk server is being used as a TDM to VoIP gateway. Calls originate on the TDM network and are converted to VoIP by Asterisk. In these cases, the Asterisk server queries an OSP server to find a route to a VoIP destination. When the call ends, Asterisk sends a CDR to the OSP server.
For SIP protocol.

```
[SIPSrcGW]
exten => _XXXX.,1,NoOp(SIPSrcGW)
; Set calling number if necessary
exten => _XXXX.,n,Set(CALLERID(numner)=CallingNumber)
; OSP lookup using default provider, if fail/error jump to lookup+101
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||j)
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 60 timeout, with call duration limit
exten => _XXXX.,n,Dial(${OSPDIALSTR},60,oL($[${OSPOUTTIMELIMIT}*1000]))
; Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

For IAX protocol.

```
[IAXSrcGW]
exten => _XXXX.,1,NoOp(IAXSrcGW)
; Set calling number if necessary
exten => _XXXX.,n,Set(CALLERID(numner)=CallingNumber)
; OSP lookup using default provider, if fail/error jump to lookup+101
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||j)
; Set outbound OSP token
exten => _XXXX.,n,Set(IAXCHANINFO(osptoken)=${OSPOUTTOKEN})
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 60 timeout, with call duration limit
exten => _XXXX.,n,Dial(${OSPDIALSTR},60,oL($[${OSPOUTTIMELIMIT}*1000]))
; Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

For H.323 protocol.

```
[H323SrcGW]
exten => _XXXX.,1,NoOp(H323SrcGW)
; Set calling number if necessary
exten => _XXXX.,n,Set(CALLERID(numner)=CallingNumber)
; OSP lookup using default provider, if fail/error jump to lookup+101
; “h” parameter is used to generate a call id
; Cisco OSP gateways use this call id to validate OSP token
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||jh)
; Set outbound call id
exten => _XXXX.,n,Set(OH323CHANINFO(callid)=${OSPOUTCALLID})
; Set outbound OSP token
exten => _XXXX.,n,Set(OH323CHANINFO(osptoken)=${OSPOUTTOKEN})
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 60 timeout, with call duration limit
exten => _XXXX.,n,Dial(${OSPDIALSTR},60,oL($[${OSPOUTTIMELIMIT}*1000]))
; Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

3.3.2 Destination Gateway

The examples in this section apply when Asterisk is being used as a VoIP to TDM gateway. VoIP calls are received by Asterisk which validates the OSP peering token and completes to the TDM network. After the call ends, Asterisk sends a CDR to the OSP server.

For SIP protocol

```
[SIPDstGW]
exten => _XXXX.,1,NoOp(SIPDstGW)
; Get peer IP
exten => _XXXX.,n,Set(OSPPEERIP=${SIPCHANINFO(peerip)})
; Get OSP token
exten => _XXXX.,n,Set(OSPINTOKEN=${SIP_HEADER(P-OSP-Auth-Token)})
; Validate token using default provider, if fail/error jump to auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; Ringing
exten => _XXXX.,n,Ringing
; Wait 1 second
exten => _XXXX.,n,Wait,1
; Dial phone, timeout 15 seconds, with call duration limit
exten => _XXXX.,n,Dial(${DIALOUTANALOG}/${EXTEN:1},15,oL($[${OSPINTIMELIMIT}*1000]))
; Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

For IAX protocol

```
[IAXDstGW]
exten => _XXXX.,1,NoOp(IAXDstGW)
; Get peer IP
exten => _XXXX.,n,Set(OSPPEERIP=${IAXPEER(CURRENTCHANNEL)})
; Get OSP token
exten => _XXXX.,n,Set(OSPINTOKEN=${IAXCHANINFO(osptoken)})
; Validate token using default provider, if fail/error jump to auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; Ringing
exten => _XXXX.,n,Ringing
; Wait 1 second
exten => _XXXX.,n,Wait,1
; Dial phone, timeout 15 seconds, with call duration limit
exten => _XXXX.,n,Dial(${DIALOUTANALOG}/${EXTEN:1},15,oL($[${OSPINTIMELIMIT}*1000]))
; Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

For H.323 protocol

```
[H323DstGW]
exten => _XXXX.,1,NoOp(H323DstGW)
; Get peer IP
exten => _XXXX.,n,Set(OSPPEERIP=${H323CHANINFO(peerip)})
; Get OSP token
exten => _XXXX.,n,Set(OSPINTOKEN=${H323CHANINFO(osptoken)})
; Validate token using default provider, if fail/error jump to auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; Ringing
exten => _XXXX.,n,Ringing
; Wait 1 second
exten => _XXXX.,n,Wait,1
; Dial phone, timeout 15 seconds, with call duration limit
exten => _XXXX.,n,Dial(${DIALOUTANALOG}/${EXTEN:1},15,oL($[${OSPINTIMELIMIT}*1000]))
; Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

3.3.3 Proxy

The example in this section applies when Asterisk is a proxy between two VoIP networks.

```
[GeneralProxy]
exten => _XXXX.,1,NoOp(GeneralProxy)
; Get peer IP and inbound OSP token
; SIP, un-comment the following two lines.
;exten => _XXXX.,n,Set(OSPPEERIP=${SIPCHANINFO(peerip)})
;exten => _XXXX.,n,Set(OSPINTOKEN=${SIP_HEADER(P-OSP-Auth-Token)})
; IAX, un-comment the following 2 lines
;exten => _XXXX.,n,Set(OSPPEERIP=${IAXPEER(CURRENTCHANNEL)})
;exten => _XXXX.,n,Set(OSPINTOKEN=${IAXCHANINFO(osptoken)})
; H323, un-comment the following two lines.
;exten => _XXXX.,n,Set(OSPPEERIP=${OH323CHANINFO(peerip)})
;exten => _XXXX.,n,Set(OSPINTOKEN=${OH323CHANINFO(osptoken)})
;----------------------------------------------------------------
; Validate token using default provider, if fail/error jump to auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; OSP lookup using default provider, if fail/error jump to lookup+101
; “h” parameter is used to generate a call id for H.323 destinations
; Cisco OSP gateways use this call id to validate OSP token
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||jh)
; Set outbound call id and OSP token
; IAX, un-comment the following line.
;exten => _XXXX.,n,Set(IAXCHANINFO(osptoken)=${OSPOUTTOKEN})
; H323, un-comment the following two lines.
;exten => _XXXX.,n,Set(OH323CHANINFO(callid)=${OSPOUTCALLID})
;exten => _XXXX.,n,Set(OH323CHANINFO(osptoken)=${OSPOUTTOKEN})
;----------------------------------------------------------------
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 14 timeout, with call duration limit
exten => _XXXX.,n,Dial(${OSPDIALSTR},14,oL($[${OSPOUTTIMELIMIT}*1000]))
; OSP lookup next destination using default provider, if fail/error jump to next1+101
exten => _XXXX.,n(next1),OSPNext(${HANGUPCAUSE}||j)
; Set outbound call id and OSP token
; IAX, un-comment the following line.
;exten => _XXXX.,n,Set(IAXCHANINFO(osptoken)=${OSPOUTTOKEN})
; H323, un-comment the following two lines.
;exten => _XXXX.,n,Set(OH323CHANINFO(callid)=${OSPOUTCALLID})
;exten => _XXXX.,n,Set(OH323CHANINFO(osptoken)=${OSPOUTTOKEN})
;----------------------------------------------------------------
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 15 timeout, with call duration limit
exten => _XXXX.,n,Dial(${OSPDIALSTR},15,oL($[${OSPOUTTIMELIMIT}*1000]))
; OSP lookup next destination using default provider, if fail/error jump to next2+101
exten => _XXXX.,n(next2),OSPNext(${HANGUPCAUSE}||j)
; Set outbound call id and OSP token
; IAX, un-comment the following line.
;exten => _XXXX.,n,Set(IAXCHANINFO(osptoken)=${OSPOUTTOKEN})
; H323, un-comment the following two lines.
;exten => _XXXX.,n,Set(OH323CHANINFO(callid)=${OSPOUTCALLID})
;exten => _XXXX.,n,Set(OH323CHANINFO(osptoken)=${OSPOUTTOKEN})
;----------------------------------------------------------------
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 16 timeout, with call duration limit
exten => _XXXX.,n,Dial(${OSPDIALSTR},16,oL($[${OSPOUTTIMELIMIT}*1000]))
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
```

```
exten => _XXXX.,auth+101,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
; Deal with 1st OSPNext fail/error
exten => _XXXX.,next1+101,Hangup
; Deal with 2nd OSPNext fail/error
exten => _XXXX.,next2+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

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

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, please use the asterisk-ss7 or asterisk-dev mailing lists (I monitor them regularly) or email me directly.

Matthew Fredrickson
creslin@digium.com

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 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. 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.

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_ooh323 (Asterisk-Addons)
- chan_gtalk
- chan_jingle
- chan_motif (Asterisk 11+)

Configuration

To set a desired packetization interval on a specific codec, append that inteval 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 prefered 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)
g723	30	300	30	30
gsm	20	300	20	20
ulaw	10	150	20	10
alaw	10	150	20	10
g726	10	300	20	10
ADPCM	10	300	20	10
SLIN	10	70	20	10
lpc10	20	20	20	20
g729	10	230	20	10
speex	10	60	20	10
ilbc	30	30	30	30

g726_aal2	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 specificed framing does not meet the increment requirement, the specified framing is rounded down to the closest valid framing options.

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,(useunavil),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.
;
;
;smdiport=/dev/ttyS0
;2565551234=1234@vmcontext1
;2565555678=5678@vmcontext2
;smdiport=/dev/ttyS1
;2565559999=9999
```

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 Dialled 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
```

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

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.
- tlscafile=</path/to/certificate> - If the server your 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 tlscafile 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

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
tlscafile=/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
```

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 then 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.

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,
 cancallforward CHAR(3)    DEFAULT 'yes',
 directmedia CHAR(3)    DEFAULT 'yes',
 context  VARCHAR(80)   DEFAULT NULL,
 defaultip 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,
 restrictcid 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]
```

- Setup 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 voicemessages 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 voicemessages table via the isql utility.

```
[jsmith2@localhost ODBC]$ echo "SELECT uniqueid, msgnum, dir, duration FROM voicemessages 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
odbctable=voicemessages
```

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:

```
[linksys]
type=friend
secret=verysecret
disallow=all
allow=ulaw
allow=gsm
context=odbc
host=dynamic
qualify=yes
```

I can check to see that my linksys phone is registered with Asterisk correctly:

```
localhost*CLI> sip show peers like linksys
Name/username              Host            Dyn Nat ACL Port     Status
linksys/linksys            192.168.0.103    D          5060     OK (9 ms)
1 sip peers [1 online , 0 offline]
```

- At last, we're finally ready to leave a voicemail message and have it stored in our database! (Who'd have guessed it would be this much trouble?!?) Pick up the phone, dial extension 100, and leave yourself a voicemail message.
  In my case, this is what appeared on the Asterisk CLI:

```
localhost*CLI>
    -- Executing VoiceMail("SIP/linksys-10228cac", "101@odbctest") in new stack
    -- Playing 'vm-intro' (language 'en')
    -- Playing 'beep' (language 'en')
    -- Recording the message
    -- x=0, open writing:  /var/spool/asterisk/voicemail/odbctest/101/tmp/dlZunm format: gsm, 0x101f6534
    -- User ended message by pressing #
    -- Playing 'auth-thankyou' (language 'en')
  == Parsing '/var/spool/asterisk/voicemail/odbctest/101/INBOX/msg0000.txt': Found
```

Now, we can check the database and make sure the record actually made it into PostgreSQL, from within the psql utility.

```
[jsmith2@localhost ~]$ psql
Password:
Welcome to psql 8.1.4, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

asterisk=# SELECT * FROM voicemessages;
 uniqueid | msgnum |                       dir                        | context | macrocontext |       callerid      |
origtime  | duration | mailboxuser | mailboxcontext | recording | label | read | sip_id | pabx_id | iax_id
----------+--------+--------------------------------------------------+---------+--------------+---------------------+---------
---+----------+-------------+----------------+-----------+-------+------+--------+---------+--------
       26 |      0 | /var/spool/asterisk/voicemail/odbctest/101/INBOX | odbc    |              | "linksys" <linksys> | 1167794179
| 7        | 101         | odbctest       | 16599     |       | f    |        |         |
(1 row)
```

Did you notice the the recording column is just a number? When a recording gets stuck in the database, the audio isn't actually stored in the voicemessages table. It's stored in a system table called the large object table. We can look in the large object table and verify that the object actually exists there:

```
asterisk=# \lo_list
    Large objects
  ID   | Description
-------+-------------
 16599 |
(1 row)
```

In my case, the OID is 16599. Your OID will almost surely be different. Just make sure the OID number in the recording column in the voicemessages table corresponds with a record in the large object table. (The trigger we added to
our voicemessages table was designed to make sure this is always the case.)

We can also pull a copy of the voicemail message back out of the database and write it to a file, to help us as we debug things:

```
asterisk=# \lo_export 16599 /tmp/odcb-16599.gsm
lo_export
```

We can even listen to the file from the Linux command line:

```
[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)

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.

Using the Hoard Memory Allocator with Asterisk

Using the Hoard Memory Allocator with Asterisk

Install the Hoard Memory Allocator

Download Hoard from http://www.hoard.org/ either via a package or the source tarball.

If downloading the source, unpack the tarball and follow the instructions in the README file to build libhoard for your platform.

Configure asterisk

Run ./configure in the root of the asterisk source directory, passing the **--with-hoard** option specifying the location of the libhoard shared library.

For example:

```
./configure --with-hoard=/usr/src/hoard-371/src/
```

Note that we don't specify the full path to libhoard.so, just the directory where it resides.

Enable Hoard in menuselect

Run 'make menuselect' in the root of the asterisk source distribution. Under 'Compiler Flags' select the 'USE_HOARD_ALLOCATOR' option. If the option is not available (shows up with XXX next to it) this means that configure was not able to find libhoard.so. Check that the path you passed to the **--with-hoard** option is correct. Re-run **./configure** with the correct option and then repeat this step.

Make and install asterisk

Run the standard build commands:

```
# make
# make install
```

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
- libswcale-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

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.

```
 ________________________________________________________________
     |  ______   ______   ______   ______   ______   ______   ______  |
     | | tn.1 | | tn.2 | | tn.3 | | tn.4 | | tn.5 | | tn.6 | | tn.7 | |
     | |______| |______| |______| |______| |______| |______| |______| |
     |  ______   ______   ______   ______   ______   ______   ______  |
     | |______| |______| |______| |______| |______| |______| |______| |
     |  _________________    __________________    _________________  |
     | |                 |  |                  |  |                 | |
     | |                 |  |                  |  |                 | |
     | |                 |  |                  |  |                 | |
     | |   remote video  |  |                  |  |   local video   | |
     | |                 |  |                  |  |          ______ | |
     | |                 |  |      keypad      |  |         |  PIP || |
     | |                 |  |                  |  |         |______|| |
     | |_________________|  |                  |  |_________________| |
     |                      |                  |                      |
     |                      |                  |                      |
     |                      |__________________|                      |
     |________________________________________________________________|
```

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, ...)

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!

Lua Dialplan Configuration

Asterisk supports the ability to write dialplan instructions in the Lua programming language. This method can be used as an alternative to or in combination with extensions.conf and/or AEL. PBX lua allows users to use the full power of lua to develop telephony applications using Asterisk. Lua dialplan configuration is done in the `extensions.lua` file.

Dependencies
To use pbx_lua, the lua development libraries must be installed before Asterisk is configured and built. You can get these libraries directly from http://lua.org, but it is easier to install them using your distribution's package management tool. The package is probably named liblua5.1-dev, liblua-dev, or lua-devel depending on your linux distribution.

PBX Lua Basics

The `extensions.lua` file is used to configure PBX lua and is a lua script (as opposed to being a standard asterisk configuration file). Any thing that is proper lua code is allowed in this file. Asterisk expects to find a global table named '`extensions`' when the file is loaded. This table can be generated however you wish. The simplest way is to define all of the extensions in line, but for more complex dialplans alternative methods may be necessary.

Each extension is a lua function that is executed when a channel lands on that extension. The extension function is passed the current context and extension as the first two arguments. These can be safely ignored if desired. There are no priorities (each extension function is treated as priority 1 by the rest of Asterisk). Patterns are allowed just as in `extensions.conf` and the matching order is identical.

extensions.lua

```
extensions = {
   default = {
      ["100"] = function(context, extension)
         app.playback("please-hold")
         app.dial("SIP/100", 60)
      end;

      ["101"] = function(c, e)
         app.dial("SIP/101", 60)
      end;
}
```

The `extensions.lua` file can be reloaded by reloading the pbx_lua module.

```
*CLI> module reload pbx_lua
```

If there are errors in the file, the errors will be reported and the existing extensions.lua file will remain in use. Channels that existed before the reload command was issued will also continue to use the existing extensions.lua file.

Runtime errors are logged and the channel on which the error occurred is hung up.

Dialplan to Lua Reference

Below is a quick reference that can be used to translate traditional `extensions.conf` dialplan concepts to their analog in `extensions.lua`.

Extension Patterns

Extension pattern matching syntax on logic works the same for `extensions.conf` and `extensions.lua`.

extensions.conf

```
[users]
exten =>
_1XX,1,Dial(SIP/${EXTEN})

exten =>
_2XX,1,Voicemail(${EXTEN:1})
```

extensions.lua

```
extensions = {}
extensions.users = {}

extensions.users["_1XX"] =
function(c, e)
   app.dial("SIP/" .. e)
end

extensions.users["_2XX"] =
function(c, e)
  app.voicemail("1" .. e:sub(2))
end
```

Context Includes

extensions.conf

```
[users]
exten => 100,1,Noop
exten => 100,n,Dial("SIP/100")

[demo]
exten => s,1,Noop
exten =>
s,n,Playback(demo-congrats)

[default]
include => demo
include => users
```

extensions.lua

```
extensions = {
   users = {
      [100] = function()
         app.dial("SIP/100")
      end;
   };

   demo = {
      ["s"] = function()

app.playback(demo-congrats)
      end;
   };

   default = {
      include = {"demo",
"users"};
   };
}
```

Loops

extensions.conf

```
exten => 100,1,Noop
exten => 100,n,Set(i=0)
exten => 100,n,While($[i < 10])
exten => 100,n,Verbose(i = ${i})
exten => 100,n,EndWhile
```

extensions.lua

```
i = 0
while i < 10 do
  app.verbose("i = " .. i)
end
```

Variables

extensions.conf

```
exten =>
100,1,Set(my_variable=my_value)
exten =>
100,n,Verbose(my_variable =
${my_variable})
```

extensions.lua

```
channel.my_variable = "my_value"
app.verbose("my_variable = " ..
channel.my_variable:get())
```

Applications

extensions.conf

```
exten =>
100,1,Dial("SIP/100",,m)
```

extensions.lua

```
app.dial("SIP/100", nil, "m")
```

Macros/GoSub

Macros can be defined in pbx_lua by naming a context 'macro-' just as in `extensions.conf`, but generally where you would use macros or gosub in `extensions.conf` you would simply use a function in lua.*

extensions.conf

```
[macro-dial]
exten => s,1,Noop
exten => s,n,Dial(${ARG1})

[default]
exten =>
100,1,Macro(dial,SIP/100)
```

extensions.lua

```
extensions = {}
extensions.default = {}

function dial(resource)
   app.dial(resource)
end

extensions.default[100] =
function()
   dial("SIP/100")
end
```

Goto

While `Goto` is an extenstions.conf staple, it should generally be avoided in pbx_lua in favor of functions.

extensions.conf

```
[default]
exten => 100,1,Goto(102,1)

exten =>
102,1,Playback("demo-thanks")
exten => 102,n,Hangup
```

extensions.lua

```
extensions = {}
extensions.default = {}

function do_hangup()
   app.playback("demo-thanks")
   app.hangup()
end

extensions.default[100] =
function()
   do_hangup()
end
```

The `app.goto()` function will not work as expected in pbx_lua in Asterisk 1.8. If you must use `app.goto()` you must manually return control back to asterisk using `return` from the dialplan extension function, otherwise execution will continue after the call to `app.goto()`. Calls to `app.goto()` should work as expected in Asterisk 10 but still should not be necessary in most cases.

