The *Asterisk Handbook
An Operator's Guide to Configuring the
Asterisk Private Branch Exchange Server

Mack Allison and Mark Spencer

Purpose of this Document -

The purpose of this document is to detail the steps to configuring an Asterisk Private Branch Exchange Server. Basic telephony concepts are reviewed, and the design and layout of an Asterisk installation is discussed. This document should help the reader plan and design an installation for an enterprise installation, from the planning stages to configuring the system.

Installation of the Asterisk software is covered in another document released separately. Both documents are made available free of charge at www.linux-support.net.

Getting Support -

Community Support: Asterisk has a number of sources for support. As an open source project, Asterisk has a community support network primarily via mailing lists and the Internet relay chat system. Asterisk.org is a central place for open source and developer information. A FAQ is available, as well as updates on the most recent developments in Asterisk. You can sign up for the Asterisk mailing list to stay in contact with other users, ask (and answer) questions about how to make things work, and share ideas and configuration tips. To sign up for the Asterisk mailing list, send an e-mail message to asterisk-request@marko.net, with the word 'subscribe' in the **body** of the message. You will receive a response e-mail verifying your request and detailing how to make use of the mailing list.

Asterisk users (including the maintainer and project staff) can also be found on the #asterisk channel on irc.openprojects.net. On IRC, you can chat with other users in real time from anywhere in the world, and get advice and help from knowledgeable users of all experience levels. For new IRC users, more information about IRC is available from www.irc.net.

Commercial Support: Commercial support for Asterisk is available from Linux Support Services, Inc. Linux Support Services is the primary sponsor of the Asterisk project, and maintains copyright to the source code. LSS was founded by the primary author of the Asterisk software, and offers a line of compatible hardware, commercial support for Asterisk, and custom development and deployment services. Special license terms are also available for the Asterisk software. For more information about products and services offered by Linux Support Services, go to www.linux-support.net, or send an e-mail tosales@linux -support.net.

Documentation: Documentation on the Asterisk PBX is distributed in the Downloads section at www.linux-support.net.

Installation of Asterisk -

Asterisk is distributed in source code form for maximum flexibility. Debian GNU/Linux 3.x users can fetch packages from the Debian archives. Other distributions may also offer Asterisk in binary format for their distributions. The best way to install Asterisk is from source code, available from Linux Support Services as Linux standard tar.gz archives. For detailed information on installing Asterisk, fetch the document 'Installing Asterisk' from the downloads section at www.linux-support.net

Asterisk and Zaptel Telephony Interfaces -

Asterisk is commonly used with the Zaptel compatible interfaces sold by Linux Support Services. In order to use Asterisk with Zaptel compatible telephony devices, the zaptel drivers and Zapata libraries must be installed on the system prior to installing Asterisk. For information on installing these software packages, go to the downloads section at www.linux-support.net, and fetch the software and installation manuals.

Using the documentation and source code available at www.linux-support.net, installation of Asterisk is quite straightforward and should take only a few minutes of interactive time. Compile time varies based on the host system performance, but it's less than 10 minutes on a Pentium 450 or G4/400.

Currently, only the PowerPC and x86 architectures have been tested, and the PowerPC has only been tested using the Motorola G4 processors used in higher end Apple hardware.

Athlon optimized kernels are not currently supported. If you have an AMD Athlon family processor (K7, Athlon, Duron, Thunderbird, or XP) it is recommended that you use a kernel compiled for i686 with MMX optimizations on.

An Introduction to Telephony (Linux Style)

This section is intended primarily for person's who have very limited telephony background, though most Asterisk users will benefit from reading this. In this section, we'll go over some common telephony concepts, the basic functions of a PBX from a technical standpoint, and how all this fits together on the Linux platform with the Asterisk server.

Asterisk is what we call a 'Hybrid VoIP/TDM Private Branch Exchange Server.' What this means is that Asterisk supports both traditional analog and digital telephony equipment (commonly referred to as TDM) and the new arena of Voice over IP, in which voice traffic is carried over modern data networks just as other data, in IP packets. For this reason Voice over IP is often referred to as Packet Voice. Both technologies have their strengths and weaknesses, and for the foreseeable future, they will live side by side. Asterisk is designed to facilitate inter-operation between VoIP and TDM as seamlessly as possible.

TDM Equipment - The backbone of traditional telephony.

TDM equipment is the stuff telephony is made of today. Most of this equipment is analog, and uses a variety of signaling types and technologies to do the job. An extensive knowledge of TDM systems is not necessary to run an Asterisk server. There are a few terms to understand when using Asterisk in conjunction with channel banks and analog handsets, or using Asterisk to terminate and route a T1 line for voice traffic.

Phone equipment is normally signaled using Foreign Exchange signaling. There are two sides to this signal, the Office side and the Station side, commonly referred to as FXO and FXS. This is (slightly) analogous to client and server. A simple example of FXO and FXS exists in (almost) everyone's home. The wall jack you plug the phone into is an FXS device. It provides FXS signaling to the FXO device you connect to it, most commonly a
simple handset. The handset, an FXO device, provides FXO signaling to the FXS device on the other end.

This may be confusing at first. FXO devices **provide** FXO signaling, and are signaled with FXS signaling. FXS devices provide FXS signaling and are signaled with FXO signaling. What this means in practice is that FXO devices are signaled with FXS, and vice versa. This will be essential to keep in mind when setting up Zaptel TDM interface devices for use with Asterisk.

Many users will choose to use a T1 interface device and a channel bank to handle both incoming and outgoing calls. A T1 line carries 24 channels. A channel bank is a device that breaks a T1 line into its separate channels and provides a means to connect these separated channels to telephone handsets or incoming telephone lines. The channel bank may have any mixture of FXO and FXS channels available depending on its model and configuration. The FXO

Channels on the channel bank can be connected to standard incoming analog lines, in place of a telephone handset. The FXO channels will typically be connected to desk phones for users in the office.

Later on, we will look more closely at FXO and FXS signaling, during the configuration of channels. To sum up the basics, telephone equipment uses FXO and FXS signaling to intercommunicate. A T1 line bundles 24 of these FXO and FXS lines (or data channels.) T1 lines use a variety of signaling. We can use these T1s both to receive our telephone service from a provider and to intercommunicate with channel banks that can break our 24 channel T1 into 24 separate telephone lines.

Voice over IP - New technologies, new possibilities.

Voice over IP has been the subject of a huge amount of buzz in recent years, because it offers great promise to simplify business networks and it's ability to be routed anywhere the intra-net or Internet can go. Voice over IP offers some distinct advantages over TDM. It's very flexible. Any decent network connection can carry voice channels over UDP/IP. VoIP can be routed over wireless LANS, tunneled through IPSEC, and sent over the wide-open Internet. It is conceivable that the use of VoIP could eliminate the need for a separate telephone network entirely.

Realistically, VoIP has certain weaknesses that have yet to be overcome. The Internet Protocols it relies on were not designed to carry voice in real time. These protocols (and the equipment that speak them) were never conceived of as zero-latency connections (latency in the VoIP world describes the time between when a caller on one end speaks and when the caller on the other end hears them). Nobody cares if it takes 50 milliseconds for an html document's first packet to be received after it was sent by the server. However, TDM is zero-latency, and that standard has proven hard for any VoIP solution to live up to.

In places where VoIP offers new possibilities, it is widely being adopted. Many users are routing VoIP over Wide Area Networks to avoid constant long distance charges between geographically separate corporate offices. The easy adaptation into wireless networks offers great possibilities for creating a phone network in remote locations without existing networks.

For those familiar with IP technologies, VoIP is not too hard to understand. It follows the usual client server model; with the calls initiated through a series of connection packets and the actual audio portion of the call transmitted as UDP packets like any other network service. Audio data is compressed into a small monaural codec (such as GSM or the proprietary G.723.1 codec) and packetized for transmission.

Clients can be hosted software within a complete environment, such as Gnophone (a client released by Linux Support Services for use with Asterisk) or Microsoft's NetMeeting H.323 client. Such client's typically use the locally available audio hardware and networking devices to create a 'soft telephone' running on local operating systems. Stand-alone clients also exist, which are embedded computing devices that have just the right hardware to act as a telephone and communicate over a network with a VoIP server. An example of this is the Snom 100, an embedded VoIP telephone capable of communicating over a variety of VoIP protocols.

A number of VoIP standards exist. Asterisk has it's own fast and lightweight protocol called Inter-Asterisk Exchange, or IAX. IAX was developed primarily as a means of sending calls to remote Asterisk servers over LANs or WANs. Clients exist for IAX as well, to allow a host system or embedded device to be used as an 'IAX telephone.'
Gnophone is a sample IAX client released under the GPL by Linux Support Services. It allows any Linux system with a full duplex sound card and a network interface to make a telephone call through an Asterisk server. The Snom 100 is available from LSS as well, with IAX client software installed on it.

There are several competing standards in the VoIP arena. H.323 is used by a number of commercial vendors, and many in the industry are adopting SIP as well. These protocols offer different features for the intermingling of Voice and Data traffic.

Asterisk supports SIP and H.323 as well as its native IAX protocol. The architecture of Asterisk is specifically designed to make it easy to interconnect VoIP clients using various protocols and the widely deployed TDM equipment that currently offers the best audio quality.

Fundamentals of the Asterisk System

Asterisk provides a flexible system to design a dial plan suitable for each installation's needs.

