Introduction to the Asterisk Open Source PBX

http://asterisk.org

Mark Spencer

Linux Support Services, Inc.
http://www.linux-support.net

Presented first at Libre Software Meeting 2002
Bordeaux, France
July 9, 2002

Introduction to Asterisk

Asterisk is a fully Open Source, hybrid TDM and packet voice PBX and IVR platform.

Asterisk is and has been Open Source under GNU GPL (with an exception permitted for linking with the OpenH323 project, in order to provide H.323 support). Commercial licensing is available from Linux Support Services, Inc. (http://www.linux-support.net) for applications in which the GPL is inappropriate.

Unlike many modern "soft switches", Asterisk can use both traditional TDM technology and packet voice (Voice over IP and Voice over Frame Relay) protocols. Calls switched on TDM interfaces provide lag-less TDM call quality, while retaining interoperability with VoIP packetized protocols.

Asterisk acts as a full featured PBX, supporting virtually all conventional call features on station interfaces, such as Caller*ID, Call Waiting,Caller*ID on Call Waiting, Call Forward/Busy, Call Forward/No Answer, Call Forward Variable, Stutter Dialtone, Three-way Calling, Supervised Transfer, Unsupervised Transfer, ADSI enhancements, Voicemail, Meet-me Conferencing, Least Cost Routing, VoIP gatewaying, Call Detail Records, etc. At the same time, Asterisk provides full IVR capability, programmable at several layers, from a low-level C interface, to high level AGI scripting (analogous to CGI) and extension logic interfaces. Asterisk IVR applications run seamlessly from one interface to another, and need not know anything about the physical interface, protocol, or codec of the call they are working with, since Asterisk provides total abstraction for all those concepts.

Supported Hardware

Asterisk supports a variety of hardware interfaces for bringing in telephony channels to a Linux box. In order of best supported to least supported:

Zaptel (Zapata Telephony) interface

The zaptel telephony infrastructure was jointly developed by Mark Spencer of Linux Support Services, Inc. and Jim Dixon of Zapata Telephony. The zaptel interface breaks with the established conventional wisdom by using the host processor to *simulate* the TDM bus which is typically built into other telephony interfaces (e.g. Dialogic and other H.100 vendors). The resulting *pseudo-TDM* architecture is a system which requires somewhat more processor, at a substantial savings in hardware cost and increase in flexibility. Resources such as echo cancelers, HDLC controllers, conferencing DSP's, DAX's and more are replaced by software equivalents. Just like traditional hard-TDM interfaces, switching is done in near-realtime, and call qualities are substantially the same. The pseudo-TDM architecture also can extend the TDM bus across Ethernet networks. Zaptel devices also support data modes on clear channel interfaces, including Cisco HDLC, PPP, and Frame Relay

A number of Zaptel compatible interfaces are available from Linux Support Services. At the time of this writing, the following cards are available:

X100P - Single FXO Analog Interface (PCI)
T100P - Single T1/PRI Interface (PCI)

E100P - Single E1/PRA Interface (PCI, exp. Q3 2002)
T400P - Quad T1/PRI Interface (PCI)
E400P - Quad E1/PRA Interface (PCI)
S100U - Single FXS Interface (USB)

Linux Telephony Interface

The Linux Telephony Interface, was developed primarily by Quicknet, Inc. with help from Alan Cox. This interface is geared toward single analog interfaces and also provides support for low bit-rate codecs. The following products are known to work with Asterisk:

Quicknet Internet Phonejack (ISA, FXS)
Quicknet Internet Phonejack PCI (PCI, FXS)
Quicknet Internet Linejack (ISA, FXO or FXS)
Quicknet Internet Phonecard (PCMCIA, FXS)
Creative Labs VoIP Blaster (limited support)

ISDN4Linux

The ISDN4Linux interface is used primarily in Europe to bring lines into Asterisk from BRI interfaces. Any adapter that is supported by ISDN4Linux should work with Asterisk.

OSS / ALSA Console Drivers

The OSS and ALSA console drivers allow a single sound card to be used as a "console phone" for placing and receiving test calls. Using autoanswer/autohangup, the console can also be used to create an intercom.