In Asterisk 1.8, use return

```
function extension_function(c, e)
   return app.goto("default", "100", 1)

   -- without that 'return' the rest of the function would execute normally
   app.verbose("Did you forget to use 'return'?")
end
```

Interacting with Asterisk from Lua (apps, variables, and functions)

Interaction with is done through a series of predefined objects provided by pbx_lua. The `app` table is used to access dialplan applications. Any asterisk application can be accessed and executed as if it were a function attached to the `app` table. Dialplan variables and functions are accessed and executed via the `channel` table.

Dialplan Applications

extensions.lua

```
app.playback("please-hold")
app.dial("SIP/100", nil, "m")
```

Any dialplan application can be executed using the `app` table. Application names are case insensitive. Arguments are passed to dialplan applications just as arguments are passed to functions in lua. String arguments must be quoted as they are lua strings. Empty arguments may be passed as `nil` or as empty strings.

Channel Variables

Set a Variable

```
channel.my_variable = "my_value"
```

After this the channel variable `${my_variable}` contains the value "my_value".

Read a Variable

```
value = channel.my_variable:get()
```

Any channel variable can be read and set using the `channel` table. Local and global lua variables can be used as they normally would and are completely unrelated to channel variables.

The following construct will NOT work.

```
value = channel.my_variable -- does not work as expected (value:get() could be used
to get the value after this line)
```

If the variable name contains characters that lua considers special use the `[]` operator to access them.

```
channel["my_variable"] = "my_value"
value = channel["my_variable"]:get()
```

Dialplan Functions

Write a Dialplan Function

```
channel.FAXOPT("modems"):set("v17,v27,v29")
```

Read a Dialplan Function

```
value = channel.FAXOPT("modems"):get()
```

Note the use of the `:` operator with the `get()` and `set()` methods.

If the function name contains characters that lua considers special use the `[]` operator to access them.

```
channel["FAXOPT(modems)"] = "v17,v27,v29"
value = channel["FAXOPT(modems)"]:get()
```

The following constructs will NOT work.

```
channel.FAXOPT("modems") = "v17,v27,v29" -- syntax error
value = channel.FAXOPT("modems")         -- does not work as expected (value:get()
could be used to get the value after this line)
```

Dialplan function names are case sensitive.

Lua Dialplan Tips and Tricks

Long Running Operations (Autoservcie)

Before starting long running operations, an autoservice should be started using the `autoservice_start()` function. An autoservice will ensure that the user hears a continuous stream of audio while your lua code works in the background. This autoservice will automatically be stopped before executing applications and dialplan functions and will be restarted afterwards. The autoservice can be stopped using autoservice_stop() and the autoservice_status() function will return `true` if an autoservice is currently running.

```
app.startmusiconhold()

autoservice_start()
do_expensive_db_query()
autoservice_stop()

app.stopmusiconhold()
```

In Asterisk 10 an autoservice is automatically started for you by default.

Defining Extensions Dynamically

Since extensions are functions in pbx_lua, any function can be used, including closures. A function can be defined that returns extension functions and used to populate the extensions table.

extensions.lua

```
extensions = {}
extensions.default = {}

function sip_exten(e)
   return function()
      app.dial("SIP/" .. e)
   end
end

extensions.default[100] = sip_exten(100)
extensions.default[101] = sip_exten(101)
```

Creating Custom Aliases for Built-in Constructs

If you don't like the app table being named 'app' or if you think typing 'channel' to access the `channel` table is too much work, you can rename them.

I prefer less typing

```
function my_exten(context, extensions)
   c = channel
   a = app

   c.my_variable = "my new channel variable"
   a.dial("SIP/100")
end
```

Re-purposing The `print` Function

Lua has a built in "print" function that outputs things to stdout, but for Asterisk, we would rather have the output go in the verbose log. To do so, we could rewrite the `print` function as follows.

```
function print(...)
   local msg = ""
   for i=1,select('#', ...) do
      if i == 1 then
         msg = msg .. tostring(select(i, ...))
      else
         msg = msg .. "\t" .. tostring(select(i, ...))
      end
   end

   app.verbose(msg)
end
```

Splitting Configuration into Multiple Files

The `require` method can be used to load lua modules or additional files.

Using External Modules

Lua modules can be loaded using the standard `require` lua method. Some of the functionality provided by various lua modules is already included in Asterisk (e.g. func_odbc provides what LuaSQL provides). It is generally better to use code built-in to Asterisk over external lua modules. Specifically, the func_odbc module uses a connection pool to provide database resources, where as with LuaSQL each channel would have to make a new connection to the database on its own.

Compile extensions.lua

The `luac` program can be used to compile your `extensions.lua` file into lua bytecode. This will slightly increase performance as pbx_lua will no longer need to parse `extensions.lua` on load. The `luac` compiler will also detect and report any syntax errors. To use `luac`, rename your `extensions.lua` f ile and then run `luac` as follows.

Assume you name your extensions.lua file extensions.lua.lua

```
luac -o extensions.lua extensions.lua.lua
```

The pbx_lua module automatically knows the difference between a lua text file and a lua bytecode file.

Lua Dialplan Hints

In Asterisk 10 dialplan hints can be specified in `extensions.lua` in a manner similar to the way extensions are specified.

extensions.lua

```
hints = {
   default = {
      ["100"] = "SIP/100";
   };

   office = {
      ["500"] = "SIP/500";
   };

   home = {
      ["200"] = "SIP/200";
      ["201"] = "SIP/201";
   };
}
```

Lua Dialplan Examples

Some example `extensions.lua` files can be found below. They demonstrate various ways to organize extensions.

Less Clutter

Instead of defining every extension inline, you can use this method to create a neater `extensions.lua` file. Since the extensions table and each context are both normal lua tables, you can treat them as such and build them piece by piece.

extensions.lua

```
-- this function serves as an extension function directly
function call_user(c, user)
   app.dial("SIP/" .. user, 60)
end

-- this function returns an extension function
function call_sales_queue(queue)
  return function(c, e)
      app.queue(queue)
   end
end

e = {}

e.default = {}
e.default.include = {"users", "sales"}

e.users = {}
e.users["100"] = call_user
e.users["101"] = call_user

e.sales = {}
e.sales["5000"] = call_sales_queue("sales1")
e.sales["6000"] = call_sales_queue("sales2")

extensions = e
```

Less Clutter v2

In this example, we use a fancy function to register extensions.

extensions.lua

```
function register(context, extension, func)
   if not extensions then
      extensions = {}
   end

   if not extensions[context] then
      extensions[context] = {}
   end

   extensions[context][extension] = func
end

function include(context, included_context)
   if not extensions then
      extensions = {}
   end

   if not extensions[context] then
      extensions[context] = {}
   end

   if not extensions[context].include then
      extensions[context].include = {}
   end

   table.insert(extensions[context].include, included_context)
end

-- this function serves as an extension function directly
function call_user(c, user)
   app.dial("SIP/" .. user, 60)
end

-- this function returns an extension function
function call_sales_queue(queue)
  return function(c, e)
      app.queue(queue)
   end
end

include("default", "users")
include("default", "sales")

register("users", "100", call_user)
register("users", "101", call_user)

register("sales", "5000", call_sales_queue("sales1"))
register("sales", "6000", call_sales_queue("sales2"))
register("sales", "7000", function()
   app.queue("sales3")
end)
```

Advanced pbx_lua Topics

Behind the scenes, a number of things happen to make the integration of lua into Asterisk as seamless as possible. Some details of how this integration works can be found below.

`extensions.lua` Load Process

The `extensions.lua` file is loaded into memory once when the pbx_lua module is loaded or reloaded. The file is then read from memory and executed once for each channel that looks up or executes a lua based extension. Since the file is executed once for each channel, it may not be wise to do things like connect to external services directly from the main script or build your extensions table from a webservice or database.

This is probably a bad idea.

```
-- my fancy extensions.lua

extensions = {}
extensions.default = {}

-- might be a bad idea, this will run each time a channel is created
data = query_webservice_for_extensions_list("site1")

for _, e in ipairs(data) do
   extensions.default[e.exten] = function()
      app.dial("SIP/" .. e.sip_peer, e.dial_timeout)
   end
end
```

The `extensions` Table

The `extensions` table is a standard lua table and can be defined however you like. The pbx_lua module loads and sorts the table when it is needed. The keys in the table are context names and each value is another lua table containing extensions. Each key in the context table is an extension name and each value is an extension function.

```
extensions = {
   context_table = {
      extension1 = function()
      end;
      extension2 = function()
      end;
   };
}
```

Where did the priorities go?

There are no priorities. Asterisk uses priorities to define the order in which dialplan operations occur. The pbx_lua module uses functions to define extensions and execution occurs within the lua interpreter, priorities don't make sense in this context. To Asterisk, each pbx_lua extension appears as an extension with one priority. Lua extensions can be referenced using the context name, extension, and priority 1, e.g. `Goto(default,1234,1)`. You would only reference extensions this way from outside of pbx_lua (i.e. from `extensions.conf` or `extensions.ael`). From with in pbx_lua you can just execute that extension's function.

```
extensions.default["1234"]("default", "1234")
```

Lua Script Lifetime

The same lua state is used for the lifetime of the Asterisk channel it is running on, so effectively, the script has the lifetime of the channel. This means you can set global variables in the lua state and retrieve them later from a different extension if necessary.

Apps, Functions, and Variables

Details on accessing dialplan applications and functions and channel variables can be found in the Interacting with Asterisk from Lua (apps, variables, and functions) page.

When accessing a dialplan application or function or a channel variable, a placeholder object is generated that provides the `:get()` and `:set()` methods.

channel variable: var is the placeholder object

```
var = channel.my_variable
var:set("my value")
value = var:get("my value")
```

dialplan function: fax_modems is the placeholder object

```
fax_modems = channel.FAXOPT("module")

-- the function arguments are stored in the placeholder

fax_modems:set("v17")
value = fax_modems:get()
```

dialplan application: dial is the placeholder object

```
dial = app.dial

-- the only thing we can do with it is execute it
dial("SIP/100")
```

There is a small cost in creating the placeholder objects so storing frequently used placeholder objects can be used as a micro optimization. This should never be necessary though and only provides benefits if you are running micro benchmarks.

Manipulating Party ID Information

Introduction

This chapter aims to explain how to use some of the features available to manipulate party ID information. It will not delve into specific channel configuration options described in the respective sample configuration files. The party ID information can consist of Caller ID, Connected Line ID, redirecting to party ID information, and redirecting from party ID information. Meticulous control is needed particularly when interoperating between different channel technologies.

- Caller ID: The Caller ID information describes who is originating a call.
- Connected Line ID: The Connected Line ID information describes who is connected to the other end of a call while a call is established. Unlike Caller ID, the connected line information can change over the life of a call when call transfers are performed. The connected line information can also change in either direction because either end could transfer the call. For ISDN it is known as Connected Line Identification Presentation (COLP), Connected Line Identification Restriction (COLR), and Explicit Call Transfer (ECT). For SIP it is known either as P-Asserted-Identity or Remote-Party-Id.
- Redirecting information: When a call is forwarded, the call originator is informed that the call is redirecting-to a new destination. The new destination is also informed that the incoming call is redirecting-from the forwarding party. A call can be forwarded repeatedly until a new destination answers it or a forwarding limit is reached.

Tools available

Asterisk contains several tools for manipulating the party ID information for a call. Additional information can be found by using the 'core show function' or 'core show application' console commands at the Asterisk CLI. The following list identifies some of the more common tools for manipulating the party ID information:

- `CALLERID(datatype,caller-id)`
- `CONNECTEDLINE(datatype,i)`
- `REDIRECTING(datatype,i)`
- `Dial()` and `Queue()` dialplan application 'I' option
- Interception macros
- Channel driver specific configuration options.

CALLERID dialplan function

The CALLERID function has been around for quite a while and its use is straightforward. It is used to examine and alter the caller information that came into the dialplan with the call. Then the call with it's caller information passes on to the destination using the Dial() or Queue() application.

The CALLERID information is passed during the initial call setup. However, depending on the channel technology, the caller name may be delayed. Q.SIG is an example where the caller name may be delayed so your dialplan may need to wait for it.

CONNECTEDLINE dialplan function

The CONNECTEDLINE function does the opposite of the CALLERID function. CONNECTEDLINE can be used to setup connected line information to be sent when the call is answered. You can use it to send new connected line information to the remote party on the channel when a call is transferred. The CONNECTEDLINE information is passed when the call is answered and when the call is transferred.

It is up to the channel technology to determine when to act upon connected line updates before the call is answered. ISDN will just store the updated information until the call is answered. SIP will immediately update the caller with a Re-INVITE.

Since the connected line information can be sent while a call is connected, you may need to prevent the channel driver from acting on a **partial** update. The 'i' option is used to inhibit the channel driver from sending the changed information immediately.

REDIRECTING dialplan function

The REDIRECTING function allows you to report information about forwarded/deflected calls to the caller and to the new destination. The use of the REDIRECTING function is the most complicated of the party information functions.

The REDIRECTING information is passed during the initial call setup and while the call is being routed through the network. Since the redirecting information is sent before a call is answered, you need to prevent the channel driver from acting on a partial update. The 'i' option is used to inhibit the channel driver from sending the changed information immediately.

The incoming call may have already been redirected. An incoming call has already been redirected if the REDIRECTING(count) is not zero. (Alternate indications are if the REDIRECTING(from-num-valid) is non-zero or if the REDIRECTING(from-num) is not empty.)

There are several things to do when a call is forwarded by the dialplan:

- Setup the REDIRECTING(to-xxx) values to be sent to the caller.
- Setup the REDIRECTING(from-xxx) values to be sent to the new destination.
- Increment the REDIRECTING(count).
- Set the REDIRECTING(reason).
- Dial() the new destination.

Special REDIRECTING considerations for ISDN

Special considerations for Q.SIG and ISDN point-to-point links are needed to make the DivertingLegInformation1, DivertingLegInformation2, and DivertingLegInformation3 messages operate properly.

You should manually send the COLR of the redirected-to party for an incoming redirected call if the incoming call could experience further redirects. For chan_misdn, just set the REDIRECTING(to-num,i) = ${EXTEN} and set the REDIRECTING(to-num-pres) to the COLR. For chan_dahdi, just set the REDIRECTING(to-num,i) = CALLERID(dnid) and set the REDIRECTING(to-num-pres) to the COLR. (Setting the REDIRECTING(to-num,i) value may not be necessary since the channel driver has already attempted to preset that value for automatic generation of the needed DivertingLegInformation3 message.)

For redirected calls out a trunk line, you need to use the 'i' option on all of the REDIRECTING statements before dialing the redirected-to party. The call will update the redirecting-to presentation (COLR) when it becomes available.

Dial() and Queue() dialplan application 'I' option

In the dialplan applications Dial() and Queue(), the 'I' option is a brute force option to block connected line and redirecting information updates while the application is running. Blocking the updates prevents the update from overwriting any CONNECTEDLINE or REDIRECTING values you may have setup before running the application.

The option blocks all redirecting updates since they should only happen before a call is answered. The option only blocks the connected line update from the initial answer. Connected line updates resulting from call transfers happen after the application has completed. Better control of connected line and redirecting information is obtained using the interception macros.

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}.

Interception routines

As Interception routines are implemented internally using the Gosub application, all routines should end with an explicit call to the Return applica tion.

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}.

Manipulation examples

The following examples show several common scenarios in which you may need to manipulate party ID information from the dialplan.

Simple recording playback

```
exten => 1000,1,NoOp
; The CONNECTEDLINE information is sent when the call is answered.
exten => 1000,n,Set(CONNECTEDLINE(name,i)=Company Name)
exten => 1000,n,Set(CONNECTEDLINE(name-pres,i)=allowed)
exten => 1000,n,Set(CONNECTEDLINE(num,i)=5551212)
exten => 1000,n,Set(CONNECTEDLINE(num-pres)=allowed)
exten => 1000,n,Answer
exten => 1000,n,Playback(tt-weasels)
exten => 1000,n,Hangup
```

Straightforward dial through

```
exten => 1000,1,NoOp
; The CONNECTEDLINE information is sent when the call is answered.
exten => 1000,n,Set(CONNECTEDLINE(name,i)=Company Name)
exten => 1000,n,Set(CONNECTEDLINE(name-pres,i)=allowed)
exten => 1000,n,Set(CONNECTEDLINE(num,i)=5551212)
exten => 1000,n,Set(CONNECTEDLINE(num-pres)=allowed)
; The I option prevents overwriting the CONNECTEDLINE information
; set above when the call is answered.
exten => 1000,n,Dial(SIP/1000,20,I)
exten => 1000,n,Hangup
```

Use of interception macro

```
[macro-add_pfx]
; ARG1 is the prefix to add.
; ARG2 is the number of digits at the end to add the prefix to.
; When the macro ends the CONNECTEDLINE data is passed to the
; channel driver.
exten => s,1,NoOp(Add prefix to connected line)
exten => s,n,Set(NOPREFIX=${CONNECTEDLINE(number):-${ARG2}})
exten => s,n,Set(CONNECTEDLINE(num,i)=${ARG1}${NOPREFIX})
exten => s,n,MacroExit

exten => 1000,1,NoOp
exten => 1000,n,Set(__CONNECTED_LINE_CALLER_SEND_MACRO=add_pfx)
exten => 1000,n,Set(__CONNECTED_LINE_CALLER_SEND_MACRO_ARGS=45,4)
exten => 1000,n,Dial(SIP/1000,20)
exten => 1000,n,Hangup
```

Simple redirection

```
exten => 1000,1,NoOp
; For Q.SIG or ISDN point-to-point we should determine the COLR for this
; extension and send it if the call was redirected here.
exten => 1000,n,GotoIf($[${REDIRECTING(count)}>0]?redirected:notredirected)
exten => 1000,n(redirected),Set(REDIRECTING(to-num,i)=${CALLERID(dnid)})
exten => 1000,n,Set(REDIRECTING(to-num-pres)=allowed)
exten => 1000,n(notredirected),NoOp
; Determine that the destination has forwarded the call.
; ...
exten => 1000,n,Set(REDIRECTING(from-num,i)=1000)
exten => 1000,n,Set(REDIRECTING(from-num-pres,i)=allowed)
exten => 1000,n,Set(REDIRECTING(to-num,i)=2000)
; The DivertingLegInformation3 message is needed because at this point
; we do not know the presentation (COLR) setting of the redirecting-to
; party.
exten => 1000,n,Set(REDIRECTING(count,i)=$[${REDIRECTING(count)} + 1])
exten => 1000,n,Set(REDIRECTING(reason,i)=cfu)
; The call will update the redirecting-to presentation (COLR) when it
; becomes available with a redirecting update.
exten => 1000,n,Dial(DAHDI/g1/2000,20)
exten => 1000,n,Hangup
```

Ideas for usage

The following is a list of ideas in which the manipulation of party ID information would be beneficial.

- IVR that updates connected name on each selection made.
- Disguise the true number of an individual with a generic company number.
- Use interception macros to make outbound connected number E.164 formatted.
- You can do a lot more in an interception macro than just manipulate party information...

Troubleshooting tips

- For CONNECTEDLINE and REDIRECTING, check the usage of the 'i' option.
- Check channel configuration settings. The default settings may not be what you want or expect.
- Check packet captures. Your equipment may not support what Asterisk sends.

For further reading...

- Relevant ETSI ISDN redirecting specification: EN 300 207-1
- Relevant ETSI ISDN COLP specification: EN 300 097-1
- Relevant ETSI ISDN ECT specification: EN 300 369-1
- Relevant Q.SIG ISDN redirecting specification: ECMA-174
- Relevant Q.SIG ISDN COLP specification: ECMA-148
- Relevant Q.SIG ISDN ECT specification: ECMA-178
- Relevant SIP RFC for P-Asserted-Id: RFC3325
- The expired draft (draft-ietf-sip-privacy-04.txt) defines Remote-Party-Id. Since Remote-Party-Id has not made it into an RFC at this time, its use is non-standard by definition.