The most fundamental concept in Asterisk is the context. Contexts provide a framework to present different dial plans to incoming callers. Using contexts within the extension logic system, Asterisk can change the available dial plan for individual callers based on the line (or channel) they're call is received on. Using various configuration options, the contexts can be used to present interactive voice prompting. Access control can be provided for reserved extensions and options. Different menu options can be presented during off-business hours as well.

The most obvious use of this is to keep incoming callers from dialing back out. Each incoming call has an initial context, defined on a per channel basis. We place our incoming lines in a special initial context. We place extensions to dial outside lines or access reserved features in another context. Those reserved extensions are accessible to incoming callers unless we explicitly make them available.

However, many extensions need to be shared amongst multiple groups of callers. We may not want our incoming callers to have access to outside lines, but we do want them to be able to reach our employees phones. Likewise, our employees need to reach each other. Since they are of necessity in a different context, they can't see the same set of extensions.

We can share parts of the dial plan selectively with the use of includes. We can create a context with a set of extensions, and 'include' that context in any other context that should have access to those extensions.

In this example we create two extension groups, in separate contexts. We place our two channels in initial contexts, and granted access to each caller group using includes. Building on this we can provide a special set of services to each group of channels, and create several initial contexts and selectively offer them related services.

Now we have a number of service contexts, and a number of starting contexts. Each starting context gets a slightly different set of services presented, depending on how their channel is configured. We might place a phone in the lobby for our customers' convenience, but we don't want to offer them free long distance. Likewise, depending on the nature of the business, we may not need (or want) to give long distance access to all employees. We may have separate special services, which we make available to upper management, such as special reserved conference rooms. We might also create an access control layer for upper management that keeps people from calling them directly, and instead only allows most callers access to their assistant's extensions.

Situations may exist where a caller in one context needs access to an extension in another context, but including the entire other context would be inappropriate. For quick hops from any place in the dial plan to any other, we can use the Goto application to jump to a specific extension, even if it's in another context.

Contexts offer us the flexibility to create multilayered interactive voice response systems with them. We can jump callers from context to context based on interactive choices, presenting different menu options and corresponding available extensions. Based on their choices, we can connect them with the correct employee or service.

Using time and date dependant options set in include statements, we can change the dial plan during off business hours or special times of day. Includes can be based on the time of day, day of the week, day of the month, or month of the year.

Extensions

In the Asterisk system, extensions are distinct from channels. Each extension takes a caller through a number of steps. In each step, Asterisk will execute an application. Every operation in a call is handled by calling an application. A large number of applications are available to perform the varied functions of a PBX.

Asterisk applications provide both basic and advanced features. Asterisk has applications to dial, hang up, and answer, and playback sound files. More advanced applications provide voicemail creating and retrieval services, conference bridges, and directory services.

Fig. A list of applications (exclude setup apps).

A typical desk extension would use the dial and voicemail applications. Several steps would be defined. Each of these steps is called a priority.

A second level of priorities is available. We can take a different path if the channel we are attempting to reach is busy, rather than simply left unanswered. We can create a slightly (or completely) different response for the caller in this case.

There are some special extensions available in addition to extensions based on user input. Asterisk provides a starting extension. When we answer an incoming call or move a caller into a new context without a specific extension, Asterisk will run the first priority in the starting extension. This is often used to present an interactive voice menu to the caller.

The timeout extension provides a way to handle a call if a caller doesn't respond with an extension within a configurable amount of time. We can use the timeout rule to recirculate the caller through the menu or play an error, rather than just dumping the call.

Asterisk provides extension matching logic to route calls based on part of the number dialed. Commonly, this is used to handle outgoing calls, matching all extensions dialed beginning with '9' to a single outgoing extension. The ability to match specific numbers and certain number lengths allows us to differentiate between local, long distance, and international calls, and calls to various area codes. Extension matching can also be used to route calls internally.

Channels

There are many technologies used to make and receive calls. Asterisk is able to interconnect incoming calls from all supported technologies. This flexibility is achieved by abstracting the technology used to make a call from the dial plan.

Asterisk supports a number of technologies, or channel types. Each available 'line' is a channel, whether that 'line' is provided by a hardware interface to TDM equipment or a Voice over IP connection. Inside the dial plan, Asterisk treats all interfaces in the same manner, and easily interconnects TDM, IAX, H.323, and SIP based channels with each other.

Asterisk has the ability to ring multiple channels at the same time. These channels can again be any mix of technologies compatible with the Asterisk server.

Planning an Asterisk Network.

When preparing to set up an Asterisk server, it is useful to visualize the topology of the telephone network, much like setting up an office data network. Take into account each service that will be offered on the PBX, and who should have access to each of these services. You may want to make a diagram of the network, showing which groups of callers should have access to which services. This will prove invaluable as you work through the configuration files. The diagram can show logical groups of callers and logical groups of services. Service groups can be connected with channel groups, showing the logical layout of contexts and includes.

First, examine the groups of callers that will exist on the network. At the simplest level, there are internal and external users. We may have some services that are only offered to some of the internal users of the network.

Fig - listing user groups for asterisk

Look at services to be offered by the Asterisk server, and who should have access to them. Many of the services will be available to several user groups. Most likely, employee desk extensions will be available to all users. Other services, such as voicemail, outgoing lines, and conference rooms, may only be made available to select callers.

Fig. listing services

Chart the services that should be made available to each user group. Group extensions that should logically go together, and place them in shared contexts. Connect groups of services with groups of callers to visualize how the dialplan should go together. There should ideally be no extensions stated twice. Any extension that will be available to more than context should be placed in a context to be included.

Fig. Connecting initial contexts and extension contexts with includes

In a number of places, we may have certain extensions we want to share with callers in another context, but including the whole surrounding extension context may not be desirable. Goto applications can connect callers with specific extensions within other contexts.

Fig. using gotos to share individual extensions

An introduction to the installation

Asterisk is configured in text based files stored in /etc/asterisk. The dial plan is constructed in extensions.conf. Channel and Application configuration files reside along side these files.

On startup, Asterisk automatically runs in the background. Error messages will be reported to the console Asterisk is started on. Optionally Asterisk may be run in the foreground, using the -c switch.

```
asterisk -c
```

The verbosity (amount of information printed to screen, can be set at startup. Three levels are available. The -v switch turns verbose mode on, and multiple 'v's raise the level.

```
asterisk -vvvc
```

If asterisk is running in the background or on another console, you can connect to the server and get a console.

```
asterisk -r
```

or

```
asterisk -vr
```

NOTE, when asterisk is running in the foreground, the 'quit' command will **shut down the server.** When connected remotely, 'quit' will disconnect from the console, leaving Asterisk running in the background.

The command line interface (CLI)

Asterisk features a command line interface with on line help system. When connected to the console, typing 'help' will list the available commands. Commands exist to shutdown or restart the server, both immediately and 'gracefully.' A graceful shutdown or restart will stop accepting new connections, and wait until there are no further calls. Shutdown or restart 'now' will immediately bring down the server, dumping all calls.

The command 'reload' forces asterisk to re-read its configuration files. No calls will be dropped during reload. Note that the zapata.conf file <u>is not</u> reloaded by the reload command. Asterisk must be restarted to reread this file and reallocate zaptel channels.

Steps to configuration

There are several steps to configuring the Asterisk server. The first is to make some channels available to Asterisk. Each channel will be assigned an initial context based on our layout. Channels are configured differently depending on the channel type, though they are treated the same in the dial plan.

Some applications have optional or required external configuration. The voicemail system, music on hold, and conference bridge applications must be configured prior to use within the dial plan.

Once channels are prepared and applications have been set up, the dial plan can be assembled. Using the notes made in the previous chapter, the dial plan will be developed in the extensions.conf file.

Setting up channels -

In this chapter, we will discuss the setting up of the channels. Asterisk commonly uses two major types of channels, though other channel driver modules are always in development. Currently, the best implemented channel drivers are the Zaptel driver, which connects Asterisk to interface devices available from Linux Support Services, Inc., and the Inter-Asterisk Exchange protocol, the lightweight Packet Voice protocol used to pass calls both between Asterisk servers and to communicate with IAX clients such as GnoPhone and the Asterisk-enabled Snom 100. H.323 drivers also exist in a fairly early state of development.

About zapata.conf -

In the first section, we will look at the zapata.conf file. The zapata.conf file configures Zapata interfaces for use with Asterisk. Within zapata.conf, zaptel channels are assigned signaling types, default contexts, and caller ID strings. You can also configure zaptel channels access to features such as three-way calling, voicemail, and call forwarding within this file.

You should have a sample configuration file in the /etc/asterisk directory after installation of Asterisk. The sample file itself can serve as a useful reference to

Some Notes on the style and flow of zapata.conf

Zapata.conf is read by the software from the top down. Configuration parameters are in the format "parameter=value." Parameter-value pairs are one per line. A semicolon precedes comments (;).

Each parameter given remains in effect until overridden by another parameter statement with a different value. For example:

```
context=local
signaling=fxo_ks
channel=1-12
context=trusted
```

```
channel=13-24
```

In the first line, we set the context for future definitions to local. In the second line, we set the signaling type. In the third line, we defined a block of channels. Since the channel definition was preceded by the context and signaling definitions, the channel will be signaled with fxo-kewlstart signaling. Callers from handsets on channel 1-12 will be placed in the local context when they pick up the phone.