Adtran Voice over Frame Relay

Asterisk supports Adtran's proprietary Voice over Frame Relay protocol. The following products are known to talk to asterisk using VoFR. You will need a Sangoma Wanpipe or other frame relay interface to talk to them:

Adtran Atlas 800
Adtran Atlas 800+
Adtran Atlas 550

Supported VoIP Protocols

Asterisk supports three VoIP protocols, two industry standards and one originally developed specifically for Asterisk, but now used by a number of other hardware and software devices. In the order of most supported to least supported:

Inter-Asterisk Exchange (IAX)

IAX is the defacto standard VoIP protocol for Asterisk networking. Perhaps its most impressive feature is its transparent interoperation with NAT and PAT (IP masquerade) firewalls, including placing, receiving, and transferring calls and registration. In this way, PBX's and phones can be totally portable. Just plug them in anywhere on the Internet and they'll register with their home PBX, instantly routing extensions to them appropriately. IAX is extremely low-overhead (four bytes of header, as compared to at least 12 bytes of header for RTP based protocols like SIP and H.323). IAX control messages are also substantially smaller. IAX supports internationalization, permitting the requesting PBX or phone to receive content from the providing PBX in its preferred language if available. IAX also supports authentication on incoming and outgoing calls, with fine-grained control to limit access to specific portions of the dialplan. Using IAX dialplan polling, the dialplan for a collection or cluster of PBX's can be centralized, with each PBX only needing to know its local extensions, and able to query the central PBX for further assistance as required.

Session Initiation Protocol (SIP)

SIP is the IETF standard for VoIP. Its call control syntax resembles SMTP, HTTP, FTP and other IETF protocols. The back-end runs on Real Time Protocol (RTP). SIP is widely regarded as the up-and-coming standard in VoIP due to its relative simplicity in comparison with H.323 and human-readability. As of the time of this writing, the Asterisk SIP stack has interoperated with a multiple vendors including SNOM and Cisco.

H.323

H.323 is the ITU standard for VoIP. Support for H.323 in Asterisk was contributed by Michael Mansous of InAccess Networks (http://www.inaccessnetworks.com), and is based on the OpenH323 project (http://www.openh323.org).

Codec and file formats

Asterisk provides seamless and transparent translation between all of the following codecs:

16-bit Linear	128 kbps
G.711u (μ-law)	64 kbps
G.711a (A-law)	64 kbps
IMA-ADPCM	32 kbps
GSM 6.10	13 kbps
MP3	(variable, decode only)
LPC-10	2.4 kbps

In addition, other codecs, such as G.723.1 and G.729 can be passed through transparently.

Asterisk seamlessly and transparently supports a variety of file formats for audio files. When a file is played on a channel, Asterisk chooses the "least expensive" format for that device. Supported formats include:

"raw"	16-bit linear raw data
"pcm"	8-bit mu-law raw data
"vox"	4-bit IMA-ADPCM raw data
"mp3"	MPEG2 Layer 3
"wav"	16-bit linear WAV file (8000 Hz)
"WAV"	GSM compressed WAV file (8000 Hz)
"gsm"	Raw GSM compressed data
"g723"	Simple G723 format with timestamp

Why Asterisk is Important

Asterisk represents a highly valuable piece of software for a number of reasons:
Extreme cost reduction - Combined with low-cost Linux telephony hardware, Asterisk can be used to create a PBX at a fraction of the price of traditional PBX and key systems, while providing a level of functionality exceeding that of many of the most expensive systems available.

Control - Asterisk allows the user to take control of their phone system. Once a call is in an Linux box with Asterisk, *anything* can be done to it. In the same way that Apache gives the user fine-grained control over virtually every aspect of its operation (and its Open Source nature gives even more flexibility), the same applies to Asterisk (in fact, it is my hope that Asterisk will one day become the Apache of the telephony world, greatly surpassing the market share of the proprietary players).

Rapid deployment and development - Asterisk allows PBX's and IVR applications to be rapidly created and deployed. Its powerful CLI and text configuration files allow both rapid configuration and real-time diagnostics.

Rich, broad feature base - Because Asterisk is Open Source and is implemented in software, not only does it provide features such as voicemail, voice menus, IVR and conferencing which are very expensive for proprietary systems, but it also allows new features to be added rapidly and with minimal effort.

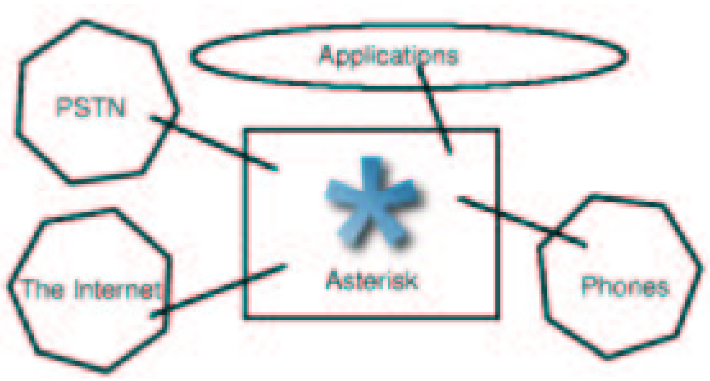
Customization - Through its internationalization support, configuration files, and source code, every aspect of Asterisk can be tweaked. For example, Asterisk's codes for call features could be changed to match an existing system.