Packet Loss Concealment (PLC)

What is PLC?

PLC stands for Packet Loss Concealment. PLC describes any method of generating new audio data when packet loss is detected. In Asterisk, there are two main flavors of PLC, generic and native. Generic PLC is a method of generating audio data on signed linear audio streams. Signed linear audio, often abbreviated "slin," is required since it is a raw format that has no companding, compression, or other transformations applied. Native PLC is used by specific codec implementations, such as iLBC and Speex, which generates the new audio in the codec's native format. Native PLC happens automatically when using a codec that supports native PLC. Generic PLC requires specific configuration options to be used and will be the focus of this document.

How does Asterisk detect packet loss?

Oddly, Asterisk does not detect packet loss when reading audio in. In order to detect packet loss, one must have a jitter buffer in use on the channel on which Asterisk is going to write missing audio using PLC. When a jitter buffer is in use, audio that is to be written to the channel is fed into the jitterbuffer. When the time comes to write audio to the channel, a bridge will request that the jitter buffer gives a frame of audio to the bridge so that the audio may be written. If audio is requested from the jitter buffer but the jitter buffer is unable to give enough audio to the bridge, then the jitter buffer will return an interpolation frame. This frame contains no actual audio data and indicates the number of samples of audio that should be inserted into the frame.

PLC Background on Translation

As stated in the introduction, generic PLC can only be used on slin audio. The majority of audio communication is not done in slin, but rather using lower bandwidth codecs. This means that for PLC to be used, there must be a translation step involving slin on the write path of a channel. This means that PLC cannot be used if the codecs on either side of the bridge are the same or do not require a translation to slin in order to translate between them. For instance, a ulaw - ulaw call will not use PLC since no translation is required. In addition, a ulaw - alaw call will also not use PLC since the translation path does not include any step involving slin. One item of note is that slin must be present on the write path of a channel since that is the path where PLC is applied. Consider that Asterisk is bridging channels A and B. A uses ulaw for audio and B uses GSM. This translation involves slin, so things are shaping up well for PLC. Consider, however if Asterisk sets up the translation paths like so:

Fig. 1

```
A ----------- B <-ulaw<-slin<-GSM| |GSM-> | Asterisk | ulaw->slin->GSM->| |<-GSM -----------
```

The arrows indicate the direction of audio flow. Each channel has a write path (the top arrow) and a read path (the bottom arrow). In this setup, PLC can be used when sending audio to A, but it cannot be used when sending audio to B. The reason is simple, the write path to A's channel contains a slin step, but the write path to B contains no slin step. Such a translation setup is perfectly valid, and Asterisk can potentially set up such a path depending on circumstances. When we use PLC, however, we want slin audio to be present on the write paths of both A and B. A visual representation of what we want is the following:

Fig. 2

```
A ----------- B <-ulaw<-slin| |slin->GSM-> | Asterisk | ulaw->slin->| |<-slin<-GSM -----------
```

In this scenario, the write paths for both A and B begin with slin, and so PLC may be applied to either channel. This translation behavior has, in the past been doable with the transcode_via_sln option in asterisk.conf. Recent changes to the PLC code have also made the genericplc option in codecs.conf imply the transcode_via_sln option. The result is that by enabling genericplc in codecs.conf, the translation path set up in Fig. 2 should automatically be used.

PLC Restrictions and Caveats

One restriction that has not been spelled out so far but that has been hinted at is the presence of a bridge. The term bridge in this sense means two channels exchanging audio with one another. A bridge is required because use of a jitter buffer is a prerequisite for using PLC, and a jitter buffer is only used when bridging two channels. This means that one-legged calls, (e.g. calls to voicemail, to an IVR, to an extension that just plays back audio) will not use PLC. In addition, MeetMe and ConfBridge calls will not use PLC. It should be obvious, but it bears mentioning, that PLC cannot be used when using a technology's native bridging functionality. For instance, if two SIP channels can exchange RTP directly, then Asterisk will never be able to process the audio in the first place. Since translation of audio is a requirement for using PLC, and translation will not allow for a native bridge to be created, this is something that is not likely to be an issue, though. Since a jitter buffer is a requirement in order to use PLC, it should be noted that simply enabling the jitter buffer via the jbenable option may not be enough. For instance, if bridging two SIP channels together, the default behavior will not be to enable jitter buffers on either channel. The rationale is that the jitter will be handled at the endpoints to which Asterisk is sending the audio. In order to ensure that a jitter buffer is used in all cases, one must enable the jbforce option for channel types on which PLC is desired.

Requirements for PLC Use

The following are all required for PLC to be used:

1. Enable genericplc in the plc section of codecs.conf
2. Enable (and potentially force) jitter buffers on channels
3. Two channels must be bridged together for PLC to be used (no Meetme or one-legged calls)
4. The audio must be translated between the two channels and must have slin as a step in the translation process.

PLC Tips

One of the restrictions mentioned is that PLC will only be used when two audio channels are bridged together. Through the use of Local channels, you can create a bridge even if the call is, for all intents and purposes, one-legged. By using a combination of the /n and /j suffixes for a Local channel, one can ensure that the Local channel is not optimized out of the talk path and that a jitter buffer is applied to the Local channel as well. Consider the following simple dialplan:

```
[example]
exten => 1,1,Playback(tt-weasels)
exten => 2,1,Dial(Local/1@example/nj)
```

When dialing extension 1, PLC cannot be used because there will be only a single channel involved. When dialing extension 2, however, Asterisk will create a bridge between the incoming channel and the Local channel, thus allowing PLC to be used.

Phone Provisioning in Asterisk

Asterisk includes basic phone provisioning support through the res_phoneprov module. The current implementation is based on a templating system using Asterisk dialplan function and variable substitution and obtains information to substitute into those templates from phoneprov.conf and users.conf. A profile and set of templates is provided for provisioning Polycom phones. Note that res_phoneprov is currently limited to provisioning a single user per device.

Configuration of phoneprov.conf

The configuration file, phoneprov.conf, is used to set up the built-in variables SERVER and SERVER_PORT, to define a default phone profile to use, and to define different phone profiles available for provisioning.

The [general] section

Below is a sample of the general section of phoneprov.conf:

```
[general]
;serveriface=eth0
;serveraddr=192.168.1.1
;serverport=5060
default_profile=polycom
```

By default, res_phoneprov will set the SERVER variable to the IP address on the server that the requesting phone uses to contact the asterisk HTTP server. The SERVER_PORT variable will default to the bindport setting in sip.conf.

Should the defaults be insufficient, there are two choices for overriding the default setting of the SERVER variable. If the IP address of the server is known, or the hostname resolvable by the phones, the appropriate serveraddr value should be set. Alternatively, the network interface that the server listens on can be set by specifying a serveriface and SERVER will be set to the IP address of that interface. Only one of these options should be set.

The default SERVER_PORT variable can be overridden by setting the serverport. If bindport is not set in sip.conf and serverport is not specified, it is set to a default value of 5060.

Any user set for auto-provisioning in users.conf without a specified profile will be assumed to belong to the profile set with default_profile.

Creating Phone Profiles

A phone profile is basically a list of files that a particular group of phones needs to function. For most phone types there are files that are identical for all phones (firmware, for instance) as well as a configuration file that is specific to individual phones. res_phoneprov breaks these two groups of files into static files and dynamic files, respectively. A sample profile:

```
[polycom]
staticdir => configs/
mime_type => text/xml
setvar => CUSTOM_CONFIG=/var/lib/asterisk/phoneprov/configs/custom.cfg
static_file => bootrom.ld,application/octet-stream
static_file => bootrom.ver,plain/text
static_file => sip.ld,application/octet-stream
static_file => sip.ver,plain/text
static_file => sip.cfg
static_file => custom.cfg
${TOLOWER(${MAC})}.cfg => 000000000000.cfg
${TOLOWER(${MAC})}-phone.cfg => 000000000000-phone.cfg config/
${TOLOWER(${MAC})} => polycom.xml
${TOLOWER(${MAC})}-directory.xml => 000000000000-directory.xml
```

A static_file is set by specifying the file name, relative to AST_DATA_DIR/phoneprov. The mime-type of the file can optionally be specified after a comma. If staticdir is set, all static files will be relative to the subdirectory of AST_DATA_DIR/phoneprov specified.

Since phone-specific config files generally have file names based on phone-specifc data, dynamic filenames in res_phoneprov can be defined with Asterisk dialplan function and variable substitution. In the above example, ${TOLOWER(${MAC})}.cfg = 000000000000.cfg would define a relative URI to be served that matches the format of MACADDRESS.cfg, all lower case. A request for that file would then point to the template found at AST_DATA_DIR/phoneprov/000000000000.cfg. The template can be followed by a comma and mime-type. Notice that the dynamic filename (URI) can contain contain directories. Since these files are dynamically generated, the config file itself does not reside on the filesystem-only the template. To view the generated config file, open it in a web browser. If the config file is XML, Firefox should display it. Some browsers will require viewing the source of the page requested.

A default mime-type for the profile can be defined by setting mime-type. If a custom variable is required for a template, it can be specified with setvar. Variable substitution on this value is done while building the route list, so ${USERNAME} would expand to the username of the users.conf user that registers the dynamic filename.

⚠ Any dialplan function that is used for generation of dynamic file names MUST be loaded before res_phoneprov. Add "preload = modulename.so" to modules.conf for required functions. In the example above, "preload = func_strings.so" would be required.

Configuration of users.conf

The asterisk-gui sets up extensions, SIP/IAX2 peers, and a host of other settings. User-specific settings are stored in users.conf. If the asterisk-gui is not being used, manual entries to users.conf can be made.

The [general] section

There are only two settings in the general section of users.conf that apply to phone provisioning: localextenlength which maps to template variable EXTENSION_LENGTH and vmexten which maps to the VOICEMAIL_EXTEN variable.

Individual Users

To enable auto-provisioning of a phone, the user in users.conf needs to have:

```
...
autoprov=yes
macaddress=deadbeef4dad
profile=polycom
```

The profile is optional if a default_profile is set in phoneprov.conf. The following is a sample users.conf entry, with the template variables commented next to the settings:

```
[6001]
callwaiting = yes
context = numberplan-custom-1
hasagent = no
hasdirectory = yes
hasiax = no
hasmanager = no
hassip = yes
hasvoicemail = yes
host = dynamic
mailbox = 6001
threewaycalling = yes
deletevoicemail = no
autoprov = yes
profile = polycom
directmedia = no
nat = no
fullname = User Two ; ${DISPLAY_NAME}
secret = test ; ${SECRET}
username = 6001 ; ${USERNAME}
macaddress = deadbeef4dad ; ${MAC}
label = 6001 ; ${LABEL}
cid_number = 6001 ; ${CALLERID}
```

The variables above, are the user-specfic variables that can be substituted into dynamic filenames and config templates.

Phone Provisioning Templates

Configuration templates are a generic way to configure phones with text-based configuration files. Templates can use any loaded dialplan function and all of the variables created by phoneprov.conf and users.conf. A short example is the included 000000000000.cfg Polycom template:

```
<?xml version="1.0" standalone="yes"?>
  <APPLICATION
    APP_FILE_PATH="sip.ld"
    CONFIG_FILES="${IF($[${STAT(e,${CUSTOM_CONFIG})}] ? "custom.cfg,
")}config/${TOLOWER(${MAC})}, sip.cfg"
    MISC_FILES="" LOG_FILE_DIRECTORY=""
 />
```

This template uses dialplan functions, expressions, and a couple of variables to generate a config file to instruct the Polycom where to pull other needed config files. If a phone with MAC address 0xDEADBEEF4DAD requests this config file, and the filename that is stored in variable CUSTOM_CONFIG does not exist, then the generated output would be:

```
<?xml version="1.0" standalone="yes"?>
  <APPLICATION
    APP_FILE_PATH="sip.ld"
    CONFIG_FILES="config/deadbeef4dad, sip.cfg"
    MISC_FILES=""
    LOG_FILE_DIRECTORY=""
/>
```

The Polycom phone would then download both sip.cfg (which would be registered in phoneprov.conf as a static file) and config/deadbeef4dad (which would be registered as a dynamic file pointing to another template, polycom.xml).

res_phoneprov also registers its own dialplan function: PP_EACH_USER. This function was designed to be able to print out a particular string for each user that res_phoneprov knows about. An example use of this function is the template for a Polycom contact directory:

```
<?xml version="1.0" standalone="yes"?>
<directory>
  <item_list>

${PP_EACH_USER(<item><fn>%{DISPLAY_NAME}</fn><ct>%{CALLERID}</ct><bw>1</bw></item>|${MAC}
)}
  </item_list>
</directory>
```

PP_EACH_USER takes two arguments. The first is the string to be printed for each user. Any variables that are to be substituted need to be in the format %{VARNAME} so that Asterisk doesn't try to substitute the variable immediately before it is passed to PP_EACH_USER. The second, optional, argument is a MAC address to exclude from the list iterated over (so, in this case, a phone won't be listed in its own contact directory).

Phone Provisioning, Putting it all together

Make sure that manager.conf has:

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

and that http.conf has:

```
[general]
enabled = yes
bindaddr = 192.168.1.1 ; Your IP here
bindport = 8088 ; Or port 80 if it is the only http server running on the machine
```

With phoneprov.conf and users.conf in place, start Astersik. From the CLI, type "http show status". An example output:

```
HTTP Server Status:
Prefix: /asterisk
Server Enabled and Bound to 192.168.1.1:8088

Enabled URI's:
/asterisk/httpstatus => Asterisk HTTP General Status
/asterisk/phoneprov/... => Asterisk HTTP Phone Provisioning Tool
/asterisk/manager => HTML Manager Event Interface
/asterisk/rawman => Raw HTTP Manager Event Interface
/asterisk/static/... => Asterisk HTTP Static Delivery
/asterisk/mxml => XML Manager Event Interface
Enabled Redirects:
 None.
POST mappings:
 None.
```

There should be a phoneprov URI listed. Next, from the CLI, type "phoneprov show routes" and verify that the information there is correct. An example output for Polycom phones woud look like:

```
Static routes

Relative URI Physical location
sip.ver configs/sip.ver
sip.ld configs/sip.ld
bootrom.ver configs/bootrom.ver
sip.cfg configs/sip.cfg
bootrom.ld configs/bootrom.ld
custom.cfg configs/custom.cfg
Dynamic routes
Relative URI Template
deadbeef4dad.cfg 000000000000.cfg
deadbeef4dad-directory.xml
000000000000-directory.xml
deadbeef4dad-phone.cfg
000000000000-phone.cfg
config/deadbeef4dad polycom.xml
```

With the above examples, the phones would be pointed to:

http://192.168.1.1:8080/asterisk/phoneprov

for pulling config files.

Templates would all be placed in AST_DATA_DIR/phoneprov and static files would be placed in AST_DATA_DIR/phoneprov/configs. Examples of valid URIs would be:

- http://192.168.1.1:8080/asterisk/phoneprov/sip.cfg
- http://192.168.1.1:8080/asterisk/phoneprov/deadbeef4dad.cfg
- http://192.168.1.1:8080/asterisk/phoneprov/config/deadbeef4dad

Reference Information Introduction

Introductory section to the Asterisk Reference Manual. Read this section first.

License Information

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 two management/control protocols: the Asterisk Manager Interface (AMI) and the Asterisk Gateway Interface (AGI). 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

Hold Music License

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.

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

Telephony Hardware

A PBX is only really useful if you can get calls into it. Of course, you can use Asterisk with VoIP calls (SIP, H.323, IAX, etc.), but you can also talk to the real PSTN through various cards.

Supported Hardware is divided into two general groups: DAHDI devices and non-DAHDI devices. The DAHDI compatible hardware supports pseudo-TDM conferencing and all call features through chan_dahdi, whereas non-DAHDI compatible hardware may have different features.

DAHDI compatible hardware

Digium, Inc (Primary Developer of Asterisk) DAHDI compatible hardware

Analog Interfaces

TDM410 and AEX410 - The TDM410P and AEX410 are modular four-port half-length PCI 2.2 and PCI-Express x1, respectively, cards that supports FXS and FXO station interfaces for connecting analog telephones and analog POTS lines through a PC.

TDM800 and AEX800 - The TDM800 and AEX800 are modular eight-port half-length PCI 2.2 and PCI-Express x1, respectively, cards that support FXS and FXO station interfaces for connecting analog telephones and analog POTS lines through a PC.

TDM2400 and AEX2400 - The TDM2400 and AEX2400 are modular twenty-four port full-length PCI 2.2 and PCI-Express x1, respectively, cards that support FXS and FXO station interfaces for connecting analog telephones and analog POTS lines through a PC.

Digital Interfaces

TE122 and TE121 - The TE122 and TE122 are half-length, half-height PCI 2.2 and PCI-Express x1, respectively, single-span E1/T1/J1 PRI/PRA software-configurable interfaces for terminating digital telephony through a PC.
TE205/7, TE210/12 and TE220 - The TE205, TE210 and TE220 are half-length PCI 2.2 5.0V, PCI 2.2 3.3V, and PCI-Express x1 dual-span E1/T1/J1 PRI/PRA software-configurable interfaces for terminating digital telephony through a PC.
TE405/7, TE410/12 and TE420 - The TE405/7, TE410/12 and TE420 are half-length PCI 2.2 5.0V, PCI 2.2 3.3V, and PCI-Express x1 quad-span E1/T1/J1 PRI/PRA software-configurable interfaces for terminating digital telephony through a PC.
HA8 and HB8 - The HA8 and HB8 are half-length PCI2.2 and PCI-Express x1, respectively, modular eight-port EuroISDN BRI and Analog FXS/FXO interfaces for terminating telephony through a PC.
B410P - The B410P is a half-length PCI-2.2 fixed four-port EuroISDN BRI interface for terminating digital telephony through a PC.

Packet Interfaces

TC400 and TCE400 - The TC400 and TCE400 are half-length, half-height PCI 2.2 and PCI-Express x1, respectively, codec translation cards for DSP-based translations of patent-encumbered codecs such as G.729a and G.723.1

Non-DAHDI compatible hardware

Non-DAHDI compatible hardware

QuickNet, Inc. http://www.quicknet.net

Internet PhoneJack - Single FXS interface. Supports Linux telephony interface. DSP compression built-in.

Internet LineJack - Single FXS or FXO interface. Supports Linux telephony interface.

mISDN compatible hardware

mISDN compatible hardware

Any adapter with an mISDN driver should be compatible with chan_misdn. See the mISDN section for more information.

- **Digium, Inc.**
  **B410P - also compatible with mISDN
- **beroNet** http://www.beronet.com
  **BN4S0 - 4 Port BRI card (TE/NT)
  **BN8S0 - 8 Port BRI card (TE/NT)
- **Billion Card - Single Port BRI card (TE (/NT with crossed cable))**

Miscellaneous other interfaces

Miscellaneous other interfaces

ALSA http://www.alsa-project.org
Any ALSA compatible full-duplex sound card
OSS http://www.opensound.com
Any OSS compatible full-duplex sound card

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, therefor 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

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.

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.

The Asterisk SIP configuration

Now, let's configure Asterisk to use TLS.