In the fourth line, we changed the context to 'trusted.' Users who pick up a phone on channels 13-24 will be placed in the 'trusted' context. We did not have to restate the signaling type, as the definition on the second line remains in effect until overridden.

Some basic keywords -

There are some basic keywords you will want to get familiar with right off.

context: this keyword will define the caller group for following channels. The context will be referenced later by extensions.conf. Contexts are used to provide different sets of features and extensions to different users on the system. For example, incoming callers would normally be placed in a different *context* from local users, since incoming callers shouldn't be able to dial an outgoing extension (and certainly not access long distance lines) The context can be any alphanumeric string. You will have to have a context section in extensions.conf matching the section here for your users to be able to use *any* services.

channel: This keyword will indicate a list of channels. The channels listed will have the parameters in effect when the statement is made. If the last context definition was 'local' and the last signaling type was 'fxo_ks' then the channels listed will take those values. The values for a channel statement should be any zaptel channel which exists (for instance, if you have 2 fxo's and 4 fxs's, you would have 6 channels, numbered 1-6.)

signaling: Sets the signaling type for following channel definitions. These should follow the channels as defined in /etc/zaptel.conf. Correct choices are based on the hardware available. Asterisk will fail to start if a channel signaling definition is incorrect or unworkable.

The values available are:

em: E&M
em_w: E&M wink
featd: Feature Group D
fxs_ls: FXS Loop Start
fxo_ls: FXO Loop Start
fxs_gs: FXS Ground Start
fxo_gs: FXO Ground Start
fxs_ks: FXS Kewl Start
fxo_ks: FXO Kewl Start
pri_cpe: PRI signaling, CPE side
pri_net: PRI signaling, network side

The majority of users will use fxo_ks and fxs_ks signaling. These signaling types are most commonly used when connecting a Zaptel T1 card to a channel bank, and are always used when connecting and FXO or FXS station card.

callerid: Sets the Caller ID information for a channel or channels. The caller id string will be transmitted to other phones connected to your Asterisk network. If you are dialing out to the public phone network via PRI, the caller id line number will be sent to the destination phone. The public carrier will attach the matching directory listing to the number transmitted. Therefore, you can set the number on PRI's, but the name will automatically match the carrier directory. If you are dialing out via an FXO to a standard line, the carrier controls the caller id sent to the recipient entirely.

The format of the callerid string looks lke this:

```
callerid="Bob Salesman"<(345) 555-1234>
```

Like all other definitions, the callerid definition will remain in effect until overridden. If you need a bunch of channels to have the same callerid, you can define them like so:

```
callerid="Bob Salesman"<(345) 555-1234>
signalling=fxo_ks
channel=1-24
signalling=fxo_ks
channel=25-26
```

In this arrangement, the callerid definition would apply to all following channels.

group: defines a group of channels to be treated as one. The group can be referenced in extensions.conf, commonly as a dial extension. For example all salespeople could be grouped, so that an extension for sales can ring all phones. This feature is commonly used to bundle all outgoing channels into a single target for extensions.conf.

The format for a group definition is straightforward:

```
group=1 (replace one with any number)
```

Keywords to turn features on or off:

Most of the Zaptel features can be turned on or off on a per channel (or channel group) basis. These features take simple yes or no parameters:

usecallerid - should callerid be received
hidecallerid - should callerid be sent?
callwaiting - should call waiting be enabled
callwaitingcallerid - should caller id be sent on call waiting
threewaycalling - should three-way calling be enabled
transfer - should call transferring be enabled
cancallforward - should call forwarding be accepted
echocancel - should echo cancelling be allowed (yes, unless there's a good reson not to)
immediate - should the channel be answered immediately, or should Asterisk provide dial tone and wait for the caller to dial an extension

Some more keywords, less often needed:

There are a few extra keywords available, which are generally not needed by may be necessary in certain situations or with certain equipment.

switchtype - switchtype is used for PRI interfaces, when terminating a providers T1 line into your Asterisk server. Available switchtypes are -

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

rxwink - Normally, only used with the Adtran Atlas, when communicating using E & M wink. The Atlas seems to use >250ms winks, longer than average. This allows Asterisk to compensate.

rxgain (also txgain) - used to set the transmit and receive gain in dB. Used to compensate for certain interface devices which may seem 'quiet.' If your device works correctly, but the volume seems low, you may consider adjusting this value.

These parameters are seldom needed, but may be used to tune communication with certain devices

prewink:	Pre-wink time
preflash:	Pre-flash time
wink:	Wink time
flash:	Flash time
start:	Start time
rxflash:	Receiver flashtime
debounce:	Debounce timing

Examples -

The format of these examples will mirror the format of the file itself. Lines beginning with a semicolon (;) are comments, and will not be needed in the file itself. Lines not beginning with a semicolon are examples of functional lines in the configuration files.

A few examples -

Here's an example of what a config file might look like. In this configuration, we've got three outgoing standard phones lines, presumably provided by PCI FXO interface cards such as the X100p, and a T1 interface to a channel bank with 12 FXS channels to drive inside phone lines.

```
; T1 interfaces always come before single channel cards (reference 'Configuring zaptel.conf')
; First, we'll set some good defaults

[channels]
```

```
; set the default context to a restrictive group. Typically, default is outgoing local calls only
context=default

; turn on call features

usecallerid=yes
hidecallerid=no
callwaiting=yes
callwaitingcallerid=yes
threewaycalling=yes

; we'll reserve this feature for those who need it, as it will tie up 2 outside lines

cancallforward=no

; set the echo canceller - this feature should always be on, unless you have a hardware echo
canceller

echocanceller=yes

; set a default group

group=1

; begin channel definitions
;set the signalling for the FXS channels. Remember that FXS channels are signaled with FXO.

signalling=fxo_ks

; We'll start with the lowest internal access levels, and move upwards.

; A public phone, in the lobby.  No long distance, so we can leave it in default.
; Remember that caller ID will only be internal, since we are using standard Bell lines instead
of PRI

callerid=<Lobby - 1234>
channel=1

;The main extensions, for most employees
;We change the context to give them long distance

;context=employ

callerid=<John Kao - 6789>
channel=2
callerid=<Mary Jameson - 6790>
channel=3
callerid=<Joseph Johnson - 6791>
channel=4
```

```
;the engineering lab has two phones, and we ring want to ring them together
;by defining them as a group, we can simply target the group in the extensions.conf file

group=2
callerid=<Engineering - 6792>
channel=67

;for the sales department, we need international access, and since they roam a lot, we give
them
;call forwarding

cancallforward=yes

; If they can call international, they pretty much have full access to run up the bills :)

context=trusted

; We may want to signal them as a group, so we'll make them their own as well

group=3
callerid=<Stan Lebowsky - 6793>
channel=8
callerid=<Susan Stanberg - 6794>
channel=9
callerid=<Dave Lewis - 6795>
channel=10

; The CEO can stay in the same context, obviously he gets full access, so no need to turn
anything off.

; Get him out of the sales group, he doesn't want their calls

group=4
callerid=<John Poindexter - 6796>
channel=11

; And his executive assistant
callerid=<Kathy Morgan - 6797>
channel = 12

; Now we get to the three outgoing lines.  We better drop the context so they can't punch 9
and
; Dial through. We'll put them in local, so they can only ring local lines

context=local
callerid=<Widgets, Inc. - 3456>

; Make them a group, so outgoing calls take the first available line

group=5
```

```
;of course, we change the signaling

signalling=fxs_ks
channel=13-15


; END CONFIG FILE
```

That example shows the commonly used keywords. The other keywords available are predominantly for setting up special channel types such as PRI. Note the way that the file flows, with each option remaining in effect until overridden. This is especially important when working with contexts. If you aren't paying attention, you could give incoming callers the same access level as the CEO, and let yourself be someone's free long distance provider, or offer your conference bridge to the world.

Setting up Linux Telephony Interface Channels -

This section has not yet been written.

Setting up Inter-Asterisk Exchange Channels -

Inter-Asterisk Exchange channels are used for passing calls between multiple Asterisk servers over IP networks, and for communicating with Voice over IP client software and embedded devices. IAX offers a number of options for optimization, context control, and authentication and access control services.

IAX channels are configured in iax.conf. This file follows a similar format to other configuration files. Options are configured in 'keyword=value' pairs. A semicolon precedes comments (;). By convention, the first section of the file is normally used for general options for all IAX traffic. These special markers delineate sections of the file.

There are three types of IAX clients. The first is a user. Users can make calls through the Asterisk server, but are not able to receive calls from the server. This is useful in a situation where you might provide some phone services to a client, but would never call that phone, such as being a long distance provider for an IAX user. The second is a peer. A peer is a client you might pass calls to, but would never receive calls from. This might be useful to deploy a phone that only received calls, or passing calls to a special use Asterisk server.

Most commonly, the server or device would need to be both a user and a peer. In that case, you would define them as a friend, which is a shortcut for both user and peer. A friend can both send calls to the server and receive calls from the server. An IAX desk handset would probably fall into this category, as would a remote server that needed to access your local extensions and also offered it's extensions to your server.

There are several options for authentication and host definition. A client may be specified by specific IP address or resolvable host name, in which case no further authentication will be needed. A client may also be specified as dynamic, with or without a password. Dynamic hosts may also have an IP range assigned that they are allowed to connect from, or an IP range that is specifically excluded.