In the **sip.conf** configuration file, set the following:

```
tlsenable=yes
tlsbindaddr=0.0.0.0
tlscertfile=/etc/asterisk/keys/asterisk.pem
tlscafile=/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.

Configuring a TLS-enabled SIP peer within Asterisk

Next, you'll need to configure a SIP peer within Asterisk to use TLS as a transport type. Here's an example:

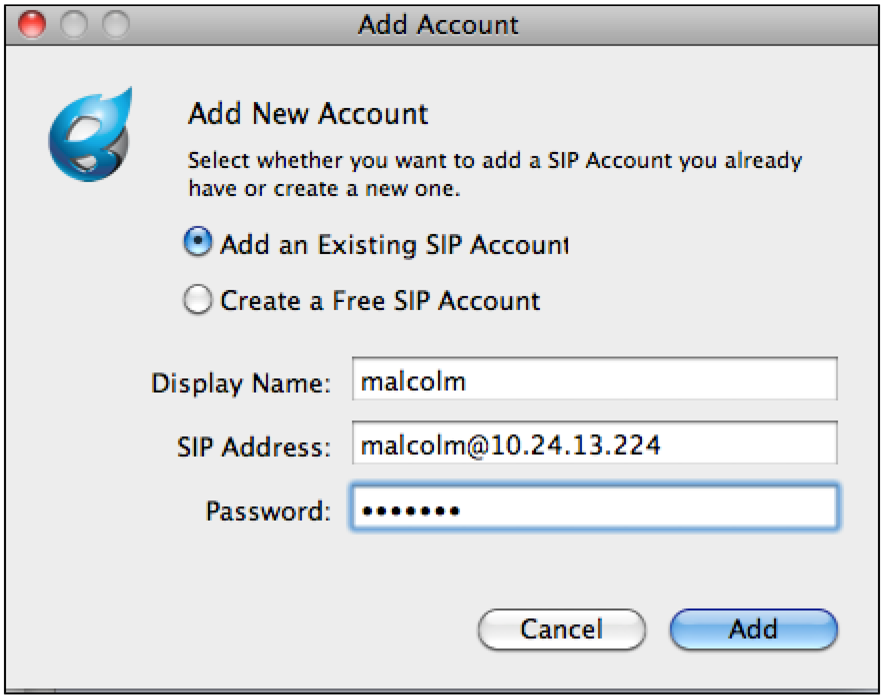
```
[malcolm]
type=peer
secret=malcolm ;note that this is NOT a secure password
host=dynamic
context=local
dtmfmode=rfc2833
disallow=all
allow=g722
transport=tls
context=local
```

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.

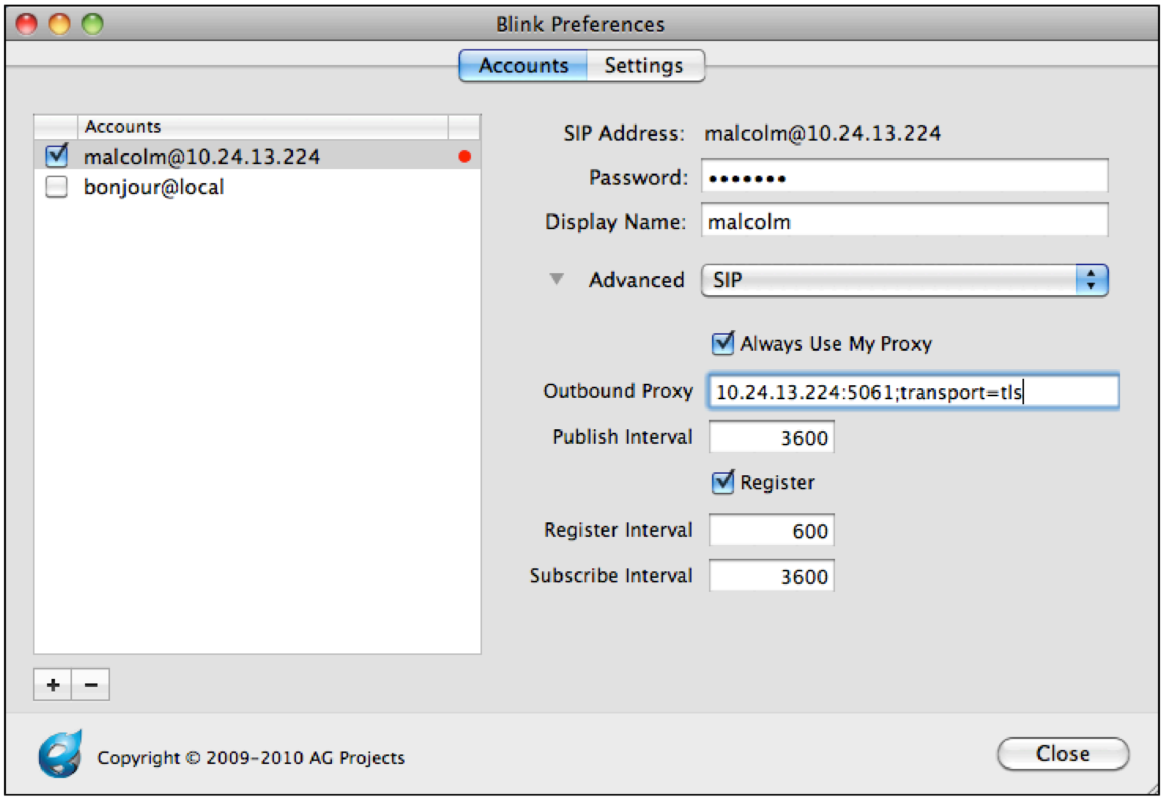


Select whether you want to add a SIP Account you already have or create a new one.

[✓ Add an Existing SIP Account]
Create a Free SIP Account

Display Name: malcolm
SIP Address: malcolm@10.24.13.224
Password: •••••••

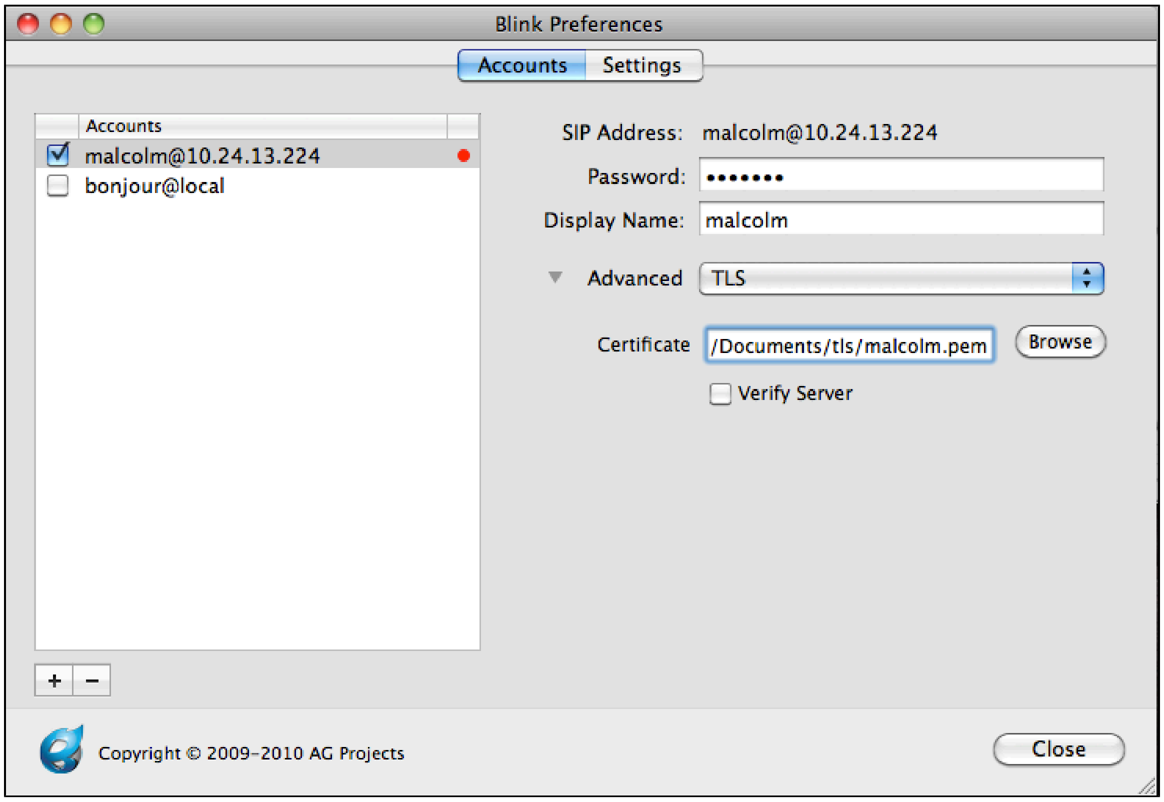
Then, we need to modify the Account Preferences, and under the SIP Settings, we need to set the outbound proxy to connect to the TLS port and transport type on our Asterisk server. In this case, there's an Asterisk server running on port 5061 on host 10.24.13.233.



[✓] malcolm@10.24.13.224
[ ] bonjour@local

SIP Address: malcolm@10.24.13.224
Password: •••••••
Display Name: malcolm
Advanced: SIP
[✓] Always Use My Proxy
Outbound Proxy: 10.24.13.224:5061;transport=tls
Publish Interval: 3600
[✓] Register
Register Interval: 600
Subscribe Interval: 3600

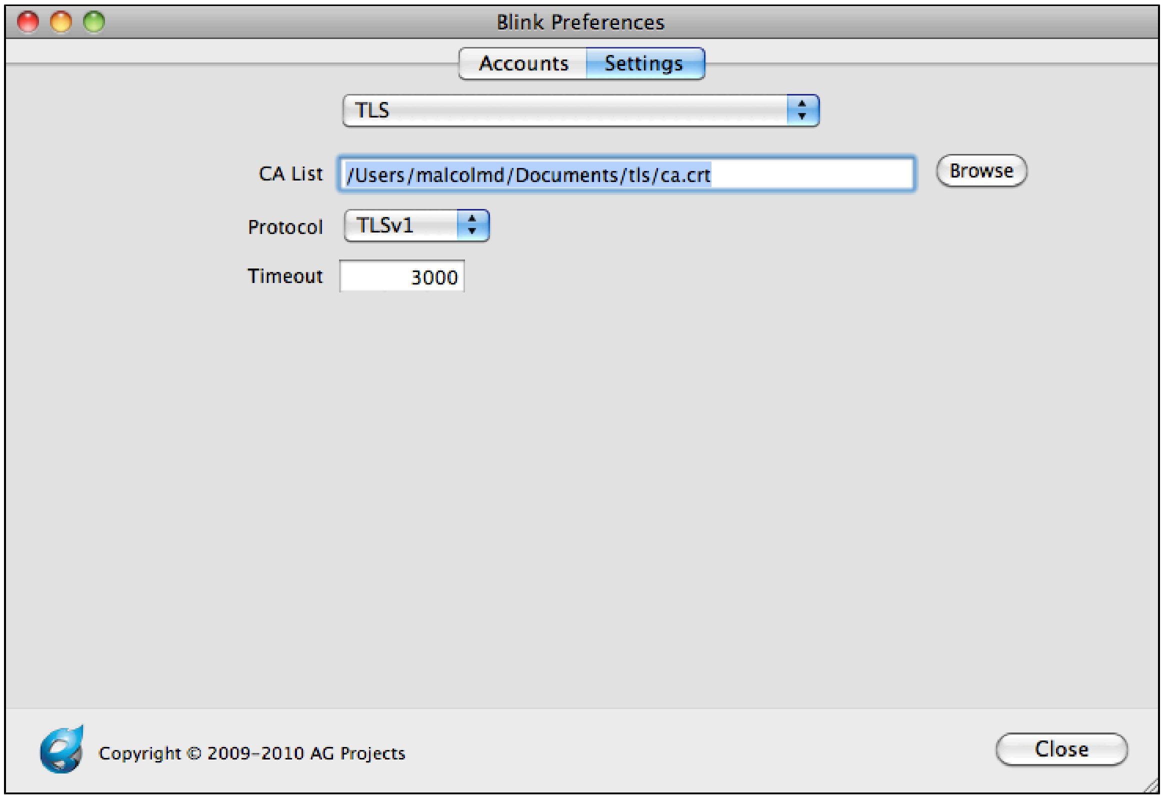
Now, we need to point the TLS account settings to the client certificate (malcolm.pem) that we copied to our computer.



[✓] malcolm@10.24.13.224
[ ] bonjour@local

SIP Address: malcolm@10.24.13.224
Password: •••••••
Display Name: malcolm
Advanced: TLS
Certificate: /Documents/tls/malcolm.pem
[ ] Verify Server

Then, we'll point the TLS server settings to the ca.crt file that we copied to our computer.



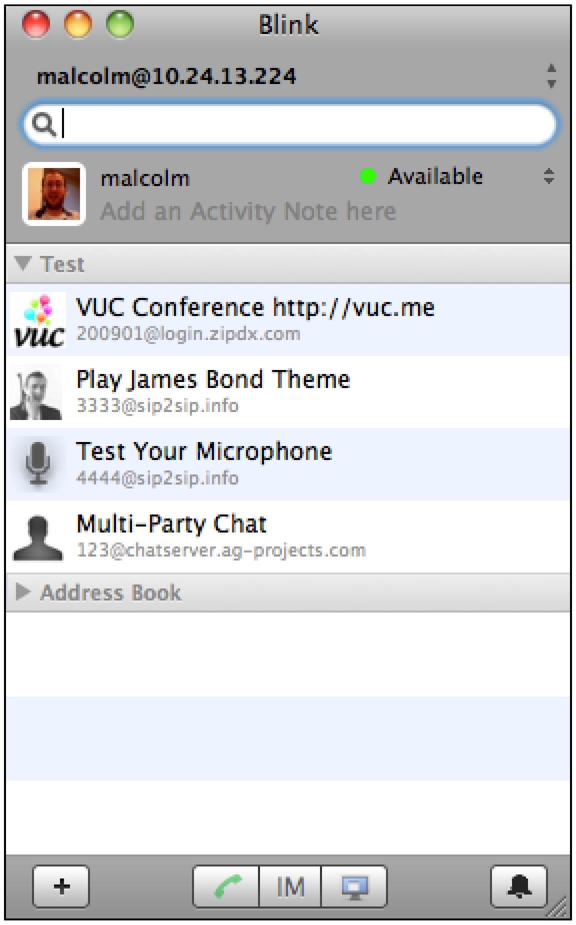
CA List /Users/malcolmd/Documents/tls/ca.crt

Protocol TLSv1

Timeout 3000

Copyright © 2009-2010 AG Projects

Press "close," and you should see Blink having successfully registered to Asterisk.

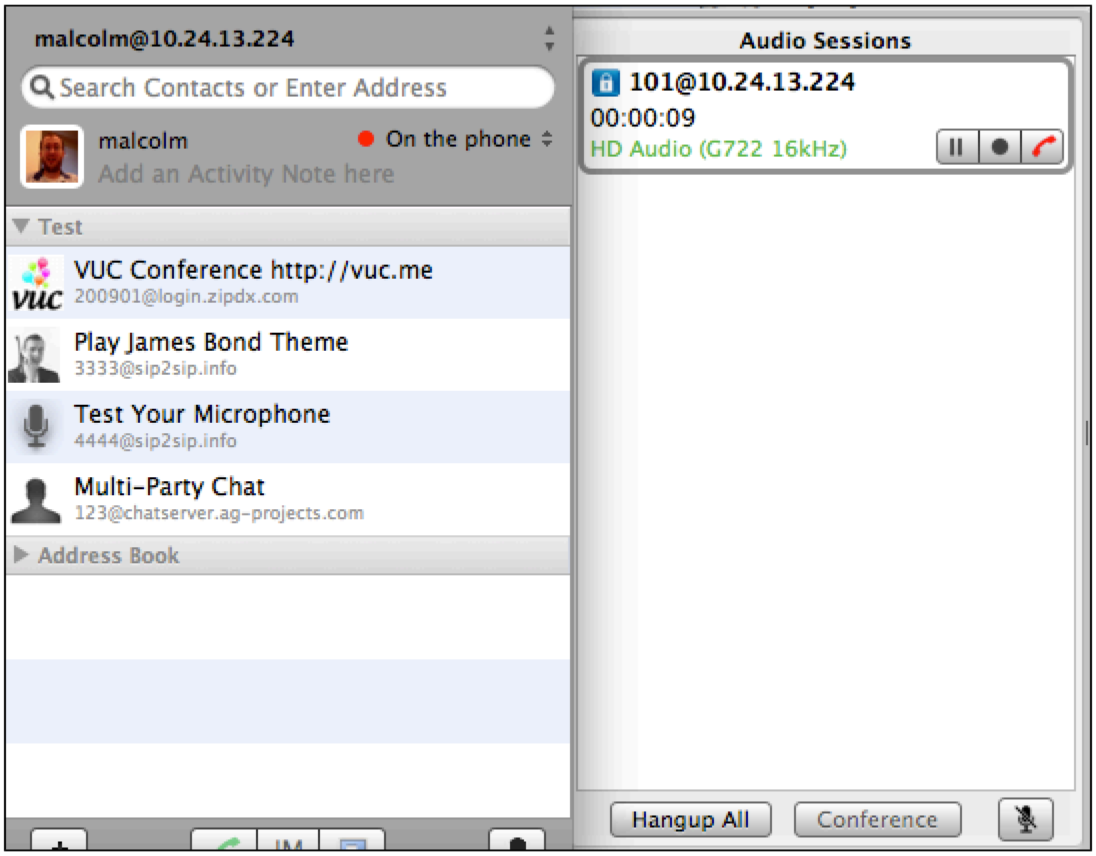


Depending on your Asterisk CLI logging levels, you should see something like:

```
-- 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:



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:

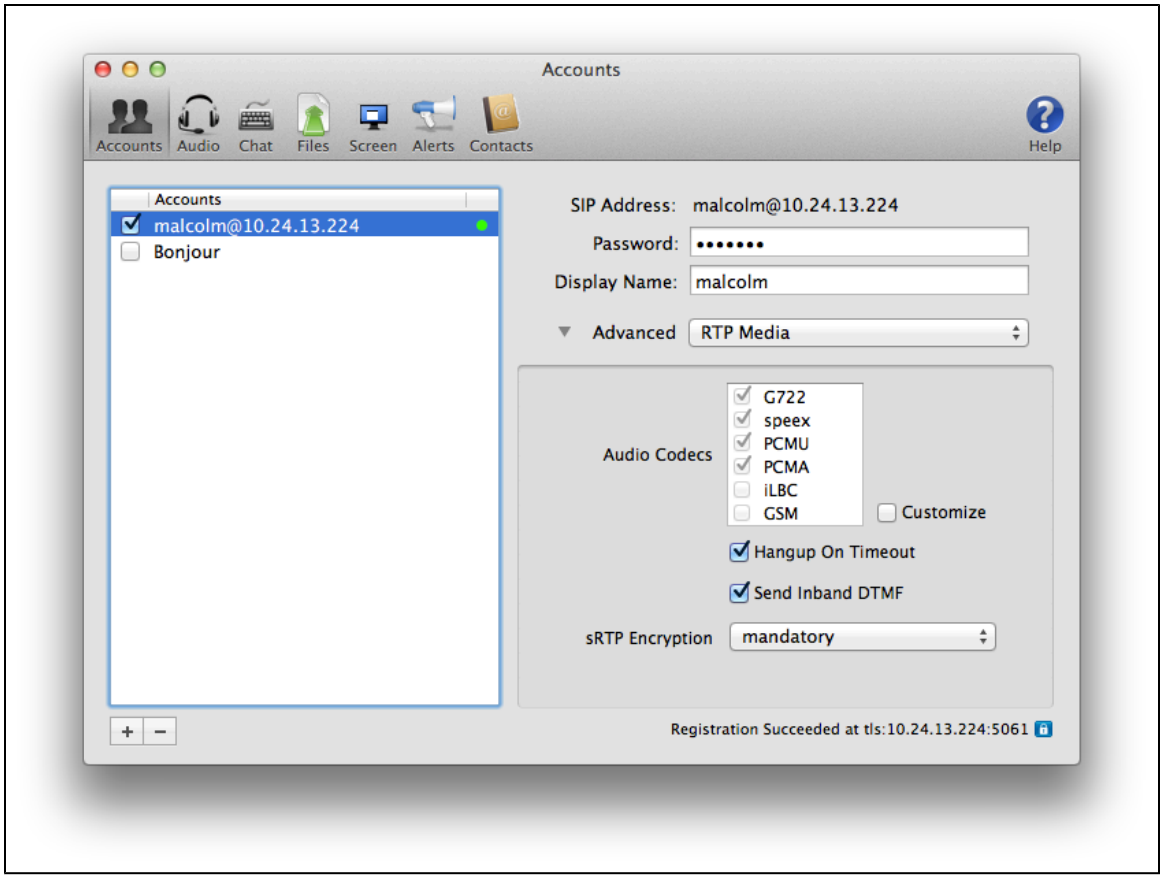
```
[Jan 24 09:29:16] ERROR[10167]: chan_sip.c:27987 setup_srtp: No SRTP module loaded, can't setup SRTP session.
```