Authentication by password is also available. A password for the client may be specified, and three token passing methods are available, plaintext, MD5, and RSA. Plaintext is the least secure, and probably shouldn't be used in hostile environments (over the internet, college LANS) as the passwords are sent clear text and easily recovered with sniffer programs. MD5 uses a one-way hash that makes it harder to get the actual password. This offers minimal security against password sniffing, but requires that the passwords be stored plaintext on the servers themselves. RSA is the most secure method. In this method, public key/private key encryption like Secure Shell is used to authenticate.

The first section of the iax.conf file is the 'general' section. In this section, general options can be configured. The defaults in this section are good for most systems, though some applications may call for adjusting the jitter buffer and codec selection options for better performance and bandwidth utilization.

Most clients will be 'friends,' the shortcut for both user and peer. The simplest way to set up a friend is as a static definition. In such case, no secret or authentication is needed. An IP address or specific hostname is defined, and all calls routed to the peer will be sent to that IP. All incoming (user side calls) presented as being from that peer must originate from that IP address or resolvable hostname. Such a friend would look like this:

```
[friendname]
type=friend
context=local
context=default
host=192.168.10.18
```

Note that friends and users can both have multiple contexts. Peers do not need contexts. When a friend/user passes a call to the server (which is their peer) they can specify which of the available contexts they are connecting into.

A more complex friend definition allows for a dynamic host, which can be open, or limited to a range of IP addresses. This can be useful for IP handsets that get dynamic IP addresses from their DHCP server, or for a software client on a laptop that could have any IP address when it attempts a connection. Though authentication is not mandatory even in this case, it is generally recommended. An example of a peer that could have any IP within the local LAN:

```
[ipphone]
type=friend
host=dynamic
allow=192.168.0.1/255.255.255.0
deny=0.0.0.0/0.0.0.0
```

We can add basic (plaintext) authentication to the mix with this addition:

```
secret=a_not_very_secret_secret
```

That authentication might suffice in a local LAN, but really isn't much better than no authentication at all. A moderately stronger way to do things would be to use MD5 sums instead of the passwords themselves to do the checking. Add the line

```
auth=MD5
```

to the friend definition to make sure the password itself isn't open to the average sniffer.

If we need to accept hosts over the Internet, with whatever IP address they happen to have at the moment, we probably want something pretty strong. In this case RSA public key/private key authentication is the best method to ensure the integrity of your phone network.

Setting up RSA encryption does take a few extra steps. You must have openssl installed on the system to proceed.

The step first is to generate a public/private key set for use in the authentication process. You will use the included utility astgenkey to do this. To run the keygen, run astgenkey at a shell prompt. The program will prompt you for a key name. This will be the name of the key as referenced in iax.conf. Following that, the program will prompt you to enter your PEM key three times. This will be the key phrase used to 'init keys' on asterisk startup.

Fill in these values. The program will exit, generating two files, keyname.key and keyname.pub. Keyname.key is the private key, and will be stored locally (on the client.) Keyname.pub is the public key, and should be stored on the Asterisk server. Copy the private key to /var/lib/asterisk/keys on the local (client) machine, and send the public key to the server machine. Place the public key in the same location, /var/lib/asterisk/keys.

To set up asterisk to make use of these keys, change the 'auth' line in our sample config above to read:

```
auth=rsa
```

and add the following line:

```
inkeys=keyname
```

On the client side, add the line:

```
outkeys=keyname
```

in a convenient place in iax.conf (I normally put outkeys before the first client definition.)

The final options available on a per client basis relate to the Call Detail Records accountability and billing system. You may set the amaflags to one of default, omit, billing or documentation. You may also specify the account code to attach all calls from a particular client to.

Those options should serve to demonstrate the available parameters for IAX client definitions. More complete line-by-line examples are below.

As mentioned before, there are some general options available to fine tune the operations of the IAX system. These options appear in the 'general' section of the file. By convention, general options are set before any client definitions.

The options available to tune the server behavior are port and bindaddr. Port specifies the IP port number to listen on, normally 5036. The bindaddr keyword specifies a particular IP address to bind to among available interfaces and aliases on the host system.

Defaults can be set for the amaflags and accountcode options. These options in the general section will apply to all clients that do not have them specifically overridden within thier client definition .

There are some options available to tune the performance of the IAX stack, particularly useful in systems using VoIP over constricted connections or very long Internet routes. Options can be set to control which codecs are ever used. Asterisk will attempt to select the best codec and rate for the available connection. The simplest way to set this is to use the bandwidth keyword, which takes one of the avaialble options high, medium, or low. Note that this option will override specific allow/disaalow options.

For finer-grained control of codecs, use the allow and disallow keywords, and comment out the bandwidth keyword. The available codecs are gsm (which is generally preferred, as it offers excellent voice quality in a fairly compact stream,) lpc10 (which is smaller still, but sounds mechanical or robotic to most ears,) and g723.1. Note that due to
patent restrictions, Asterisk cannot encode or decode g723.1 internally, though it can connect clients together that use this codec. In addition, you may allow all, which is the same as bandwidth=high.

For even finer control of IAX behavior, you can adjust some settings to control the jitter buffer. The jitter buffer tries to compensate for the variation in times between packets, so that the sound doesn't break when a packet takes a few milliseconds longer than the previous. You can turn off the jitter buffer entirely, though this is not really recommended. Otherwise, you can set the maximum size of the jitter buffer, the maximum drop rate (the rate at which packets are dropped to reduce latency), and the maximum excess buffer.

Setting the maximum jitter buffer can improve performance and reduce memory consumption. By default, we use a value of 500, which works for most connections. Setting the drop rate can improve latency at the expense of a bit of quality. Raising the drop rate will cause Asterisk to drop packets (potentially producing 'breaks' in the audio) in order to reduce the latency of the conversation and reduce the jitter buffer (and hence, the memory usage.) Setting the maximum excess jitter buffer will cause Asterisk to attempt to maintain an empty buffer of the size defined, by more aggressively dropping packets to force the buffer to shrink. In the case of the default values (jitterbuffer=500, maxexcessbuffer=100) Asterisk will try to slowly shrink the jitter buffer when usage exceeds 400.

The last options in the general section relate to registration with another server. Registering is not necessary if the client is specified by specific IP. If the client is dynamic, it will have to register to the host so the server can find it when a call arrives. Registration can be done with or without authentication, of course. The format of a register entry is like this:

with no password:

register => client@asterisk.widgets.net

with a password (plaintext or MD5):

register => client:password@asterisk.widget.net using RSA:

```
register => client:[keyname]@asterisk.widgets.net
```

Registering happens on the peer side, so that the server can pass calls to the client. In the case of a friend (or user) the password or key name will also need to be used in extensions.conf to pass calls to the server. More is available on that in the section on extensions.conf.

Now, let's look at a complete simple example.

```
;Up here at the top, we'll set the general options. This tag denotes the beginning of the
;general section
[general]
;we'll set the port and IP for the server
port=5036
;In this case, I want to listen on all interfaces, so comment this out
;bindaddr=192.168.0.1

;we'll set some default accountability options

amaflags=default
accountcode=corpinternal01

;now we'll get to the codec options
;this option is commented out, we'll use allow/disallow instead

;bandwidth=low

;allow only certain codecs

disallow=g723.1
disallow=lpc10
allow=gsm

;and set some jitter buffer options
;this is unnecessary, it's default already
jitterbuffer=yes
;the included files use three, but perhaps my connection has too many hops
dropcount=5
;set a sane maximum, to avoid runaway memory
maxjitterbuffer=500
;and let it know when to 'softly' pull the jitter buffer down, so it doesn't run out and get ;ugly
maxexcessbuffer=100

;I have a couple of sister servers that I am a peer to.  One is internal and the other is a ;remote
internet host

;I use plaintext inside, cause I trust my network (famous last words)
```

```
register => server1:floodle@server2.widget.net

;over the net is a different story, I'll use RSA for that
register => server1:[mykey]@remoteserver.widget.net

;Don't forget to load my RSA key(s)
outkeys=mykey:myotherkey
;Now I can start with the clients.

;an IP phone that only rings, but can't be called through. Since it only recieves calls from ;me,
it's a peer
[trouble]
type=peer
host=192.168.0.1
;in this case, I am a user...here's what I'll send him to authenticate as his user
username=server1
secret=foo_bar

;I have another phone that's a user.  I never ring it, but It can be used to dial others
[dialout]
type=user
secret=bar_foo
;just to be on the safe side
auth=md5
;it has to register, because I don't know where it is
host=dynamic
;except that it better be in my LAN
allow=192.168.0.1/255.255.255.0
deny=0.0.0.0/0.0.0.0
;the boss want's to know how much this phone gets used
accountcode=dialout1
amaflags=documentation
;and I'll set the caller ID on it, as well
callerid="Basement Phone" <(234) 567 8901>

;the majority of my iax clients will be friends, as we'll pass calls back and forth
;you'll recognizer these two, because I register to them as a peer so they can give me calls
;I won't set caller IDs, even though they are users, because they'll pass me thier own
[server2]
type=friend
auth=plaintext
secret=bah_humbug
allow=192.168.0.1/255.255.255.0
deny=0.0.0.0/0.0.0.0

[remoteserver]
type=friend
auth=md5
inkeys=remotekey
host=dynamic
;I can set a default, in case it hasn't registered yet, I'll try this.
```

```
defualtip=233.122.144.23
```

Setting up SIP channels

The sip.conf file is /etc/asterisk is used to configure

Setting up H.323 channels

This section has not yet been written

Configuring Applications

The more complex application services need to be configured prior to use. In this chapter, we will prepare voicemail, set up conference bridges, and configure the music on hold system for our configuration.