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:


Accounts

SIP Address: malcolm@10.24.13.224

Password: •••••••

Display Name: malcolm

Advanced: RTP Media

Audio Codecs:
- [✓] G722
- [✓] speex
- [✓] PCMU
- [✓] PCMA
- [ ] iLBC
- [ ] GSM

[ ] Customize

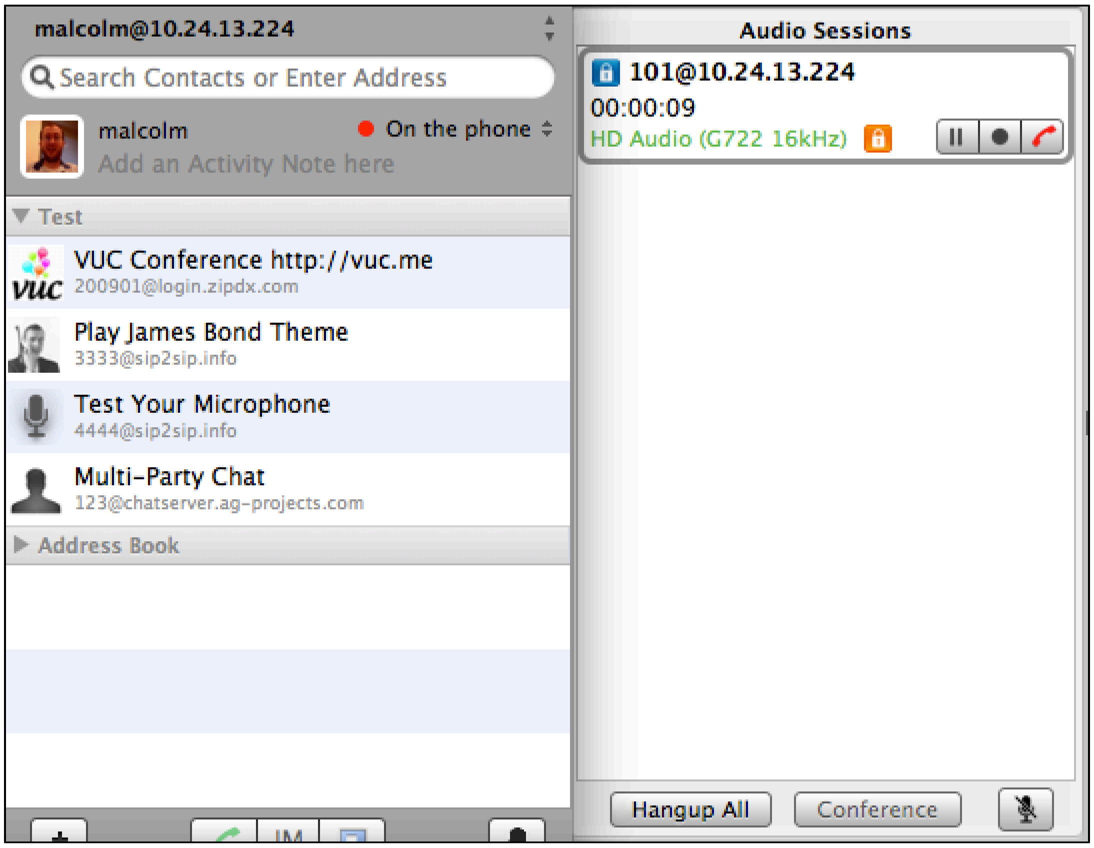
[✓] Hangup On Timeout

[✓] Send Inband DTMF

sRTP Encryption: mandatory

Registration Succeeded at tls:10.24.13.224:5061

Reload Asterisk's SIP configuration (sip reload), make a call, and voilà:


Audio Sessions

101@10.24.13.224

00:00:09

HD Audio (G722 16kHz)

We're making secure calls with TLS (signaling) and SRTP (media).

Shared Line Appearances (SLA)

What are shared line appearances and how do they work in Asterisk? Read on...

Introduction to Shared Line Appearances (SLA)

The "SLA" functionality in Asterisk is intended to allow a setup that emulates a simple key system. It uses the various abstraction layers already built into Asterisk to emulate key system functionality across various devices, including IP channels.

SLA Configuration

How-to configure SLA in Asterisk

SLA Configuration Summary

An SLA system is built up of virtual trunks and stations mapped to real Asterisk devices. The configuration for all of this is done in three different files: extensions.conf, sla.conf, and the channel specific configuration file such as sip.conf or dahdi.conf.

SLA Dialplan Configuration

The SLA implementation can automatically generate the dialplan necessary for basic operation if the "autocontext" option is set for trunks and stations in sla.conf. However, for reference, here is an automatically generated dialplan to help with custom building of the dialplan to include other features, such as voicemail.

However, note that there is a little bit of additional configuration needed if the trunk is an IP channel. This is discussed in the section on Trunks.

There are extensions for incoming calls on a specific trunk, which execute the SLATrunk application, as well as incoming calls from a station, which execute SLAStation. Note that there are multiple extensions for incoming calls from a station. This is because the SLA system has to know whether the phone just went off hook, or if the user pressed a specific line button.

Also note that there is a hint for every line on every station. This lets the SLA system control each individual light on every phone to ensure that it shows the correct state of the line. The phones must subscribe to the state of each of their line appearances.
Please refer to the examples section for full dialplan samples for SLA.

SLA Trunk Configuration

An SLA trunk is a mapping between a virtual trunk and a real Asterisk device. This device may be an analog FXO line, or something like a SIP trunk. A trunk must be configured in two places. First, configure the device itself in the channel specific configuration file such as dahdi.conf or sip.conf. Once the trunk is configured, then map it to an SLA trunk in sla.conf.

```
[line1]
type=trunk
device=DAHDI/1
```

Be sure to configure the trunk's context to be the same one that is set for the "autocontext" option in sla.conf if automatic dialplan configuration is used. This would be done in the regular device entry in dahdi.conf, sip.conf, etc. Note that the automatic dialplan generation creates the SLATrunk() extension at extension 's'. This is perfect for DAHDI channels that are FXO trunks, for example. However, it may not be good enough for an IP trunk, since the call coming in over the trunk may specify an actual number.

If the dialplan is being built manually, ensure that calls coming in on a trunk execute the SLATrunk() application with an argument of the trunk name, as shown in the dialplan example before.

IP trunks can be used, but they require some additional configuration to work.

For this example, let's say we have a SIP trunk called "mytrunk" that is going to be used as line4. Furthermore, when calls come in on this trunk, they are going to say that they are calling the number "12564286000". Also, let's say that the numbers that are valid for calling out this trunk are NANP numbers, of the form _1NXXNXXXXXX.

In sip.conf, there would be an entry for [mytrunk]. For [mytrunk], set context=line4.

```
[line4]
type=trunk
device=Local/disa@line4_outbound

[line4]
exten => 12564286000,1,SLATrunk(line4)

[line4_outbound]
exten => disa,1,Disa(no-password,line4_outbound)
exten => _1NXXNXXXXXX,1,Dial(SIP/${EXTEN}@mytrunk)
```

So, when a station picks up their phone and connects to line 4, they are connected to the local dialplan. The Disa application plays dialtone to the phone and collects digits until it matches an extension. In this case, once the phone dials a number like 12565551212, the call will proceed out the SIP trunk.

SLA Station Configuration

An SLA station is a mapping between a virtual station and a real Asterisk device. Currently, the only channel driver that has all of the features necessary to support an SLA environment is chan_sip. So, to configure a SIP phone to use as a station, you must configure sla.conf and sip.conf.

```
[station1]
type=station
device=SIP/station1
trunk=line1
trunk=line2
```

Here are some hints on configuring a SIP phone for use with SLA:

- Add the SIP channel as a station in sla.conf.
- Configure the phone in sip.conf. If automatic dialplan configuration was used by enabling the "autocontext" option in sla.conf, then this entry in sip.conf should have the same context setting.
- On the phone itself, there are various things that must be configured to make everything work correctly. Let's say this phone is called "station1" in sla.conf, and it uses trunks named "line1" and line2".
    - Two line buttons must be configured to subscribe to the state of the following extensions: - station1_line1 - station1_line2
    - The line appearance buttons should be configured to dial the extensions that they are subscribed to when they are pressed.
    - If you would like the phone to automatically connect to a trunk when it is taken off hook, then the phone should be automatically configured to dial "station1" when it is taken off hook.

SLA Configuration Examples

Example configurations for SLA

Basic SLA Configuration Example

This is an example of the most basic SLA setup. It uses the automatic dialplan generation so the configuration is minimal.
sla.conf:

```
[line1]
type=trunk
device=DAHDI/1
autocontext=line1

[line2]
type=trunk
device=DAHDI/2
autocontext=line2

[station]
type=station
trunk=line1
trunk=line2
autocontext=sla_stations

[station1](station)
device=SIP/station1

[station2](station)
device=SIP/station2

[station3](station)
device=SIP/station3
```

With this configuration, the dialplan is generated automatically. The first DAHDI channel should have its context set to "line1" and the second should be set to "line2" in dahdi.conf. In sip.conf, station1, station2, and station3 should all have their context set to "sla_stations".
For reference, here is the automatically generated dialplan for this situation:

```
[line1]
exten => s,1,SLATrunk(line1)

[line2]
exten => s,2,SLATrunk(line2)

[sla_stations]
exten => station1,1,SLAStation(station1)
exten => station1_line1,hint,SLA:station1_line1
exten => station1_line1,1,SLAStation(station1_line1)
exten => station1_line2,hint,SLA:station1_line2
exten => station1_line2,1,SLAStation(station1_line2)
exten => station2,1,SLAStation(station2)
exten => station2_line1,hint,SLA:station2_line1
exten => station2_line1,1,SLAStation(station2_line1)
exten => station2_line2,hint,SLA:station2_line2
exten => station2_line2,1,SLAStation(station2_line2)
exten => station3,1,SLAStation(station3)
exten => station3_line1,hint,SLA:station3_line1
exten => station3_line1,1,SLAStation(station3_line1)
exten => station3_line2,hint,SLA:station3_line2
exten => station3_line2,1,SLAStation(station3_line2)
```

SLA and Voicemail Example

This is an example of how you could set up a single voicemail box for the phone system. The voicemail box number used in this example is 1234, which would be configured in voicemail.conf.

For this example, assume that there are 2 trunks and 3 stations. The trunks are DAHDI/1 and DAHDI/2. The stations are SIP/station1, SIP/station2, and SIP/station3.

In dahdi.conf, channel 1 has context=line1 and channel 2 has context=line2.

In sip.conf, all three stations are configured with context=sla_stations.

When the stations pick up their phones to dial, they are allowed to dial NANP numbers for outbound calls, or 8500 for checking voicemail.

sla.conf:

```
[line1]
type=trunk
device=Local/disa@line1_outbound

[line2]
type=trunk
device=Local/disa@line2_outbound

[station]
type=station
trunk=line1
trunk=line2

[station1](station)
device=SIP/station1

[station2](station)
device=SIP/station2

[station3](station)
device=SIP/station3
```

extensions.conf:

```
[macro-slaline]
exten => s,1,SLATrunk(${ARG1})
exten => s,n,Goto(s-${SLATRUNK_STATUS},1)
exten => s-FAILURE,1,Voicemail(1234,u)
exten => s-UNANSWERED,1,Voicemail(1234,u)

[line1]
exten => s,1,Macro(slaline,line1)

[line2]
exten => s,2,Macro(slaline,line2)

[line1_outbound]
exten => disa,1,Disa(no-password,line1_outbound)
exten => _1NXXNXXXXXX,1,Dial(DAHDI/1/${EXTEN})
exten => 8500,1,VoicemailMain(1234)

[line2_outbound]
exten => disa,1,Disa(no-password|line2_outbound)
exten => _1NXXNXXXXXX,1,Dial(DAHDI/2/${EXTEN})
exten => 8500,1,VoicemailMain(1234)

[sla_stations]
exten => station1,1,SLAStation(station1)
exten => station1_line1,hint,SLA:station1_line1
exten => station1_line1,1,SLAStation(station1_line1)
exten => station1_line2,hint,SLA:station1_line2
exten => station1_line2,1,SLAStation(station1_line2)
exten => station2,1,SLAStation(station2)
exten => station2_line1,hint,SLA:station2_line1
exten => station2_line1,1,SLAStation(station2_line1)
exten => station2_line2,hint,SLA:station2_line2
exten => station2_line2,1,SLAStation(station2_line2)
exten => station3,1,SLAStation(station3)
exten => station3_line1,hint,SLA:station3_line1
exten => station3_line1,1,SLAStation(station3_line1)
exten => station3_line2,hint,SLA:station3_line2
exten => station3_line2,1,SLAStation(station3_line2)
```

SLA and Call Handling

Read about call handling related to SLA

SLA Call Handling Summary

This section is intended to describe how Asterisk handles calls inside of the SLA system so that it is clear what behavior is expected.

SLA Station goes off hook (not ringing)

When a station goes off hook, it should initiate a call to Asterisk with the extension that indicates that the phone went off hook without specifying a specific line. In the examples in this document, for the station named "station1", this extension is simply named, "station1".

Asterisk will attempt to connect this station to the first available trunk that is not in use. Asterisk will check the trunks in the order that they were specified in the station entry in sla.conf. If all trunks are in use, the call will be denied.

If Asterisk is able to acquire an idle trunk for this station, then trunk is connected to the station and the station will hear dialtone. The station can then proceed to dial a number to call. As soon as a trunk is acquired, all appearances of this line on stations will show that the line is in use.

SLA Station goes off hook (ringing)

When a station goes off hook while it is ringing, it should simply answer the call that had been initiated to it to make it ring. Once the station has answered, Asterisk will figure out which trunk to connect it to. It will connect it to the highest priority trunk that is currently ringing. Trunk priority is determined by the order that the trunks are listed in the station entry in sla.conf.

SLA Line button on a station is pressed

When a line button is pressed on a station, the station should initiate a call to Asterisk with the extension that indicates which line button was pressed. In the examples given in this document, for a station named "station1" and a trunk named "line1", the extension would be "station1_line1".
If the specified trunk is not in use, then the station will be connected to it and will hear dialtone. All appearances of this trunk will then show that it is now in use.

If the specified trunk is on hold by this station, then this station will be reconnected to the trunk. The line appearance for this trunk on this station will now show in use. If this was the only station that had the call on hold, then all appearances of this trunk will now show that it is in use. Otherwise, all stations that are not currently connected to this trunk will show it on hold.

If the specified trunk is on hold by a different station, then this station will be connected to the trunk only if the trunk itself and the station(s) that have it on hold do not have private hold enabled. If connected, the appeareance of this trunk on this station will then show in use. All stations that are not currently connected to this trunk will show it on hold.

Short Message Service (SMS)

Information about Asterisk and SMS

Introduction to SMS

The SMS module for Asterisk was developed by Adrian Kennard, and is an implementation of the ETSI specification for landline SMS, ETSI ES 201 912, which is available from http://www.etsi.org. Landline SMS is starting to be available in various parts of Europe, and is available from BT in the UK. However, Asterisk would allow gateways to be created in other locations such as the US, and use of SMS capable phones such as the Magic Messenger. SMS works using analogue or ISDN lines.

SMS and extensions.conf

The following contexts are recommended.

```
; Mobile Terminated, RX. This is used when an incoming call from the SMS arrives
; with the queue (called number and sub address) in ${EXTEN}
; Running an app after receipt of the text allows the app to find all messages
; in the queue and handle them, e.g. email them.
; The app may be something like smsq --process=somecommand --queue=${EXTEN} to
; run a command for each received message
; See below for usage
[smsmtrx]
exten = _X.,1,SMS(${EXTEN},a)
exten = _X.,2,System("someapptohandleincomingsms ${EXTEN}")
exten = _X.,3,Hangup
;
; Mobile originated, RX. This is receiving a message from a device, e.g.
; a Magic Messenger on a sip extension
; Running an app after receipt of the text allows the app to find all messages
; in the queue and handle then, e.g. sending them to the public SMSC
; The app may be something like smsq --process=somecommand --queue=${EXTEN}
; to run a command for each received message
; See below for example usage
[smsmorx]
exten = _X.,1,SMS(${EXTEN},sa)
exten = _X.,2,System("someapptohandlelocalsms ${EXTEN}")
exten = _X.,3,Hangup
```

smsmtrx is normally accessed by an incoming call from the SMSC. In the UK this call is from a CLI of 080058752X0 where X is the sub address. As such a typical usage in the extensions.conf at the point of handling an incoming call is:

```
exten = _X./8005875290,1,Goto(smsmtrx,${EXTEN},1)
exten = _X./_80058752[0-8]0,1,Goto(smsmtrx,${EXTEN}-${CALLERID(num):8:1},1)
```

Alternatively, if you have the correct national prefix on incoming CLI, e.g. using dahdi_hfc, you might use:

```
exten = _X./08005875290,1,Goto(smsmtrx,${EXTEN},1)
exten = _X./_080058752[0-8]0,1,Goto(smsmtrx,${EXTEN}-${CALLERID(num):9:1},1)
```

smsmorx is normally accessed by a call from a local sip device connected to a Magic Messenger. It could however by that you are operating Asterisk as a message centre for calls from outside. Either way, you look at the called number and goto smsmorx. In the UK, the SMSC number that would be dialed is 1709400X where X is the caller sub address. As such typical usage in extension.config at the point of handling a call from a sip phone is:

```
exten = 17094009,1,Goto(smsmorx,${CALLERID(num)},1)
exten = _1709400[0-8],1,Goto(smsmorx,${CALLERID(num)}-{EXTEN:7:1},1)
```

SMS Background

Short Message Service (SMS), or texting is very popular between mobile phones. A message can be sent between two phones, and normally contains 160 characters. There are ways in which various types of data can be encoded in a text message such as ring tones, and small graphic, etc. Text messaging is being used for voting and competitions, and also SPAM...

Sending a message involves the mobile phone contacting a message centre (SMSC) and passing the message to it. The message centre then contacts the destination mobile to deliver the message. The SMSC is responsible for storing the message and trying to send it until the destination mobile is available, or a timeout.

Landline SMS works in basically the same way. You would normally have a suitable text capable landline phone, or a separate texting box such as a Magic Messenger on your phone line. This sends a message to a message centre your telco provides by making a normal call and sending the data using 1200 Baud FSK signaling according to the ETSI spec. To receive a message the message centre calls the line with a specific calling number, and the text capable phone answers the call and receives the data using 1200 Baud FSK signaling. This works particularly well in the UK as the calling line identity is sent before the first ring, so no phones in the house would ring when a message arrives.

SMS Delivery Reports

The SMS specification allows for delivery reports. These are requested using the srr bit. However, as these do not work in the UK yet they are not fully implemented in this application. If anyone has a telco that does implement these, please let me know. BT in the UK have a non standard way to do this by starting the message with *0#, and so this application may have a UK specific bodge in the near future to handle these.
The main changes that are proposed for delivery report handling are :

- New queues for sent messages, one file for each destination address and message reference.
- New field in message format, user reference, allowing applications to tie up their original message with a report.
- Handling of the delivery confirmation/rejection and connecting to the outgoing message - the received message file would then have fields for the original outgoing message and user reference allowing applications to handle confirmations better.

SMS File Formats

By default all queues are held in a director /var/spool/asterisk/sms. Within this directory are sub directories mtrx, mttx, morx, motx which hold the received messages and the messages ready to send. Also, /var/log/asterisk/sms is a log file of all messages handled.