Voicemail -

The voicemail system is configured in the file 'voicemail.conf.' In this file, Mailboxes are associated with an owner's name, password, and an e-mail address to notify when a new message is received. A mailbox must be created for the user as well.

The fastest way to create a mailbox is to use the 'addmailbox' script provided by the Asterisk installation. Simply run this application as root (or someone who has write permissions to /var/spool/asterisk/vm.) It will prompt for a mailbox number. Enter the number, and the script will create the directory /var/spool/asterisk/vm/[boxnumber] and copy the default busy, unavailable, and greeting messages (found in /var/lib/asterisk/sounds) to this directory.

Configuration is necessary in voicemail.conf, as well. Each mailbox should have an entry in the voicemail.conf file in this format.

```
[mailboxnumber] => [passkey],[User Name],[email@address.com]

6161 => 1234,James Murdoch,jmurdoch@widgets.com
```

The passkey will be used when the box owner needs to access his voice messages over the phone. The username is referenced by the Directory application (more on that in the section on directory.) The last entry is the email address the message waiting notification should be sent to.

Conference Bridges -

Conference Bridges are configured in the file meetme.conf. In meetme.conf we list the conference rooms available. Each conference room has a number, which will be referenced when the application is called from within extensions.conf. The application can also be called without a room number, in which case the caller will be prompted for a room number. The format of meetme.conf is:

```
conf => [roomnumber]
```

Room number can be any numeric string. Once a room has been defined in meetme.conf, it is available to the meetme application.

Music On Hold -

The Music On Hold application provides background audio for parked or held calls, and during call transfers. Audio files are stored in the common mp3 format, typically in /var/lib/asterisk/mohmp3. Multiple 'classes' can be configured and used, each with different audio file collections.

Configuring Music on Hold

Classes are configured in /etc/asterisk/musiconhold.conf. Each class statement comes in this format:

```
[classname] => mp3:[/full/path/to/mp3/dir], [extra args to mpg123]
```

Classname will be the name referenced in extensions.conf (see SetMusicOnHold) Currently, two main playback modes exist, mp3 and quietmp3. Quietmp3 is identical to mp3, except that it reduces the volume by 75 percent. The path should lead to an accessible directory containing one or more mp3 files. By default, files will be played in alphanumeric order. Arguments can be passed to mpg123, after the path definition. A common use would be to put mpg123 in shuffle or random mode with the '-z' options

```
random => mp3:/var/lib/asterisk/mohmp3, -z
```

We are already using the -q, -s, --mono, and -r options, and setting the sampling rate to 8000. Be careful not to override these options blindly. The Asterisk Music On Hold architecture depends on the very common free mpeg audio decoder program 'mpg123.' Though most distributions ship with this program by default, it may need to be installed on the local system. Some newer systems have replaced mpg123 with a work-alike program called 'mpg321.' This program does not work for Asterisk Music On Hold, because it does not properly implement some of the resampling features used to generate phone quality mono audio. If you experience 'dragging' audio files that play back at lower speeds than they should, this is most likely the problem.

Building the dial plan - extensions.conf

Extensions.conf is divided into sections by context. Each section begins with the context name, followed by the list of extensions within that context, and any includes that will added. Comments are proceeded by a semicolon (;), not the more common hash (#), because that's a digit.

A typical PBX extension takes a caller through a number of steps. A common desk extension, for instance, will ring a phone for a period of time, then roll over to voicemail. After voicemail, if the caller does not disconnect, we might want to circle the caller back to an earlier menu, such as the entry menu that offers a directory.

Within Asterisk, each extension is a numbered list of applications to run. NOTE: There are a sizable number of applications available. This chapter does not attempt to document the apps themselves. See 'Appendix B: Applications' for detailed information about each application. Every action the PBX takes is an application, from simple apps like Dial (dial a channel, and connect the two channels if anyone answers) to more complex functions, such as receiving a voicemail message from the caller. We call each step of the extension the priority. Here's a simple example of a mostly complete desk extension.

```
;Run dial on a specified channel (ring a phone)
;In this case, we are ringing Zaptel channel 12,
;for 20 seconds before moving to priority 2

exten => 6000,1,Dial,Zap/12|20

;hmm, no one answered.  I guess we'll just take a message.

exten => 6000,2,Voicemail,u6000
```

Using the second level of priorities for busy channels, we can create a different response when all channels referenced are busy. We use the priority series 1XX.

```
exten => 102,Voicemail,b6000
```

In this example, the only thing changed was the rollover message when we go to voicemail, instead of playing a message saying the person is unavailable, we played a message saying the person is busy.

Special Extensions

Asterisk has three special extensions it looks for under defined circumstances. The 's' extension is used for 'start.' When a caller arrives in a context without going to a particular extension. Asterisk will look for 's' and run that extension. For example, when Asterisk answers a ringing outside line, it will start with extension s in the initial context for that channel. For example:

```
;play a 'Thanks for calling' audio file.
exten => s,1,Playback,thanksfor
```

As soon as Asterisk answers a line in this context, it will play back the specified audio file.

Extension 't' is used when a caller times out, such as while waiting for user input. Without the 't' extension, Asterisk will dump a caller once the timeout is reached. Using this extension, we can properly handle this condition. This extension will recirculate the caller to the start extension for his context.

```
exten => t,1,Goto,s|1
```

The third special extension is 'i', used when a caller dials an invalid extension, one which does not exist in the callers context.

```
exten => i,1,Playback,invalid
exten => i,2,Goto,s|1
```

The latest special extension is ‘o’ to go to an operator. This extension is used within the Voicemail application. If a user presses zero during the Voicemail dialog, Asterisk will look for an extension ‘o’ which would presumably be a Goto or similar to take the caller to the receptionist’s phone.

```
exten => o,1,Goto,6000
```

where 6000 would be the receptionist’s extension (or whereever you would want to send the caller.

Includes and Gotos

Includes and Gotos provide flexibility within the dial plan. The include statement makes extensions in one context available to another, optionally with a time argument. The basic form is :

```
include => anothercontext
```

That statement will unconditionally offer all the extensions in 'anothercontext' to callers in the including context. Time parameters added to includes can allow us to make extensions available during parts of the day. We can change the menu our callers receive by including contexts with different 's' extensions during out of business hours, or only allow use of some services during certain parts of the day.

We can supply these time dependant arguments to include contexts during certain times or days.

```
include => anothercontext|time-range|day-range|days of month|months

include => anothercontext|09:00-18:00|mon-wed,fri|*|1-11
```

Spans may be indicated with a dash, such as mon-fri, and separated with a comma, such as mon-wed,fri. Any time can be specified with a * wildcard. The above include statement applies between the hours of 9am and 6pm (18:00), on Monday though Wednesday and Friday, January through November.

Gotos are used to go from any extension in the dial plan to any other extension. Gotos can jump to extensions in the same or other contexts, and to specific priorities within the target extension. The format is:

```
Goto,context|extension|priority
```

Not all values are necessary. Goto will assume one argument is a priority, two arguments are an extension and priority, and three arguments are context, extension and priority.

'Goto,1' will jump to priority one in the same extension, while

'Goto,s|1' will jump to the 's' extension, priority 1, in the same context.

'Goto,local|s|1' will jump to the local context, extrension s, priority 1.

Goto is useful when you need to move a caller to a particular extension, or when a caller in one context needs access to an extension in another context, but not all of the extensions in that context.

Extension matching-

In some cases, we may want to route calls based on part of the extension dialed. We could need to pass all extensions beginning with '6' to another server, or jump to another context to route certain extensions without including the entire context. Commonly, we will need to route all calls beginning with '9' in the same way, passing them to an outside line.

For this purpose, Asterisk features an extension wildcard system to match parts of dialed extensions. For instance, we might simply want to route anyone pressing 9 to an outside line, and let them send the rest of their digits straight to our service provider. If we proceed an extension with an underscore (_) Asterisk will interpret that extension as a wildcard. Digits will be matched exactly, and two matching symbols will be available, 'N' and 'X.'

'N' matches any digit greater than 1. 'X' matches any digit. In our sample, we can allow callers to access the outside line and pass any phone number to the service provider with this extension match:

```
exten => _9NXXXXXX,1,StripMSD,1
exten => _NXXXXXX,1,Dial,Zap/g2/BYEXTENSION
```

In this case, we matched any extension beginning with 9, followed by a number greater than 1, and a total of 8 digits including the 9, directly to g2. The g2 would indicate group #2 in zapata.conf. Assumptively, that would be the group containing our outside (FXO) interfaces.

This match is quite specific, and will only allow local calls (in the US, at least.) Since the match excludes 1 as the first digit after the nine, this extension will only match local calls, and will not allow callers to access long distance.

BYEXTENSION tells Asterisk to connect the specified channel, and send the digit string whole to the receiving channel. In this case, this means that when a match is found, open channel g2 (or actually, the first available channel in group 2) and passes the phone number to the other end of the channel. The StripMSD application strips the first digit from the number before sending, so that the nine isn't sent to the provide, which of course would mangle the call and get an error message back.

We can separately match long distance calls (in the US) by creating a match to _91NXXNXXXXXX. That would match any domestic long distance call, but would not match an international call.

Matching international calls can get very tricky, as different countries use different numbering schemes, and a few countries, such as Germany, do not have a universal length for phone numbers. For most companies, simply allowing a group of channels access to all international calls is sufficient. Simply giving trusted users open access to the outside line, with a match for _9X, could do that. However, if you wish to specifically restrict international calls to certain countries, you may create extension matches containing country codes, and have attempts to dial unapproved countries result in no match.

Breaking down an extension.

Each extension consists of a series of priorities. In each priority, an application is called. Each application may take optional arguments, such as a voice mail box for the voicemail app, and channels passed to the dial app. The format of a single priority is:

```
exten => [exten],[priority],[application],[arguments]
```

Looking at a sample desk extension closely, we can see how these lines come together. For the first priority, we run the 'Dial' application, passing as an argument the channel to dial and a timeout in seconds. More about the Dial app can be found in Appendix B, Applications.

```
exten => 6070,1,Dial,Zap/10|20
```

Assuming no one answers, we'll fall through to the next priority. In the next priority, we run the Voicemail app, passing a voicemail box as an argument.

```
exten => 6070,2,Voicemail,u6070
```

The voicemail application will automatically disconnect the caller after the message is received. u6070 tells voicemail to play the unavailable message, as opposed to the busy (b6070) message. More about voicemail can be found in Appendix B, Applications.

If the phone line is busy, Asterisk will look for the next priority, +100. If Priority 1, Dial, finds the target channel busy, it will look for priority 102, instead of priority 2. The simplest use of this is to change the message played by the voicemail. We might also offer them the opportunity to wait, and try again after some amount of time. We can do this:

```
exten => 6070,102,Voicemail,b6070
```

Using Asterisk to create layered Voice Response Menus

Using context switching and Gotos, Asterisk can be configured to create layered voice menus. A typical use for this would be to prompt people calling the company number, and gather information from them to route their call correctly.

We can create a voice response menu on extension 's' for our incoming callers. We use the sister applications Playback and Background to play voice prompts. Playback and Background differ only in that Background excepts user input while playing, and Playback ignores digit presses from the user. We can 'Playback' a short welcome, then background the menu itself.

Perhaps a couple of departments have their own special menus, to present options specific to those departments. We can use Goto's to jump from this menu to the 's' extension in another context where the next menu resides.

```
[start]
include => desks
exten => s,1,Playback,welcome
exten => s,2,Background,menu1
exten => s,3,Wait,20
exten => s,4,Goto,s|2
exten => 1,1,Goto.sales|s|1
exten => 2,1,Goto,shipping|s|1
exten => 3,1,Goto,support|s|1
exten => 4,1,Directory,desks
```

Here, we make them listen to a welcome message (Thanks for calling . . .). We follow that by backgrounding a menu, so that the caller pick their choice as soon as they hear it. After playing the menu, we'll wait 20 seconds, and play it again. Extension 1 jumps to a sales department context containing it's own menu system. Extensions 2 and 3 jump to similar contexts for shipping and support. Extension 4 runs the Directory application, which attempts to match a caller with the correct extension based on the name of the person they are trying to reach (Appendix B). If we make 'start' the initial context for incoming calls, this menu will automatically be played when a call is received. We could also include start another context and make the extension 's' available that way.

In order to make direct dialing of desk extensions possible, we also included the 'desks' context, which in this case should be a context with the direct dial extensions for each of our employees.

Custom audio files can be created with the Record application. We can create an extension to record an audio file, and use those files with the playback and background apps to customize the menus and prompts for a PBX.

```
exten => 3000,1,Record,foo|gsm
```

That extension will record a file 'foo.gsm' in the default sounds directory. See Appendix B.

Setting up some initial defaults

Often, we may want to set some initial defaults for various behaviours. A number of applications are available to set up such things as timeouts and music on hold classes. The 's' extension is often used for this purpose.

```
;set some defaults
exten => s,1,SetResponseTimout,20
exten => s,2,SetDigitTimeout,10
exten => s,3,SetMusicOnHold,default

;the menu
exten => s,4,Playback,welcome
exten => s,5,Background,menu1
exten => s,6,Wait,20
exten => s,7,Goto,s|5
```

```
;the options
exten => 1,1,Goto.sales|s|1
exten => 2,1,Goto,shipping|s|1
exten => 3,1,Goto,support|s|1
exten => 4,1,Directory,desks

;special extensions
exten => t,1,Goto,s|5
exten => i,1,Playback,invalid
exten => i,2,Goto,s|5
```

Bringing it all together - a sample file.

With our dialplan designed, we can create an extensions.conf file. A simple extension.conf file might be a single context, but most will need more contexts. This should provide a decent example of the how a small business system would be laid out.

```
;the starting context for incoming calls

[start]
;set some defaults
exten => s,1,SetResponseTimout,20
exten => s,2,SetDigitTimeout,10
exten => s,3,SetMusicOnHold,default

;the menu
exten => s,4,Playback,welcome
exten => s,5,Background,menu1
exten => s,6,Wait,20
exten => s,7,Goto,s|5

;the options
exten => 1,1,Goto.sales|s|1
exten => 2,1,Goto,shipping|s|1
exten => 3,1,Goto,support|s|1
exten => 4,1,Directory,desks

;special extensions
exten => t,1,Goto,s|5
exten => i,1,Playback,invalid
exten => i,2,Goto,s|5

;our employees extensions
[desks]
exten => 6070,1,Dial,Zap/10|20
exten => 6070,2,Voicemail,u6070
exten => 6070,102,Voicemail,b6070

exten => 6071,1,Dial,Zap/11|20
exten => 6071,2,Voicemail,u6071
exten => 6071,102,Voicemail,b6071
```

```
;more similar entries for each employee

[service]
;put some internal services in this context
exten => 8500,1,VoicemailMain
exten => 8600,1,Meetme,8600
exten => 8601,1,Meetme,8601
exten => 8602,1,Meetme,8602
;and the extension to dial out can be here too.
exten => 9,1,Dial,Zap/g2/BYEXTENSION

;create two contexts to be used as initial contexts
;we use includes to control who gets which services

[local]
;this will be our contexts for incoming calls
include => start
include => desks

[default]
;this is our context for internal users
;dial tone continues after they dial 9
ignorepat => 9
include => desks
include => service
;give access to call parking. See appendix B
include => parkedcalls
```

That configuration would provide a simple framework for a PBX. Advanced features could be added to it to enhance the functionality. One that may make sense for many installations is Direct System Inward Access (DISA), which allows us to authenticate an outside caller and grant them access to a more privileged context. We might add a DISA to the 'start' context, and allow our employees to authenticate and get access to the company conference rooms and voicemail services.

See Appendix A, Sample configurations, for examples of other configurations and options available.

Appendix A - Sample Configurations

This appendix contains some sample configurations to use as a reference for configuring your Asterisk server.

Home Voicemail - a simple setup.

Asterisk can be combined with a single FXO channel to provide a nice voicemail system for a home user. The channel should NOT be set to autoanswer.

```
;this should be the initial context for the fxo line, as set in zapata.conf
[incoming]
;wait 20 seconds for someone to pickup
exten => s,1,Wait,20
exten => s,2,Answer
;Play a thank you, and offer three mail boxes
;for different members of the household
exten => s,3,Playback,thanks
exten => s,4,Background,menu
;the mail boxes
;these should be configured in voicemail.conf and generated with addmailbox
exten => 1,1,Voicemail,1
exten => 2,1,Voicemail,2
exten => 3,1,Voicemail,3
;should prolly be possible to check the voicemail
exten => 8500,1,VoicemailMain
```

A more complex home setup, with one fxs and one fxo.

This setup uses some nicer options, because with the fxs channel, Asterisk can run and provide services to the call. Now we can put callers on hold and play them special hold music. If we had more incoming lines available, we could use a conference bridge as well.

```
;the fxo initial context
[incoming]
exten => s,1,Playback,thanks

;set music on hold
exten => s,2,Setmusiconhold,default

;Dial the FXS's channel for 20 seconds
exten => s,2,Dial,Zap/2|20

;nobody's home
exten => s,3,Playback,nobody
exten => s,4,Background,menu

;the mail boxes
exten => 1,1,Voicemail,1
exten => 2,1,Voicemail,2
exten => 3,1,Voicemail,3

exten => 8500,1,VoicemailMain

[default]
include => incoming
include => parkedcalls
```

Appendix B - Application Modules for the Asterisk System

The Asterisk system comes with a wide selection of applications to perform the varied functions of a PBX. Some applications require external configuration. Application config files will reside in the /etc/asterisk. Here we'll look over the applications available to the default Asterisk installation, their arguments, and any necessary external configuration.

Dial

Dial is probably the most commonly used application in the Asterisk system. The purpose of the Dial application is to Dial a channel and wait for someone (or something) to answer the other line. If anyone picks up, Dial will bridge the incoming and receiving channels together. Dial will gladly connect any channel type with any other channel type. The Dial app requires no external configuration (except, of course, the presence of configured channels.) Multiple channels may be specified on a single priority, in which case they will be rung simultaneously. Optionally, a timeout (in seconds) may be specified, which will be the amount of time the channel rings before we give up and move to the next priority.

The Dial application takes a specific channel as an argument. This can be any VoIP channels available, a channel provided by a Zapata telephony interface, an OSS full-duplex sound card, or a channel provided by the Linux Telephony Interface and configured in phone.conf (typically Quicknet interface devices.) The format of the arguments to Dial change depending on the type of channel being signaled.