The file name in each queue directory starts with the queue parameter to SMS which is normally the CLI used for an outgoing message or the called number on an incoming message, and may have -X (X being sub address) appended. If no queue ID is known, then 0 is used by smsq by default. After this is a dot, and then any text. Files are scanned for matching queue ID and a dot at the start. This means temporary files being created can be given a different name not starting with a queue (we recommend a . on the start of the file name for temp files). Files in these queues are in the form of a simple text file where each line starts with a keyword and an = and then data. udh and ud have options for hex encoding, see below.

UTF-8.

The user data (ud) field is treated as being UTF-8 encoded unless the DCS is specified indicating 8 bit format. If 8 bit format is specified then the user data is sent as is. The keywords are as follows:

- oa - Originating address The phone number from which the message came Present on mobile terminated messages and is the CLI for morx messages
- da - Destination Address The phone number to which the message is sent Present on mobile originated messages
- scts - The service centre time stamp Format YYYY-MM-DDTHH:MM:SS Present on mobile terminated messages
- pid - One byte decimal protocol ID See GSM specs for more details Normally 0 or absent
- dcs - One byte decimal data coding scheme If omitted, a sensible default is used (see below) See GSM specs for more details
- mr - One byte decimal message reference Present on mobile originated messages, added by default if absent
- srr - 0 or 1 for status report request Does not work in UK yet, not implemented in app_sms yet
- rp - 0 or 1 return path See GSM specs for details
- vp - Validity period in seconds Does not work in UK yet
- udh - Hex string of user data header prepended to the SMS contents, excluding initial length byte. Consistent with ud, this is specified as udh# rather than udh= If blank, this means that the udhi flag will be set but any user data header must be in the ud field
- ud - User data, may be text, or hex, see below

udh is specified as as udh# followed by hex (2 hex digits per byte). If present, then the user data header indicator bit is set, and the length plus the user data header is added to the start of the user data, with padding if necessary (to septet boundary in 7 bit format). User data can hold an USC character codes U+0000 to U+FFFF. Any other characters are coded as U+FEFF

ud can be specified as ud= followed by UTF-8 encoded text if it contains no control characters, i.e. only (U+0020 to U+FFFF). Any invalid UTF-8 sequences are treated as is (U+0080-U+00FF).

ud can also be specified as ud# followed by hex (2 hex digits per byte) containing characters U+0000 to U+00FF only.

ud can also be specified as ud## followed by hex (4 hex digits per byte) containing UCS-2 characters.

When written by app_sms (e.g. incoming messages), the file is written with ud= if it can be (no control characters). If it cannot, the a comment line ;ud= is used to show the user data for human readability and ud# or ud## is used.

SMS Sub Address

When sending a message to a landline, you simply send to the landline number. In the UK, all of the mobile operators (bar one) understand sending messages to landlines and pass the messages to the BTText system for delivery to the landline.

The specification for landline SMS allows for the possibility of more than one device on a single landline. These can be configured with Sub addresses which are a single digit. To send a message to a specific device the message is sent to the landline number with an extra digit appended to the end. The telco can define a default sub address (9 in the UK) which is used when the extra digit is not appended to the end. When the call comes in, part of the calling line ID is the sub address, so that only one device on the line answers the call and receives the message.

Sub addresses also work for outgoing messages. Part of the number called by the device to send a message is its sub address. Sending from the default sub address (9 in the UK) means the message is delivered with the sender being the normal landline number. Sending from any other sub address makes the sender the landline number with an extra digit on the end.

Using Asterisk, you can make use of the sub addresses for sending and receiving messages. Using DDI (DID, i.e. multiple numbers on the line on ISDN) you can also make use of many different numbers for SMS.

SMS Terminology

- SMS - Short Message Service i.e. text messages
- SMSC - Short Message Service Centre The system responsible for storing and forwarding messages
- MO - Mobile Originated A message on its way from a mobile or landline device to the SMSC
- MT - Mobile Terminated A message on its way from the SMSC to the mobile or landline device
- RX - Receive A message coming in to the Asterisk box
- TX - Transmit A message going out of the Asterisk box

SMS Typical Use with Asterisk

Sending messages from an Asterisk box can be used for a variety of reasons, including notification from any monitoring systems, email subject lines, etc.

Receiving messages to an Asterisk box is typically used just to email the messages to someone appropriate - we email and texts that are received to our direct numbers to the appropriate person. Received messages could also be used to control applications, manage competitions, votes, post items to IRC, anything.

Using a terminal such as a magic messenger, an Asterisk box could ask as a message centre sending messages to the terminal, which will beep and pop up the message (and remember 100 or so messages in its memory).

Using SMSq

smsq is a simple helper application designed to make it easy to send messages from a command line. it is intended to run on the Asterisk box and have direct access to the queue directories for SMS and for Asterisk.

In its simplest form you can send an SMS by a command such as smsq 0123456789 This is a test to 0123456789 This would create a queue file for a mobile originated TX message in queue 0 to send the text "This is a test to 0123456789" to 0123456789. It would then place a file in the /var/spool/asterisk/outgoing directory to initiate a call to 17094009 (the default message centre in smsq) attached to application SMS with argument of the queue name (0).

Normally smsq will queue a message ready to send, and will then create a file in the Asterisk outgoing directory causing Asterisk to actually connect to the message centre or device and actually send the pending message(s).

Using --process, smsq can however be used on received queues to run a command for each file (matching the queue if specified) with various environment variables set based on the message (see below); smsq options:

- --help Show help text
- --usage Show usage
- --queue -q Specify a specific queue In no specified, messages are queued under queue "0"
- --da -d Specify destination address
- --oa -o Specify originating address This also implies that we are generating a mobile terminated message
- --ud -m Specify the actual message
- --ud-file -f Specify a file to be read for the context of the message A blank filename (e.g. --ud-file= on its own) means read stdin. Very useful when using via ssh where command line parsing could mess up the message.
- --mt -t Mobile terminated message to be generated
- --mo Mobile originated message to be generated Default
- --tx Transmit message Default
- --rx -r Generate a message in the receive queue
- --UTF-8 Treat the file as UTF-8 encoded (default)
- --UCS-1 Treat the file as raw 8 bit UCS-1 data, not UTF-8 encoded
- --UCS-2 Treat the file as raw 16 bit bigendian USC-2 data
- --process Specific a command to process for each file in the queue Implies --rx and --mt if not otherwise specified. Sets environment variables for every possible variable, and also ud, ud8 (USC-1 hex), and ud16 (USC-2 hex) for each call. Removes files.
- --motx-channel Specify the channel for motx calls May contain X to use sub address based on queue name or may be full number Default is Local/1709400X
- --motx-callerid Specify the caller ID for motx calls The default is the queue name without -X suffix
- --motx-wait Wait time for motx call Default 10
- --motx-delay Retry time for motx call Default 1
- --motx-retries Retries for motx call Default 10
- --mttx-channel Specify the channel for mttx calls Default is Local/ and the queue name without -X suffix
- --mtttx-callerid Specify the callerid for mttx calls May include X to use sub address based on queue name or may be full number Default is 080058752X0
- --mttx-wait Wait time for mttx call Default 10
- --mttx-delay Retry time for mttx call Default 30
- --mttx-retries Retries for mttx call Default 100
- --default-sub-address The default sub address assumed (e.g. for X in CLI and dialled numbers as above) when none added (-X) to queue Default 9
- --no-dial -x Create queue, but do not dial to send message
- --no-wait Do not wait if a call appears to be in progress This could have a small window where a message is queued but not sent, so regular calls to smsq should be done to pick up any missed messages
- --concurrent How many concurrent calls to allow (per queue), default 1
- --mr -n Message reference
- --pid -p Protocol ID
- --dcs Data coding scheme
- --udh Specific hex string of user data header specified (not including the initial length byte) May be a blank string to indicate header is included in the user data already but user data header indication to be set.
- --srr Status report requested
- --rp Return path requested
- --vp Specify validity period (seconds)
- --scts Specify timestamp (YYYY-MM-DDTHH:MM:SS)
- --spool-dir Spool dir (in which sms and outgoing are found) Default /var/spool/asterisk

Other arguments starting '' or '' are invalid and will cause an error. Any trailing arguments are processed as follows:

- If the message is mobile originating and no destination address has been specified, then the first argument is assumed to be a destination address
- If the message is mobile terminating and no destination address has been specified, then the first argument is assumed to be the queue name
- If there is no user data, or user data file specified, then any following arguments are assumed to be the message, which are concatenated.
- If no user data is specified, then no message is sent. However, unless --no-dial is specified, smsq checks for pending messages and generates an outgoing anyway

When smsq attempts to make a file in /var/spool/asterisk/outgoing, it checks if there is already a call queued for that queue. It will try several filenames, up to the --concurrent setting. If these files exist, then this means Asterisk is already queued to send all messages for that queue, and so Asterisk should pick up the message just queued. However, this alone could create a race condition, so if the files exist then smsq will wait up to 3 seconds to confirm it still exists or if the queued messages have been sent already. The --no-wait turns off this behaviour. Basically, this means that if you have a lot of messages to send all at once, Asterisk will not make unlimited concurrent calls to the same message centre or device for the same queue. This is because it is generally more efficient to make one call and send all of the messages one after the other.

smsq can be used with no arguments, or with a queue name only, and it will check for any pending messages and cause an outgoing if there are any. It only sets up one outgoing call at a time based on the first queued message it finds. A outgoing call will normally send all queued messages for that queue. One way to use smsq would be to run with no queue name (so any queue) every minute or every few seconds to send pending message. This is not normally necessary unless --no-dial is selected. Note that smsq does only check motx or mttx depending on the options selected, so it would need to be called twice as a general check.

UTF-8 is used to parse command line arguments for user data, and is the default when reading a file. If an invalid UTF-8 sequence is found, it is treated as UCS-1 data (i.e, as is). The --process option causes smsq to scan the specified queue (default is mtrx) for messages (matching the queue specified, or any if queue not specified) and run a command and delete the file. The command is run with a number of environment variables set as follows. Note that these are unset if not needed and not just taken from the calling environment. This allows simple processing of incoming messages

- - $queue Set if a queue specified $?srr srr is set (to blank) if srr defined and has value 1. $?rp rp is set (to blank) if rp defined and has value 1. $ud User data, UTF-8 encoding, including any control characters, but with nulls stripped out Useful for the content of emails, for example, as it includes any newlines, etc. $ude User data, escaped UTF-8, including all characters, but control characters \n, \r, \t, \f, \xxx and \ is escaped as

Useful guaranteed one line printable text, so useful in Subject lines of emails, etc $ud8 Hex UCS-1 coding of user data (2 hex digits per character) Present only if all user data is in range U+0000 to U+00FF $ud16 Hex UCS-2 coding of user data (4 hex digits per character) other Other fields set using their field name, e.g. mr, pid, dcs, etc. udh is a hex byte string

Voicemail

All things voicemail

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 odbcstorage variable in the general section of voicemail.conf.

You may modify the voicemessages table name by using odbctable=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 agains -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.

Asterisk SIP Connections

Fishook: We need a single spot to compile the definitive guide to Asterisk's SIP capabilities.

Content coming. WIP beginning here:
1.8 SIP Scratchpad

Asterisk GUI

Introduction to Asterisk GUI

Asterisk GUI is a framework for the creation of graphical interfaces for configuring Asterisk. Some sample graphical interfaces for specific vertical markets are included for reference or for actual use and extension.

Software License

Asterisk GUI HTML and Javascript files Copyright (C) 2006-2011 Digium, Inc.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 2 only. This software is also available under commercial terms from Digium, Inc.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Please contact Digium for information on alternative licensing arrangements for Asterisk GUI.

Download

While package release is inconsistent and infrequent, you can always get a current copy of Asterisk GUI from subversion. The current stable version will always be under `branches` and is currently located in `branches/2.0`.

Support

Please note that Asterisk GUI is not officially supported, though bugs, patches, and feature requests may be submitted at http://issues.asterisk.org and should reference the Asterisk GUI project. You may also find peer support in the #asterisk-gui IRC channel and the Asterisk GUI forum.

Installation and Configuration

Installation

```
$ ./configure
$ make
$ make install
$ make checkconfig
```

Configuration

You may install sample configuration files by doing "make samples". Also you will need to edit your Asterisk configuration files to enable Asterisk GUI properly, specifically:

http.conf:

Enable http.

```
[general]
enabled=yes
enablestatic=yes
#bindaddr=0.0.0.0 # allow GUI to be accessible from all IP addresses.
bindaddr=127.0.0.1 # require access from the machine Asterisk is running on
bindport=8088
```

manager.conf

Enable manager access.

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

Create an appropriate entry in `manager.conf` for the administrative user (PLEASE READ THE security.txt FILE!)

```
[admin]
secret = thiswouldbeaninsecurepassword
read = system,call,log,verbose,command,agent,config,read,write,originate
write = system,call,log,verbose,command,agent,config,read,write,originate
```

Access

Access Asterisk GUI via a URL formatted in the following way, where $IP is the IP address on which both Asterisk and Asterisk GUI are installed, $PORT is `bindport` from `http.conf`, and $PREFIX is the `prefix` from `http.conf`, and it can be omitted if blank.

```
http://$IP:$PORT/$PREFIX/static/config/index.html
```

Troubleshooting

Check your filesystem permissions:

```
$ chown -R asterisk:asterisk /etc/asterisk/ /var/lib/asterisk /usr/share/asterisk # if
asterisk runs as the user "asterisk"
$ chmod 644 /etc/asterisk/*
```

Check that the bindaddr value in `/etc/asterisk/http.conf` matches the IP address of the machine you're using to **access** Asterisk GUI, not necessarily the IP address Asterisk GUI is running on.

Check on the Asterisk CLI that Asterisk is receiving the values you've set.

```
http show status
manager show settings
```

Output should look like this:

```
amelia*CLI> http show status
HTTP Server Status:
Prefix: /asterisk
Server Enabled and Bound to 0.0.0.0:8088

Enabled URI's:
/asterisk/httpstatus => Asterisk HTTP General Status
/asterisk/phoneprov/... => Asterisk HTTP Phone Provisioning Tool
/asterisk/amanager => HTML Manager Event Interface w/Digest authentication
/asterisk/arawman => Raw HTTP Manager Event Interface w/Digest authentication
/asterisk/manager => HTML Manager Event Interface
/asterisk/rawman => Raw HTTP Manager Event Interface
/asterisk/static/... => Asterisk HTTP Static Delivery
/asterisk/amxml => XML Manager Event Interface w/Digest authentication
/asterisk/mxml => XML Manager Event Interface

Enabled Redirects:
  None.
amelia*CLI> manager show settings

Global Settings:
----------------
  Manager (AMI):             Yes
  Web Manager (AMI/HTTP):    Yes
  TCP Bindaddress:           0.0.0.0:5038
  HTTP Timeout (minutes):    60
  TLS Enable:                No
  TLS Bindaddress:           Disabled
  TLS Certfile:              asterisk.pem
  TLS Privatekey:
  TLS Cipher:
  Allow multiple login:      Yes
  Display connects:          Yes
  Timestamp events:          No
  Channel vars:
  Debug:                     No
  Block sockets:             No
```

Check that the ports you've specified are open by using telnet from another computer.

```
telnet $ASTERISK_SERVER_IP_ADDRESS 5038
telnet $ASTERISK_SERVER_IP_ADDRESS 8088
```

Check that the `dahdi_genconf` script runs correctly and creates `/etc/asterisk/dahdi_guiread.conf`.

Check the existence of `/etc/asterisk/guipreferences.conf` and inside that file, the existence of the following line:

```
config_upgraded = yes
```

Check the last modified date of `/etc/asterisk/http.conf`. Asterisk GUI updates the timestamp on this file every time it is loaded. If the timestamp is not getting updated, your HTTP request is either not making it to Asterisk or it is not being processed correctly by Asterisk. This indicates a configuration error.

Check that the user Asterisk runs as has a login shell. Asterisk GUI depends on Asterisk being able to use the System application.

```
su - asterisk
```

If you installed Asterisk GUI via 'yum' or 'apt-get', you may need to symlink `/var/lib/asterisk/static-http` to `/usr/share/asterisk/static-http`. Unfortunately `/usr/share/asterisk/static-http` will already exist, but fortunately it does not contain any useful files.

```
ls /var/lib/asterisk/static-http/config && rm -rf /usr/share/asterisk/static-http && ln
-s /var/lib/asterisk/static-http /usr/share/asterisk/static-http
```

Development

Debugging

To turn on debug messages, open `config/js/session.js`. On line 30, set "log" to "true":

```
log: true, /**< boolean toggling logging */
```

Hide Menu Categories

Hide menu categories by changing the HTML class attribute to "AdvancedMode" in index.html. Show them by enabling Advanced Options in the GUI.

Historical Pages

What is a Historical Page?

A Historical Page is old documentation that no longer applies to the current version of Asterisk. Ways of doing things have changed, perhaps, or a particular bit of functionality is no longer available / supported / recommended.

Jabber in Asterisk

This information applies to Asterisk 10 and earlier versions. XMPP Support for Asterisk 11 has been completely rewritten

XMPP (Jabber) is an XML based protocol primarily for presence and messaging. It is an open standard and there are several open server implementations, such as ejabberd, jabberd(2), openfire, and others, as well as several open source clients, Psi, gajim, gaim etc. XMPP differs from other IM applications as it is immensly extendable. This allows us to easily integrate Asterisk with XMPP. The Asterisk XMPP Interface is provided by res_jabber.so.

`res_jabber` allows for Asterisk to connect to any XMPP (Jabber) server and is also used to provide the connection interface for `chan_jingle` and `chan_gtalk`.

Functions (`JABBER_STATUS`, `JABBER_RECEIVE`) and applications (`JabberSend`) are exposed to the dialplan.

You'll find examples of how to use these functions/applications hereafter. We assume that 'asterisk-xmpp' is properly configured in `jabber.conf`.

`JabberSend`

`JabberSend` sends an XMPP message to a buddy. Example:

extensions.ael

```
context default {
 _XXXX => {
       JabberSend(asterisk-xmpp,buddy@gmail.com,${CALLERID(name)} is calling ${EXTEN});
       Dial(SIP/${EXTEN}, 30);
       Hangup();
 }
}
```

`JABBER_STATUS`

As of Asterisk 1.6.0, the corresponding application `JabberStatus` is still available, but marked as deprecated in favor of this function.

`JABBER_STATUS` stores the status of a buddy in a dialplan variable for further use. Here is an AEL example of how to use it:

extensions.ael

```
1234 => {
 Set(STATUS=${JABBER_STATUS(asterisk-xmpp,buddy@gmail.com)});
 if (${STATUS}=1) {
  NoOp(User is online and active, ring his Gtalk client.);
  Dial(Gtalk/asterisk-xmpp/buddy@gmail.com);
 } else {
  NoOp(Prefer the SIP phone);
  Dial(SIP/1234);
 }
}
```

`JABBER_RECEIVE`

`JABBER_RECEIVE` waits (up to X seconds) for a XMPP message and returns its content. Used along with `JabberSend` (or `SendText`, provided it's implemented in the corresponding channel type), `JABBER_RECEIVE` helps Asterisk interact with users while calls flow through the dialplan.

`JABBER_RECEIVE`/`JabberSend` are not tied to the XMPP media modules `chan_gtalk` and `chan_jingle`, and can be used anywhere in the dialplan. In the following example, calls targeted to extension 1234 (be it accessed from SIP, DAHDI or whatever channel type) are controlled by user bob@domain.com. Asterisk notifies him that a call is coming, and asks him to take an action. This dialog takes place over an XMPP chat.

extensions.ael

```
context from-ext {
 1234 => {
  Answer();
  JabberSend(asterisk-xmpp,bob@jabber.org,Call from $CALLERID(num) - choose an option to
process the call);
  JabberSend(asterisk-xmpp,bob@jabber.org,1 : forward to cellphone);
  JabberSend(asterisk-xmpp,bob@jabber.org,2 : forward to work phone);
  JabberSend(asterisk-xmpp,bob@jabber.org,Default action : forward to your voicemail);
  Set(OPTION=${JABBER_RECEIVE(asterisk-xmpp,bob@jabber.org,20)});
  switch (${OPTION}) {
   case 1:
    JabberSend(asterisk-xmpp,bob@jabber.org,(Calling cellphone...);
    Dial(SIP/987654321);
    break;
   case 2:
    JabberSend(asterisk-xmpp,bob@jabber.org,(Calling workphone...);
    Dial(SIP/${EXTEN});
    break;
   default:
    Voicemail(${EXTEN}|u)
  }
 }
}
```

When calling from a GoogleTalk or Jingle client, the `CALLERID(name)` is set to the XMPP id of the caller (i.e. his JID). In the following example, Asterisk chats back with the caller identified by the caller id. We also take advantage of the `SendText` implementation in `chan_gtalk` (available in `chan_jingle`, and `chan_sip` as well), to allow the caller to establish SIP calls from his GoogleTalk client:

extensions.ael

```
context gtalk-in {
 s => {
   NoOp(Caller id : ${CALLERID(all)});
   Answer();
   SendText(Please enter the number you wish to call);
   Set(NEWEXTEN=${JABBER_RECEIVE(asterisk-xmpp,${CALLERID(name)})});
   SendText(Calling ${NEWEXTEN} ...);
   Dial(SIP/${NEWEXTEN);
   Hangup();
 }
}
```

The maintainer of `res_jabber` is Philippe Sultan <philippe.sultan@gmail.com>.

Old Calling using Google

Note that Google's changes to their Google Voice system have rendered the functionality of chan_gtalk and res_jabber unreliable. Additionally, ongoing maintenance of of chan_gtalk and res_jabber for Asterisk versions prior to Asterisk 11 is not provided by Digium and is instead in the charge of the community. For more information, please see the current page Calling using Google, where things have changed, as well as 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_gtalk Channel Driver and the res_jabber Resource module. Before proceeding, please ensure that both are compiled and part of your installation. Compilation of res_jabber and chan_gtalk 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 1.8.1.1 or greater. The 1.6.x versions of Asterisk only support calls made using the legacy GoogleTalk external client.

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.

Gtalk configuration

The chan_gtalk Channel Driver is configured with the gtalk.conf configuration file, typically located in /etc/asterisk. What follows is the minimum required configuration for successful operation.

Minimum Gtalk Configuration

```
[general]
context=local
allowguest=yes
bindaddr=0.0.0.0
externip=216.208.246.1

[guest]
disallow=all
allow=ulaw
context=local
connection=asterisk
```

This general section of this configuration specifies several items.

1. That calls will terminate to or originate from the **local** context; context=local
2. That guest calls are allowd; allowguest=yes
3. That RTP sessions will be bound to a local address (an IPv4 address must be present); bindaddr=0.0.0.0
4. (optional) That your external (the one outside of your NAT) IP address is defined; externip=216.208.246.1

The guest section of this configuration specifies several items.

1. That no codecs are allowed; disallow=all
2. That the ulaw codec is explicitly allowed; allow=ulaw
3. That calls received by the guest user will be terminated into the **local** context; context=local
4. That the Jabber connection used for guest calls is called "asterisk;" connection=asterisk

Jabber Configuration

The res_jabber Resource is configured with the jabber.conf configuration file, typically located in /etc/asterisk. What follows is the minimum required configuration for successful operation.

Minimum Jabber Configuration

```
[general]
autoregister=yes

[asterisk]
type=client
serverhost=talk.google.com
username=your_google_username@gmail.com/Talk
secret=your_google_password
port=5222
usetls=yes
usesasl=yes
statusmessage="I am an Asterisk Server"
timeout=100
```

The general section of this configuration specifies several items.

1. Debug mode is enabled, so that XMPP messages can be seen on the Asterisk CLI; debug=yes
2. Automated buddy pruning is disabled, otherwise buddies will be automatically removed from your list; autoprune=no
3. Automated registration of users from the buddy list is enabled; autoregister=yes

The asterisk 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/resource, where our resource is "Talk;" username=your_google_username@gmail.com/Talk
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. TLS encryption is required by Google; usetls=yes
7. Simple Authentication and Security Layer (SASL) is used by Google; usesasl=yes
8. We set a status message so other Google chat users can see that we're an Asterisk server; statusmessage="I am an Asterisk Server"
9. We set a timeout for receiving message from Google that allows for plenty of time in the event of network delay; timeout=100

Phone configuration

Now, let's place a phone into the same context as the Google calls. 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:

```
exten => s,1,Answer()
exten => s,n,Wait(2)
exten => s,n,SendDTMF(1)
exten => s,n,Dial(SIP/malcolm,20)
```

Note that you might have to adjust the "Wait" time from 2 (in seconds) to something larger, like 8, depending on the current mood of Google. Otherwise, your incoming calls might not be successfully picked up.

This example uses the "s" unmatched extension, because Google does not forward any DID when it sends the call to Asterisk.

In this example, we're Answering the call, Waiting 2 seconds, 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 the D dial option. 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. 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,Set(crazygooglecid=${CALLERID(name)})
exten => s,n,Set(stripcrazysuffix=${CUT(crazygooglecid,@,1)})
exten => s,n,Set(CALLERID(all)=${stripcrazysuffix})
exten => s,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(gtalk/asterisk/mybuddy@gmail.com)
```

Where the technology is "gtalk," the dialing peer is "asterisk" as defined in jabber.conf, and the dial string is the Google account name.

Outgoing calls made to Google Voice take the form of:

```
exten => _1XXXXXXXXXX,1,Dial(gtalk/asterisk/+${EXTEN}@voice.google.com)
```

Where the technology is "gtalk," the dialing peer is "asterisk" as defined in jabber.conf, and the dial string is a full E.164 number (plus character followed by country code, followed by the rest of the digits).

Interactive Voice with Text Response (IVTR)

Because the Google Talk web client provides both audio and text interface, you can use it to provide a text-based way of traversing Interactive Voice Response (IVR) menus. This is necessary since the client does not have any DTMF inputs.

In the following dialplan example, we will answer the call, wait a bit, send some text that will show up in the caller's Google Talk client, play back a prompt, capture the caller's text-based response, and then dial the appropriate SIP device.

```
exten => s,1,Answer()
exten => s,n,SendText("If you know the extension of the party you wish to reach, dial it now.")
exten => s,n,Background(if-u-know-ext-dial)
exten => s,n,Set(OPTION=${JABBER_RECEIVE(asterisk,${CALLERID(name)::15},5)})
exten => s,n,Dial(SIP/${OPTION},20)
```

Note that with the JABBER_RECEIVE function, we're receiving the text from **asterisk** which we defined earlier in this page as our connection to Google. We're also specifying with **${CALLERID(name)::15}** that we want to strip off the last 15 characters from the CallerID name string - which is the number of characters that Google is appending, as of this writing, to represent an internal call ID number, and that we want to wait **5** seconds for a response.

Webinar

A Webinar was conducted on Tuesday, March 24, 2011 detailing Asterisk configuration for calling using Google as well as several usage cases. A copy of the slides, in PDF format, is available here - Google Calling Webinar - Public.pdf

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.

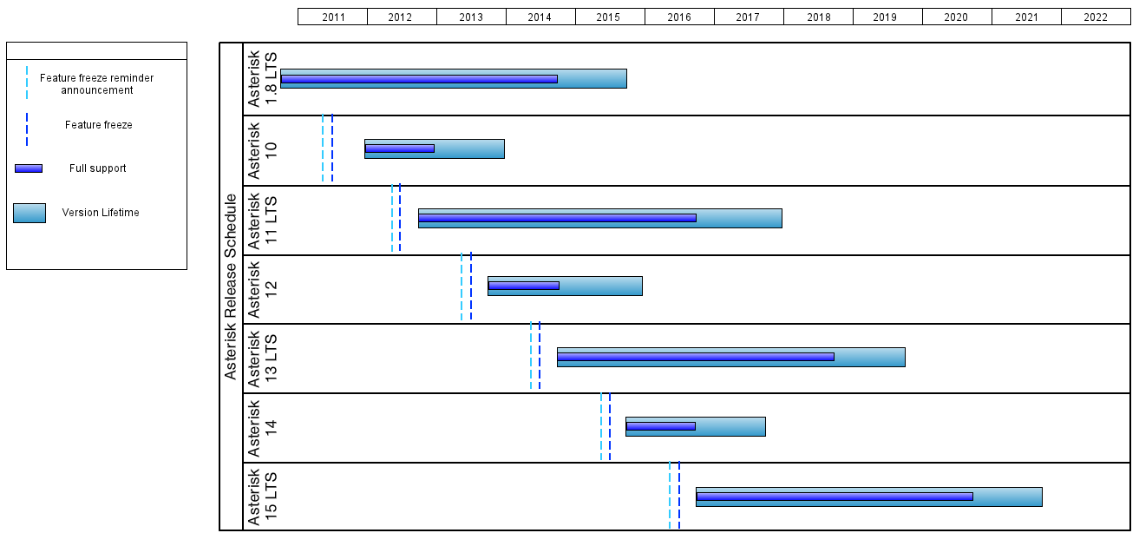
The following table shows the release time lines for all releases of Asterisk, including those that have reached End of Life.

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	2013-10 (tentative)	2014-10 (tentative)	2015-10 (tentative)
13.x	LTS	2014-10 (tentative)	2018-10 (tentative)	2019-10 (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 Announcement Reminder	3rd Wednesday of May
Feature Freeze of Asterisk Branch	3rd Wednesday of July
First Release of Asterisk from Branch	3rd Wednesday of October

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. Issues reported against these modules may have a low level of support. Some extended modules have active community developers monitoring issues though.

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

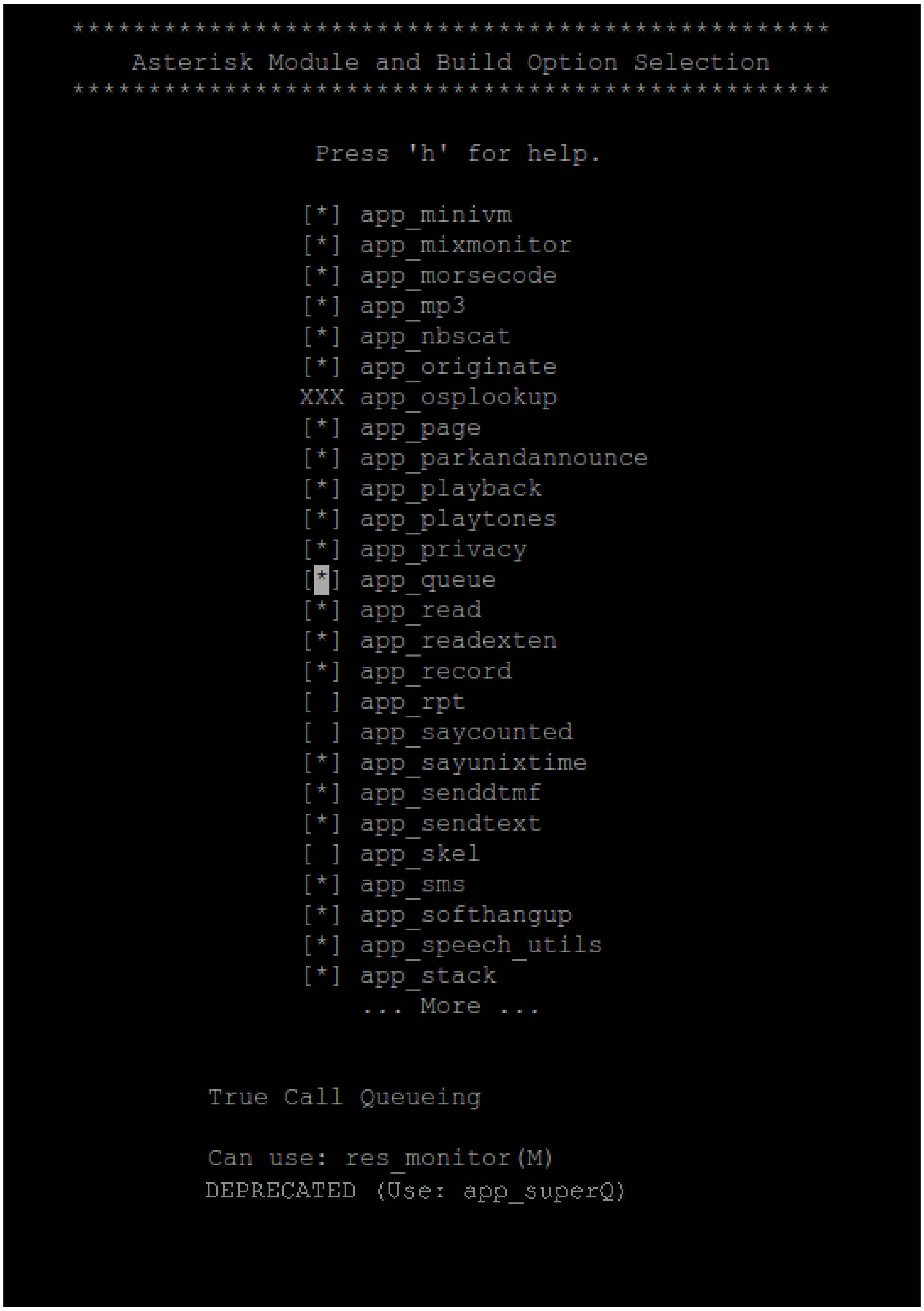
On top of the existing tags, we would add two additional tags to communicate via menuselect that a module was extended or deprecated (and what module supersedes the deprecated module). These tags include:

- **<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

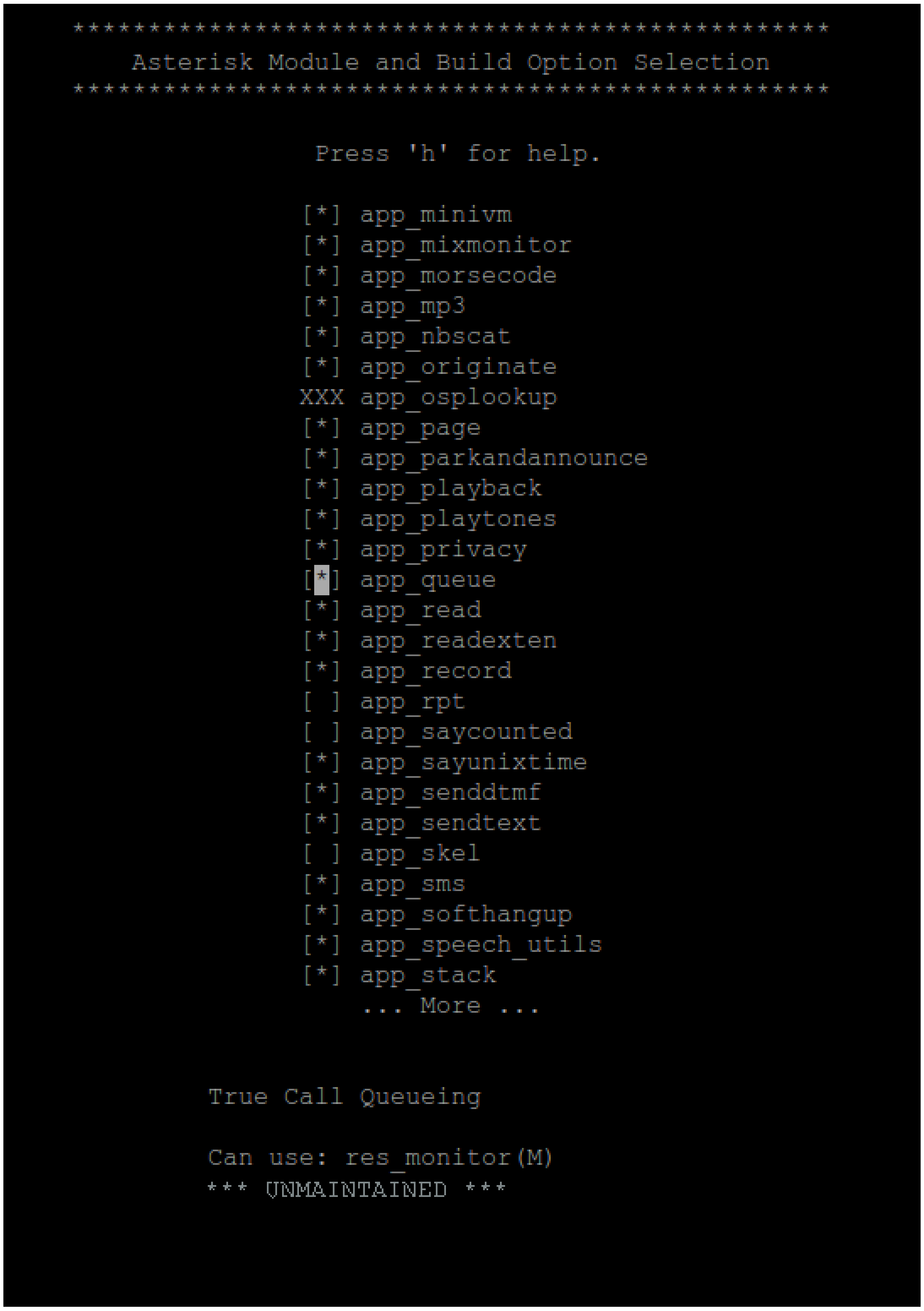
The following two images show the suggested menuselect output based on the addition of the *<support_level>* and *<replacement>* tags. This would be a new line that has not been used before, and therefore would be added to menuselect as that new line.

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**).



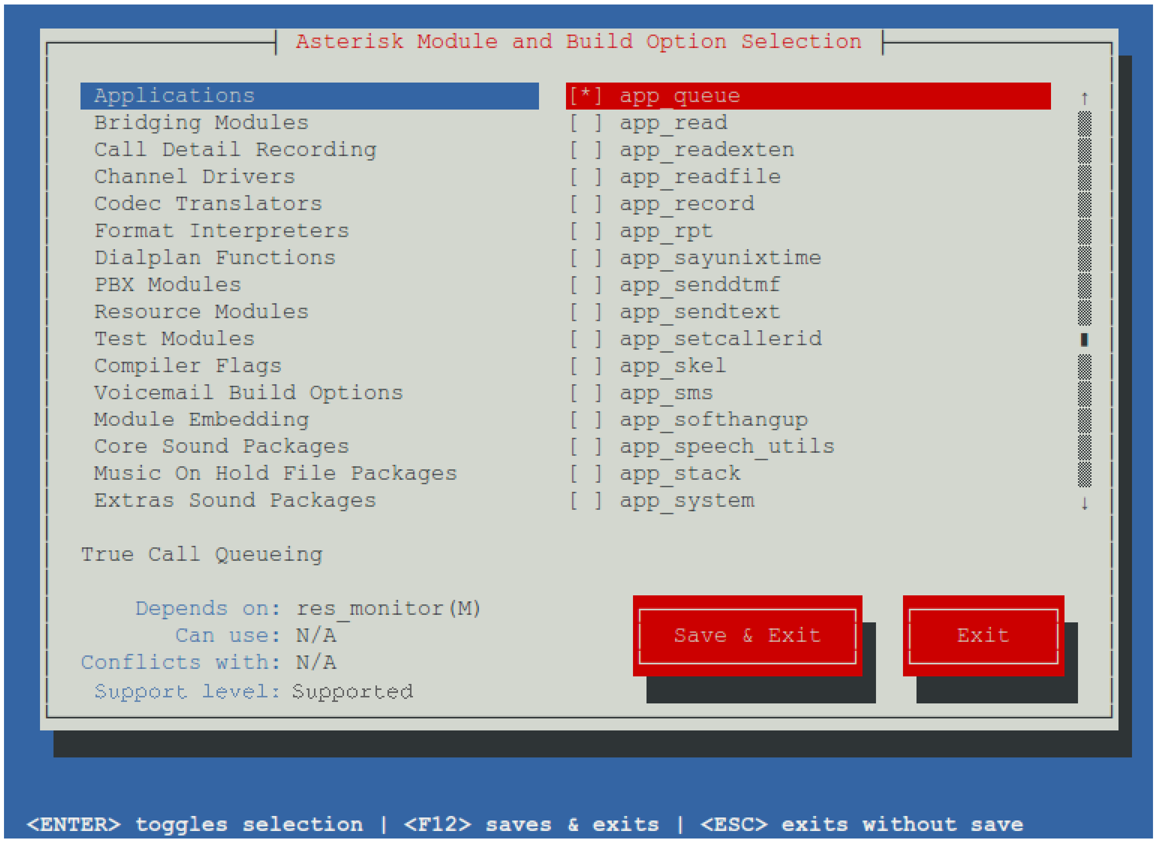
```
    ***************************************************
        Asterisk Module and Build Option Selection
    ***************************************************

                    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.