Formats for Dial arguments:

For Zaptel -

```
Dial,Zap/[channel]|[timeout]
```

Note that between the channel and the optional timeout is a pipe symbol.

For IAX -

Due to the many options available to IAX, Dialing IAX has several available arguments, some of which may be optional.

```
Dial,IAX/[username]:[password, if needed]@[my.asterisk.server.com]/extension@[context]
```

The context is optional, only needed if the IAX client has access to multiple contexts (configured in iax.conf on the server side.) If multiple contexts are available but none is specified, then the context will default to the first one listed.

The password argument is in a slightly different format when using RSA encryption to protect our authentication tokens. When using plaintext or MD5 authentication, we would replace [password] with the secret:

```
exten 1,1,Dial,IAX/me:notverysecret@asterisk.server.com/6161@local
```

where notverysecret should be the secret. In the case of RSA, we would replace the plaintext secret with the RSA keyname in brackets, like:

```
exten => 1,1,Dial,IAX/me:[supersneaky]@asterisk.server.com/6161@local|10
```

Hopefully, that's the last time I have to actually **use** brackets in an argument!

In some cases, we may omit the password. A secret may not be specified on the server, i.e. when using host-based authentication. The context may be omitted if we only have one available, or want the first of several contexts made available to us. The timeout is always optional.

If the IAX client to be signaled registers with the Asterisk server, we can simply specify the peer name and the extension on the peer we want to connect to.

```
exten => 1,1,Dial,IAX/registeredpeer/1|20
```

Linux Telephony Interface

Linux telephony interface (LTI) devices are accessed through their device file descriptors. The ringing of LTI channels is straightforward, like Zaptel.

```
exten => 1,1,Dial,Phone/phone0|10
```

Where phone0 is analogous to /dev/phone0, the device file provided by the Linux Kernel Driver.

Ringing H.323

This section not yet written

Ringing SIP

This section not yet written

Ringing multiple channels

In some cases, it may be useful to ring multiple channels when a call comes in. We could set an extension up to ring a person's desk phone and their cell phone at the same time. We could also make Dial attempt to ring an IAX client at the same time it rings a zaptel device (or ring all three at once.)

To do this, we separate the channel arguments with a '&.' Any number of channels can be specified on the priority in this fashion.

```
exten => 1,1,Dial,Zap/10&IAX/snomphone/1|25
```

Note that we can only set the timeout for the Dial app as a whole, not the individual channels we are ringing. If we do want one channel to ring longer than the other, we can do it like this:

```
exten => 1,1,Dial,Zap/10&IAX/snomphone/1|15
exten => 1,2,Dial,IAX/snomphone/1|15
```

In this case, we rang a zaptel channel and an IAX channel simultaneously for 15 seconds, before giving up on the zaptel channel, and trying the IAX channel for another 15 seconds (if the IAX channel could be contacted.)

Dial gives us great flexibility in attempting to connect a caller to his destination. Using the Dial application, we can create extensions that will find our employees wherever they might be, via PSTN connections (and cell phones) and over IP connections, or in the office.

Goto -

Goto is our dialplan jumping application. Though we looked at it briefly earlier, we'll now go over the systax of Goto arguments. Goto is capable of jumping from one priority to another within an extension, jumping to another extension in the context, or jumping to an extension in an entirely different context.

The format for Goto is like this:

```
exten => 5,4,Goto,context|extension|priority.
```

The context and extension arguments are optional. If no pipe (|) is present, Goto will interpret the argument as a priority within the current extension. If we add an extension, in the format extension|priority, Goto will jump to that extension within the current context, to the particulat priority specified. If we expand further, we get context|extension|priority, which will jump the caller to any specified point in the dialplan.

Goto takes no other configuration, other than the presence of the target priority.

Voicemail -

The voicemail application records messages from callers. The voicemail application requires external configuration. A destination mailbox must be created, and configured in voicemail.conf.

The fastest way to create a mailbox is to use the 'addmailbox' script provided by the Asterisk installation. Simply run this application as root (or someone who has write permissions to /var/spool/asterisk/vm.) It will prompt for a mailbox number. Enter the number, and the script will create the directory /var/spool/asterisk/vm/[boxnumber] and copy the default busy, unavailable, and greeting messages (found in /var/lib/asterisk/sounds) to this directory.

Configuration is necessary in voicemail.conf, as well. Each mailbox should have an entry in the voicemail.conf file in this format.

```
[mailboxnumber] => [passkey],[User Name],[email@address.com]
```

```
6161 => 1234,James Murdoch,jmurdoch@widgets.com
```

The passkey will be used when the box owner needs to access his voice messages over the phone. The username is referenced by the Directory application (more on that in the section on directory.) The last entry is the email address the message waiting notification should be sent to.

Some general options are available in this file as well. We can set the file formats in which the message is stored.

```
format=gsm
```

will store the messages in gsm format

```
format=gsm|wav|wav49
```

will store the files in all three listed formats. Available formats are gsm, wav, wav49, and g723sf.

We can also set the originating email address for message waiting notifications.

```
serveremail=asterisk@widgets.com
```

A complete configuration file looks like this

```
;set some general options
[general]
format=gsm|wav

;define the voicemail boxen
[default]
6161 => 1234,James Murdoch,jmurdoch@widgets.com
6162 => 1234,Blue Dawg,bdawg@widgets.com
;and so on, for each available box.
```

Once the boxes are configured, we can send the caller to the users box like this:

```
exten => 6161,3,Voicemail,u6161
```

Prepending 'u' to the box number plays the unavailable message for the caller. We can also use b, to play the busy message, and s, to play no message at all.

A sister application to Voicemail is VoicemailMain. VoicemailMain is the voicemail system gateway for users checking their voicemail over the phone. It requires no configuration or arguments.

Directory

The Directory application attempts to guide the caller to the correct extension, if the caller does not know the extension number of the person he is trying to reach. Directory will ask the caller for the first three letters of the recipients last name, and attempt to match that with the names stored in voicemail.conf. If the caller response matches more than one entry (quite possible, since each digit matches three letters) the application will present the caller with the available choices.

Once the correct entry is determined, Directory will attempt to connect to the caller to the extension matching the voicemail box number selected. For this reason, it is recommended that voicemail box numbers match the main extension number for the box owner.

Directory takes one argument, the context the extension should be found in. For example, if the extensions for employee phones reside the 'employee' context, the Directory call should look like this:

```
exten => 2,1,Directory,employees
```

When an extension is determined, Directory will attempt to connect to that extension number within the employee context.

Sounds for Directory and VoicemailMain are stored in /var/lib/asterisk/sounds. The sound files may be replaced with custom files using the same names.

MeetMe

MeetMe is the conference bridging application for Asterisk. MeetMe creates 'rooms,' to which multiple callers can be connected and intercommunicate in a conference. MeetMe requires external configuration in the file meetme.conf. Each conference room available must be listed in meetme.conf in this format

```
conf => 4000
```

to create a room '4000.'

We can send callers to this room with this statement:

```
exten => 44,1,MeetMe,4000
```

The conference room number argument is optional. If omitted, the user will be prompted to enter a room number, which must exist in meetme.conf. A sister application, MeetMeCount, plays the number of callers presently in the room specified as an argument. It is generally used just before the caller is connected with the specified room.

Playback and Background

The Playback and Background applications are used to play audio files for the caller. The two applications are largely identical, with one difference. The playback application plays the audio file entirely for the caller, ignoring any caller input. The background application plays the audio file while listening for caller input. During the background application, if the caller dials an extension, the audio file will be stopped, and the caller will be immediately connected to the indicated extension.

The format for playback is like this:

```
exten => s,1,Playback,welcome
```

Unless the filename in prefixed with a '/' filenames are assumed to be relative to /var/lib/asterisk/sounds. Sound files should be stored in gsm encoding. Audio files can be generated with the Record application, or recorded externally and encoded into gsm (sox is a good program to encode files into gsm.) The .gsm suffix should be left off, as it is assumed by the application.

Record -

Record is an application to record a sound file in gsm. It takes one argument, the filename relative to /var/lib/asterisk/sounds. The stored filename will have .gsm appended to it, so that should not be specified.

To Record a sound file 'blah.gsm' for later playback, do:

```
exten => 1,1,Record,blah
```

It should be noted that the audio quality of recordings made this way currently leaves much to be desired, and it is better to use external software to record custom audio files for Asterisk.

StripMSD

StripMSD strips digits from a dialed extension. It takes one argument, the number of digits to strip starting from the beginning. This is useful primarily to strip digits used in internal routing from numbers before passing them forward. A common use is to strip the 9 from a number before routing a call to an outside line. The format is:

```
exten = 1234,StripMSD,1
```

after this application, the call will now match a different extension, 234, as the one will be removed. See the section on extension matching to see this in common practice.

MusicOnHold

The Music On Hold application provides background audio for parked or held calls, and during call transfers. Audio files are stored in the common mp3 format, typically in /var/lib/asterisk/mohmp3. Multiple 'classes' can be configured and used, each with different audio file collections.

Configuring Music on Hold

Classes are configured in /etc/asterisk/musiconhold.conf. Each class statement comes in this format:

```
[classname] => mp3:[/full/path/to/mp3/dir], [extra args to mpg123]
```

Classname will be the name referenced in extensions.conf (see SetMusicOnHold)

Currently, two main playback modes exist, mp3 and quietmp3. Quietmp3 is identical to mp3, except that it reduces the volume by 75 percent. The path should lead to an accessible directory containing one or more mp3 files. By default, files will be played in alphanumeric order. Arguments can be passed to mpg123, after the path definition. A common use would be to put mpg123 in shuffle or random mode with the '-z' options

```
random => mp3:/var/lib/asterisk/mohmp3, -z
```

FYI - we are already using the -q, -s, --mono, and -r options, and setting the sampling rate to 8000. Be careful not to override these options blindly.

IMPORTANT: The Asterisk Music On Hold architecture depends on the very common free mpeg audio decoder program 'mpg123.' Though most distributions ship with this program by default, it may need to be installed on the local system. Some newer systems have replaced mpg123 with a work-alike program called 'mpg321.' This program does not work for Asterisk Music On Hold, because it does not properly implement some of the resampling features used to generate phone quality mono audio. If you experience 'dragging' audio files that play back at lower speeds than they should, this is most likely the problem.

Using Music on Hold

The Music On Hold system is operated using three application modules. The primary application is SetMusicOnHold, which sets the music class for the caller. This is commonly called within the special extension 's' while setting up general call settings for a given context. Once the class has been set, anytime the caller is parked or put on hold for a call transfer, the music specified in their set class will be played. We set the MusicOnHold class as a standard extension priority. The application takes on argument, the class.

```
exten => s,1,SetMusicOnHold,default
```

This would make the first priority to set their music class when we answered an incoming line.

There are two other Music On Hold applications available. WaitMusicOnHold will wait for a specified time while playing music. The only argument is the time to wait. This is analogous to the Wait application, only differing in that it plays the Music On Hold for caller. WaitMusicOnHold requires that the Music class be previously set, otherwise it behaves exactly as Wait (waiting for a specified amount of time, silently.)

```
exten => s,1,WaitMusicOnHold,20
```

This plays Music On Hold for a previously defined class for 20 seconds, before moving to the next priority.

MusicOnHold will play MusicOnHold for the caller indefinitely. There is no option to retrieve the caller from this application. It is intended primarily for testing the Music On Hold system. It takes on argument, the Music On Hold class.

```
exten => 1,3,MusiconHold,default
```

ParkedCall

Call parking is an unusual application in that it is included implicitly in the dial plan if parking.conf exists. Call parking can be configured to define the extension to transfer a call to to park it, the range of extensions parked calls will be transferred to, and the name of the context parked call extensions will exist in.

The format of the parking.conf file is:

```
;begins the file
[general]
;define the extension to dial to park a call
parkext => 700
;the destination extensions for parked calls
parkpos => 701-720
;the context parked calls include in
contect => parkedcalls
```

Enabling call parking and retrieval for a given context is as simple as including the defined parked calls context in the enabled context.

```
[default]
include => parkedcalls
```

To park a call, simply transfer them to the defined extension (in this example, 700). The extension to retrieve the call at will be played, then asterisk will disconnect them. To retrieve the call, simply dial the stated extension from any phone that has access to the parked call extension.

Absolute/Digit/Response Timeout

The Timeout applications allow you to specify the amount of time Asterisk waits for responses in certain situations. All applications take one argument, the amount of time to wait.

The AbsoluteTimeout application sets an absolute timeout for a call. When this timeout is reached, the call will be dumped. This can be useful, for example, when setting up an information line that could be tied up indefinitely by a phone not properly hung up.

```
exten => s,1,AbsoluteTimeout,600
```

This sets the absolute timeout to 600 seconds, or ten minutes. You might use this if you have an informational recording 3 minutes long, set to loop indefinitely. After three plays (and change) the call will be dumped, as we can safely assume the caller thinks their phone is hung up.

The ResponseTimeout application sets the amount of time to wait for a response from the caller. If we are waiting for the caller to dial an extension, we can set a response timeout to do something if they never respond. After the timeout is reached, Asterisk will jump to the special extension 't' for the current context. See **Special Extensions** for more information.

```
exten => s,1,ResponseTimeout,10
```

The DigitTimeout application sets the timeout between digits pressed. Once the timeout is reached, Asterisk will attempt to find the extension indicated by the digits that were received. If a full extension is received, Asterisk will jump to it immediately, so in most cases, if the digit timeout is reached the extension will be invalid, and Asterisk will look for special extension 'i.' See **Special Extensions'** for more information.

```
exten => s,1,DigitTimeout,10
```

Appendix C - syntax in extensions.conf

This section details the specific syntax and available options for extensions.conf.

Extensions.conf is broken up by context. A special context [general] is reserved, used for certain variables that may be set globally. It is currently unused.

Contexts start with a context definition in this format:

```
[context]
```

Each context contains a number of keywords, followed by thier arguments, in this format:

```
keyword => arguments
```

The following keywords are available -

ignorepat - specifies a digit (or digits) to 'ignore' These digits are received, but the dial tone continues.
switch - specifies a remote server to ask about extensions which are unknown to the local PBX.
include - specifies a context to include in the current context, with optional time arguments
exten - specifies one line of an extension configuration.

Usage of these keywords

Each of the keywords takes arguments unique to that keyword.

ignorepat takes a digit or digits to be ignored.

```
ignorepat => 9
```

will continue the dialtone after a user presses '9'. Note that 9 will still be part of the extension dialed. The dialtone will continue, but the digit is recieved and processed as ususal.

switch takes as it's argument a remote Asterisk server, to be contacted over IAX. If an extension is dialed on the local PBX that is unknown, and a switch statement exists, Asterisk will query the remote server to attempt to bridge the call to that extension. If the extension is unknown to the remote, or the remote server cannot be contacted, the extension will be handled as invalid. The format of 'switch' is

with no password:

register => IAX/client@asterisk.widgets.net/context

with a password (plaintext or MD5):

register => IAX/client:password@asterisk.widget.net/context

using RSA:

register => IAX/client:[keyname]@asterisk.widgets.net/context

where context is the context the remote server should look in to find the extension.

In order to use an IAX switch, the remote server must have a user definition for the incoming client. See **Configuring Channels - IAX** for more details about users and authentication.

include takes a context to be included, and optionally a time argument.

include => anothercontext

include => anothercontext|time-range|day-range|days of month|months

include => anothercontext|09:00-18:00|mon-wed,fri|*|1-11

exten is the keyword to define extensions. Each extension statement follows this format:

exten => [extension],[priority],[application],[arguments]

The first argument is a number, one of the special extensions 's', 'i', or 't', or a wildcard match. Wildcard matches begin with an underscore (_). Wildcard matches can contain these characters:

Any dialable digit, including hash (#) and asterisk (*).

N - matches any number greater than 1

X - matches any digit

The special extensions s, i, and t are used under certain conditions.

s - the 'start' extension. If a caller enters a context going to no particular extension, they will be connected to extension s. For example, if we answer an incoming call but have no specific rule to place that call in an extension, the call will go to 's'

t - the 'timeout' extension. If a call times out while waiting for a response, the extension 't' will be used. Applications exist to configure timeouts for various conditions. See Appendix B.

i - if an extension is dialed that does not exist, the extension 'i' will be used.

Appendix D - Files installed by Asterisk

Asterisk attempts to comply with the Linux Filesystem Hierarchy standard, and follow generally accepted conventions for file placement on a Linux system. Asterisk installs files in several locations on the system.

/etc/asterisk -
The directory /etc/asterisk contains configuration files for Asterisk and it's application modules.

/usr/lib/asterisk -
This directory contains loadable modules for channels, codecs, and applications for Asterisk

/var/spool/asterisk -
/var/spool/asterisk is used to store accumulated voicemail files for users. Custom voicemail messages created by users are also stored here, for ease of moving or removing a complete voicemail account.

/var/lib/asterisk -
Several types of files are stored under this directory, primarily data files such as music on hold and playback messages. The following directories exist in a default install, relative to /var/lib/asterisk:

./mohmp3 - the default location of music on hold files. Audio files in the mp3 format can be placed in this directory to be used by the MusicOnHold application. See the Applications section for more information.

./agi-bin - the location to store Asterisk Gateway Interface scripts

./keys - where public and private RSA keys are stored. RSA keys may be used to facilitate secure authentication for the Inter-Asterisk Exchange protocol.

./images - a place to put images which may be sent over the IAX protocol to clients which support this feature.

./sounds - the location of sound files to be played by Asterisk for voice prompting.

/usr/sbin/ -

asterisk - the program binary.
astgenkey - a program to generate RSA keys for use with asterisk
addmailbox - a program to create new voice mail boxes.

Appendix E - using Asterisk features

Asterisk features three way calling, call transfer, call parking, and music on hold.

To transfer a call -

Flash-hook (press the hang-up button briefly. On cordless phones, there may be a flash button instead.) You receive a special stutter dial tone. Dial the extension to transfer to, and hangup.

To three way a call -

Flash-hook. You receive a stutter dial tone. Dial the extension you wish to three -way into your call. When the other party answers, flash-hook again to connect all three channels. While you are waiting for the third party, music on hold is payed for the other caller if available.

Call parking -

Call parking is done by transferring a caller into a special extension, defined in parking.conf. Flash-hook and dial the parking extension. The server will state the parking slot by number, and disconnect. Dial the stated number to retrieve the caller. Music on hold will be played for the parked caller if available.

Call forwarding -