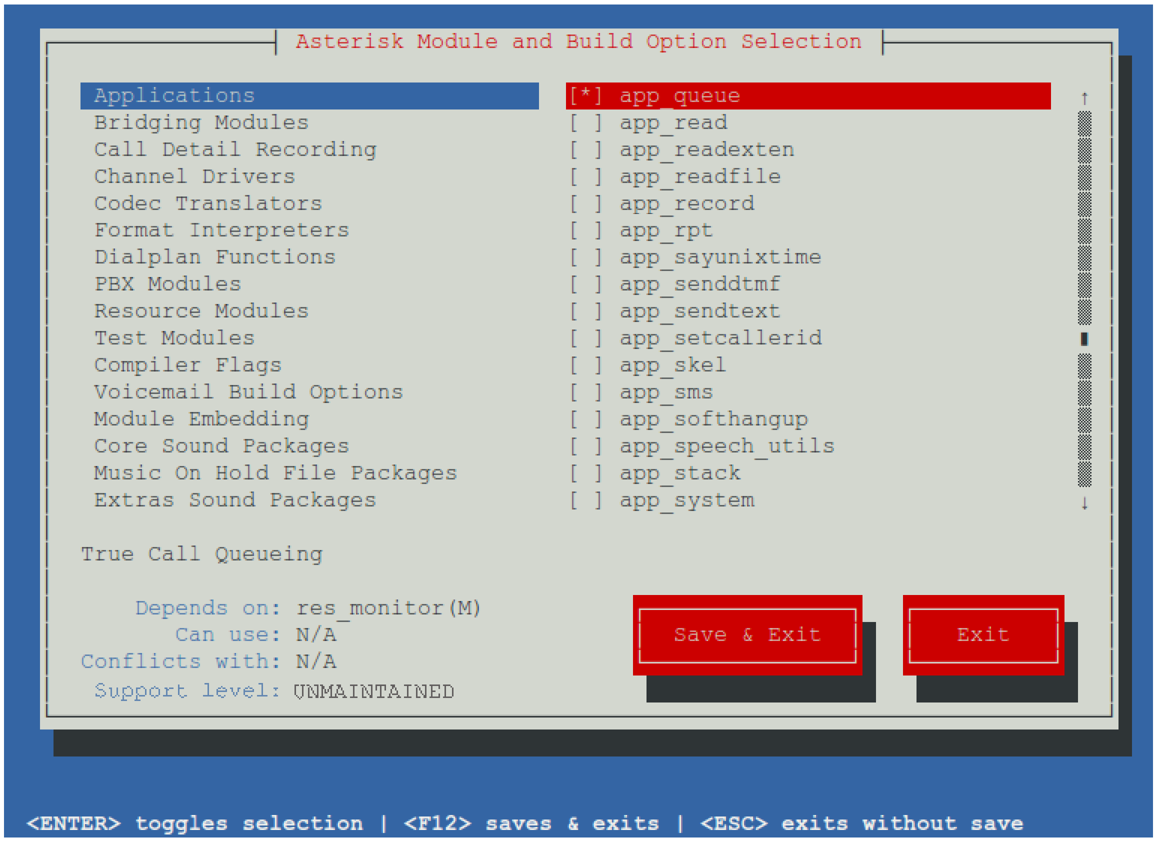
```
**************************************************
    Asterisk Module and Build Option Selection
**************************************************

                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:



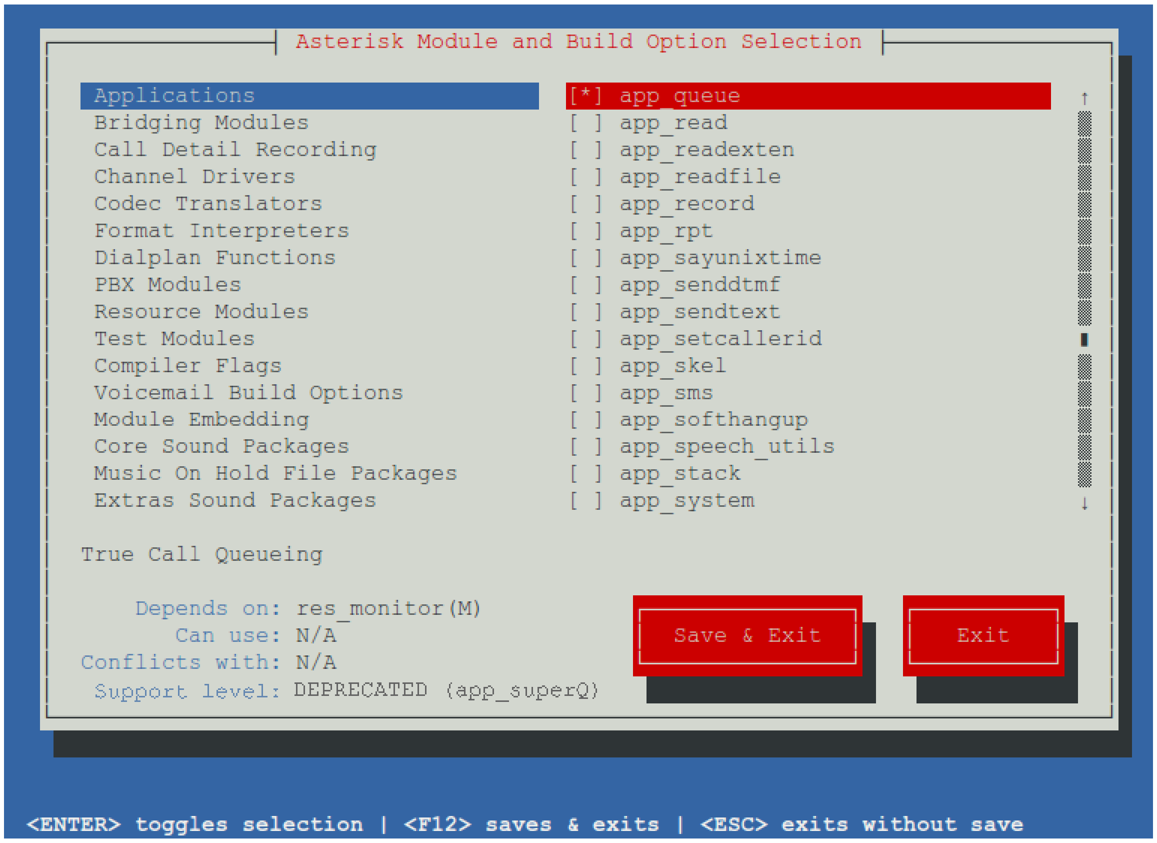
```
Asterisk Module and Build Option Selection

Applications                          [*] app_queue
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: Supported

<ENTER> toggles selection | <F12> saves & exits | <ESC> exits without save
```

Display in menuselect-newt for extended modules:



```
Asterisk Module and Build Option Selection

Applications                          [*] app_queue
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: UNMAINTAINED

<ENTER> toggles selection | <F12> saves & exits | <ESC> exits without save
```

Display in menuselect-newt for deprecated modules:



```
Asterisk Module and Build Option Selection

Applications                       [*] app_queue
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)
```

```
<ENTER> toggles selection | <F12> saves & exits | <ESC> exits without save
```

Asterisk Issue Guidelines

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 not accepted through the issue tracker and in general 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 or IRC channels 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 http://www.asterisk.org/community
- **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
- **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

Why should you read this?

To get the attention of helpful individuals and the Asterisk development team, you'll want to read, understand and act on this article. The steps here will help you provide all the information the Asterisk team needs to solve your issue.

How to report a bug

STOP! BEFORE YOU EVEN THINK ABOUT FILING A BUG REPORT...

Your issue may not be a bug or could have been fixed already. Run through the checklist below to verify you have done your due diligence.

- **Verify you are on a supported version of Asterisk:** https://wiki.asterisk.org/wiki/display/AST/Asterisk+Versions, i.e. a version of Asterisk that has not reached End of Life
- **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 point release. http://svnview.digium.com/svn/asterisk/tags/
- **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?** The documentation is located here: https://wiki.asterisk.org/wiki/display/AST/Home
- **Have you searched the Asterisk bug tracker to see if an issue is already filed for this potential bug?** https://issues.asterisk.org/jira you'll find the search field in the top right of the page.
- **Can you reproduce the problem?** If you can't find a way to simulate or reproduce the issue, then at least get the system to a point where it's capturing relevant debug during the times the seemingly random failure occurs. Yes that could mean running debug for days or weeks if necessary.

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.

Submitting the bug report - information requirements

You'll report the issue through the tracker, under the Asterisk project. https://issues.asterisk.org/jira/browse/ASTERISK

0. Sign up for an asterisk.org account

Account signup for asterisk.org community services (including JIRA, Confluence and FishEye/Crucible) can be found at https://signup.asterisk.org.

1.Create a new issue in the tracker

The "Create Issue" button in the top right hand corner of the page will prompt you for the project and issue type. Pick the Asterisk project, and choose an appropriate issue type.

Issue types:

- Bug
- New feature
- Improvement
- Information request (This type is not currently used, and your issue is likely to be closed or redefined)

2. 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'll also need SIP traces for anything involving a SIP channel plus backtraces and valgrind output for crashes or memory issues. Unless the issue is extremely trivial, you'll need to also include an Asterisk debug log, including DEBUG and VERBOSE type messages with at least a level of 5. You can find instructions HERE and HERE.

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.

3. 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 above and include all information that we require. Watch for emails where we may ask for additional data and help us test any patches or possible fixes for the issue. If you didn't follow the information requirements - it'll often be the first thing we ask for. A developer **cannot** fix your issue until we have sufficient data to verify the problem.

Reasons your report may be closed without resolution

If your bug is determined to be a configuration error or something that is clearly not reproducible, it may be closed immediately by a bug marshal. If you believe this to be in error, please go in and edit the bug and say so, or comment to the bug marshal that made the closure (visible on the ticket itself.) If that fails, bring it up on the mailing list(s) and it will be sorted out by the community.

Common reasons your issue may be closed without resolution:

- The issue can't be reproduced from the original report.
- The report lacks sufficient information to investigate, and you haven't responded to our requests for specific information.
- The issue may exist, but can't be fixed due to underlying architecture or infrastructure issues.
- The issue is not a bug, but is a support request. The reporter will be directed to community resources for support.

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 SVN 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. However just writing a patch doesn't guarantee it'll make it into an Asterisk release. See the paragraphs below on Patch and Code submission.

Do you have a feature request and a patch to go with it?

Great, then move on to the Patch and Code submission section below.

Patch and Code submission

Basic patch requirements:

- Make sure you read the coding guidelines and make your code compliant with these.
- Add comments that explain your code in Doxygen format (Qt style) to make it easier to understand both for those that commit and for other developers and users.
- Read the Digium License Agreement section below, and then sign the agreement within JIRA.

Format of patch: Please use "diff -u" or "svn diff" on all your patches. Patches which include alternate formatting are almost certainly going to be thrown out or ignored; there are too few hours in the day to wade through difficult-to-follow C code fixes without the help of "diff -u".

Features and Improvements

We all love new features, but new features bring new complexities, and new complexities bring new bugs. So, to help cut down on new bugs being introduced with your new features, please consider as much of what you're touching as possible. In other words: everything is linked in very subtle ways; make sure you look very closely at each piece of what your patch influences before you submit it. Don't be afraid to submit revised copies of the patch if that's what's required. Bug marshals are there to help you and their suggestions are based on their experience of working with Asterisk.

Once you have a patch that follows the requirements noted above then you can post it on the issue tracker in the Asterisk project as type "Improvement" or "New Feature".

Patch testing & Review board

After you create a new feature or bug patch submission, it will need to be tested before it can be committed to the SVN repository. By "tested", we mean that multiple persons other than yourself should patch/run/abuse the patch in various manners so that all possible variations of input/output are run through the test. In instances where there are limited test environments, please document exactly how and why you were unable to get others to test the patch so that the developers know that this simply isn't "waiting for testing" and is in fact ready to go. Insufficient testing is a sure way to have your feature put on the back burner - the SVN maintainers don't have time to run exhaustive tests on each new feature. Find people on the #asterisk or #asterisk-dev IRC channels who may be able to assist you in testing, and develop a working relationship with other Asterisk users so that you can swap testing routines; this greatly speeds the process. Mail the mailing lists, informing other Asterisk users about your patch and ask for feedback.

If you have submitted code in the past and are great at following the coding guidelines, then you may want to request access to Review Board. You can find the instructions for review board here. Code submitted to Review Board is likely to be included much faster than an untested patch just sitting in the tracker. The development team will provide access to Review Board for those that submit code often, write quality code and have a history of following the coding guidelines.

Who commits the code into Asterisk?

There are two ways code gets pushed into Asterisk.

1. Patches can always be merged by a member of the Asterisk Developer Community. Folks with commit access are those who have shown a willingness to work with the review process and are trusted shepherds of the project. Anyone with commit access can take ownership of a proposed patch and work to get it included in trunk - hence why the asterisk-dev mailing list is the best place to discuss patches.
2. Digium works through the queue of all issues on the tracker, both bugs and improvements. There's typically hundreds of issues in the backlog, including the work that the Asterisk Developer Community commits to at AstriDevCon every year. As such, we try to address as much as we can but there's no guarantee that we will get to any one issue.

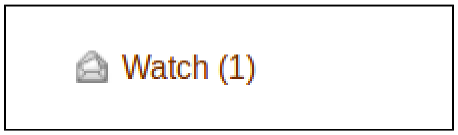
What is the status of my issue?

The status of all issues in the tracker are reflected in the tracker. For any particular issue you'll want to look at the "Status" field and the comments tab under "Activity".

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.

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 to avoid cluttering up the main project. The clone being created does not indicate any particular status.

All comments, work, and anything of relevance to the patch or issue is done in the ASTERISK project

Why did my feature not make it into Asterisk?

Many times, a feature that might sound good to you and might work as expected just isn't important enough to make it into the system if it isn't deemed useful to a large enough audience. This is, of course, a subjective decision by the SVN maintainers and the bug marshals. Their opinions can be swayed if enough users get together and rally around a particular feature that seems to be useful to them, so getting others to chime in on your neat request will perhaps advance it further towards the top of the queue. Sometimes, it's just a matter of timing - nobody has had the time to look at it yet. It's perfectly legitimate (and good) for someone who has a bug or feature request to go to IRC and lobby for it with the bug marshal to push it through, schedule time, etc.

Digium Submission License Agreement

If you are going to contribute code to the Asterisk project, then please thoroughly read and then electronically sign the Digium Open Source Software Project Submission Agreement upon logging in to the Asterisk issue tracker. You'll see a "Sign a License Agreement" link at the top of the JIRA interface.

The gist of the agreement is that your contribution must not introduce any encumbrance to the Asterisk code base; Digium does not own your contribution, and they cannot take released Asterisk out of GPL. Relax; it's a very fair and reasonable license, and does not remove your rights or threaten the open source nature of the project. See the mailing list archives for long explanations of why everyone who contributes agrees that it's a fair and sane thing to do. You only need to sign the agreement once; it applies to all code that you send in via the issue tracker.

The issue tracker keeps a record of contributions and the license agreements that were in effect when those contributions were uploaded; it will currently restrict you from uploading a patch or documentation without a signed license agreement.

How to properly upload an attachment that requires a Submission License Agreement:

- When using the "Attach Files" form on a particular issue, be sure to select the "Yes, this is a code or documentation contribution" radio button.
- If you don't have a user license agreement, you'll see the message
  " You dont appear to have a current, signed submission license agreement on file. Please sign one before attempting to upload a code or documentation contribution."
- To sign a submission license agreement, use the "Sign a license Agreement" link at the top navigation bar in JIRA.
- The license agreement will be reviewed by Digium's legal department, then you'll be notified by E-mail of acceptance or rejection. If accepted you can now repeat the upload process and you'll be able to upload the code with a proper license associated to your account and showing by the attachment.

If you upload a patch and do not mark it as a code submission, and a bug marshal or developer later determines it to be a code submission, they may have to delete your patch and ask you to re-upload it, properly marking it as a code submission so that your license will be associated.

Multiple Authors of a Single Patch

- If you are the sole author of your patch, read and sign the agreement (Digium Submission License Agreement), then submit your patch.
- If there are multiple authors, all part of a single organization or corporate entity, then the representative of that entity, with that entity's permission may sign the agreement and submit the patch.
- If the patch has multiple authors, not part of a single organization or corporate entity, then all authors must sign the agreement and then either submit the patch or comment on the issue stating that they also provide and acknowledge the submission.

Get involved in Asterisk Development - influence the future of Asterisk

Reading the guidelines here is a great start! There are plenty of other wiki pages on Asterisk development which include coding guidelines. Feel free to start submitting relevant patches, documentation additions or fixes as soon as you can. There are many minor features and changes that need to be tested, reviewed and possibly expanded or tweaked before they can go into Asterisk.

Start associating with other Asterisk developers where they hang out http://www.asterisk.org/community/. You'll find them on primarily on IRC and the mailing lists. The forums are populated primarily by users.

Once you get to know the community and you really want to influence the Asterisk road-map, you'll want to check out AstriDevCon. AstriDevCon is an event held alongside AstriCon where the developers get together to review what was done in the past year and plan Asterisk's future.

Asterisk Community

- Asterisk Community Services
- Community Services Signup
- IRC
- Mailing Lists

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.

Community Services Signup

The 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 is the same account you can use to access wiki.asterisk.org.

To create an asterisk.org account, visit signup.asterisk.org and fill out the form.

The community services that currently support this centralized authentication are:

- issues.asterisk.org
- wiki.asterisk.org
- 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
```

Asterisk Developers Room

```
#asterisk-dev
```

Mailing Lists

There are several mailing lists where community members can go do discuss Asterisk. Some of the key lists are:

List	Description
asterisk-addons-commits	SVN commits to the Asterisk addons project
asterisk-announce	Asterisk releases and community service announcements
asterisk-biz	Commercial and Business-Oriented Asterisk Discussion
Asterisk-BSD	Asterisk on BSD discussion
asterisk-bugs	Bugs
asterisk-commits	SVN commits to the Asterisk project
asterisk-dev	Discussions about the development of Asterisk. #Development List Note
asterisk-doc	Discussions regarding The Asterisk Documentation Project
asterisk-embedded	Asterisk Embedded Development
asterisk-gui	Asterisk GUI project discussion
asterisk-gui-commits	SVN commits to the Asterisk-GUI project
asterisk-ha-clustering	Asterisk High Availability and Clustering List - Non-Commercial Discussion
Asterisk-i18n	Discussion of Asterisk internationalization
asterisk-security	Asterisk Security Discussion
asterisk-speech-rec	Use of speech recognition in Asterisk
asterisk-users	Discussions about the use and configuration of Asterisk.
asterisk-video	Development discussion of video media support in Asterisk
asterisk-wiki-changes	Changes to the Asterisk space on wiki.asterisk.org
asterisknow	AsteriskNOW Discussion
svn-commits	SVN commits to the Digium repositories
Test-results	Results from automated testing

Development List Note

This list is geared toward the development of Asterisk itself. For discussions about developing applications that *use* Asterisk, please see the asterisk-users l
ist.