Dynamic Content Deployment - In the same way that web servers like Apache allow a user to deploy dynamic content, such as account information, movie show times, etc, on the web, Asterisk permits you to deploy such dynamic content over the telephone, with the same ease as CGI.

Extremely Flexible Dialplan - Asterisk's unusually flexible dialplan allows seamless integration of IVR and PBX functionality. Many of Asterisks existing features (and desired features of the future) can be implemented using nothing more than extension logic. Asterisk supports a mix of extension lengths.

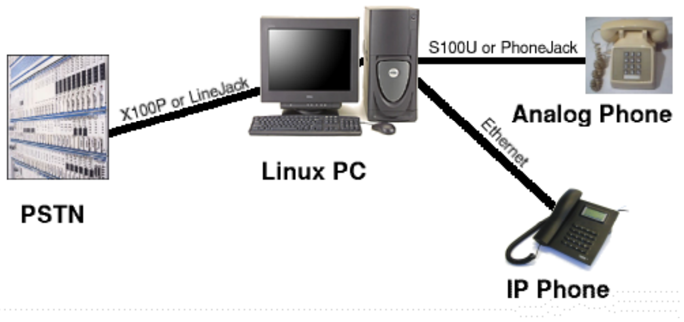
Asterisk as a Black Box

Asterisk connects any phone, phone line, or packet voice connection, to any other interface or service, using Asterisk applications. Asterisk's codec, file format, channel, CDR, application, switch and other Apia's distance the developer/deployer from the intricacies of the entire system.

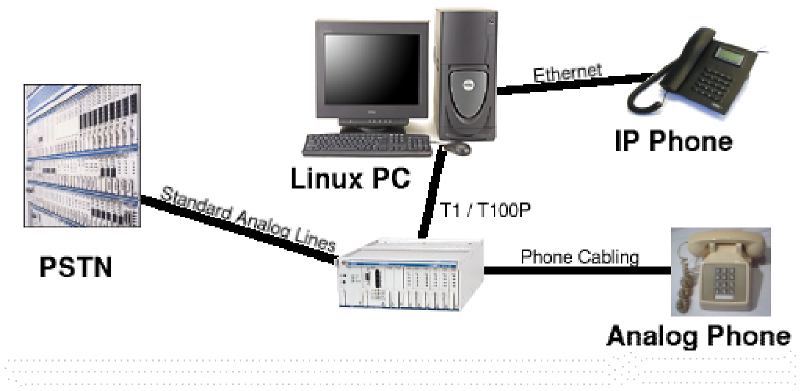


Example Networks

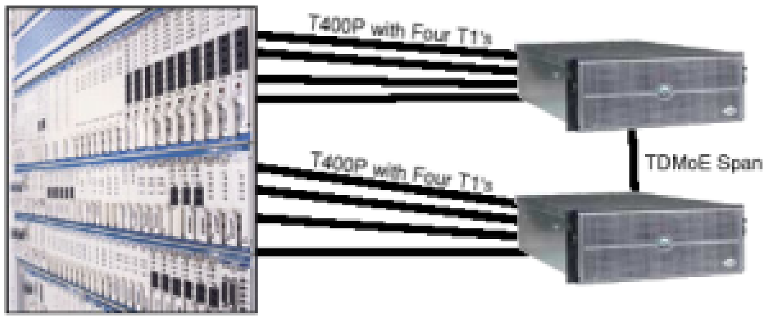
It may be helpful to see some sample diagrams of real-world Asterisk installations to see just how you do, in fact, get calls into and out of a Linux box.



One of the beauties of Asterisk is its ability to scale from very small architectures to very large architectures relatively seamlessly. An X100P from LSS, or a LineJack from Quicknet can be used to bring in an incoming phone line. A telephone can be hooked up via an S100U from LSS, or a Phonejack from Quicknet. Or, alternatively, an IP phone, such as the SNOM 100 could be used, or even a Cisco ATA186 to bring out phone lines. In this way, Asterisk can be used to create a very low-density PBX framework.

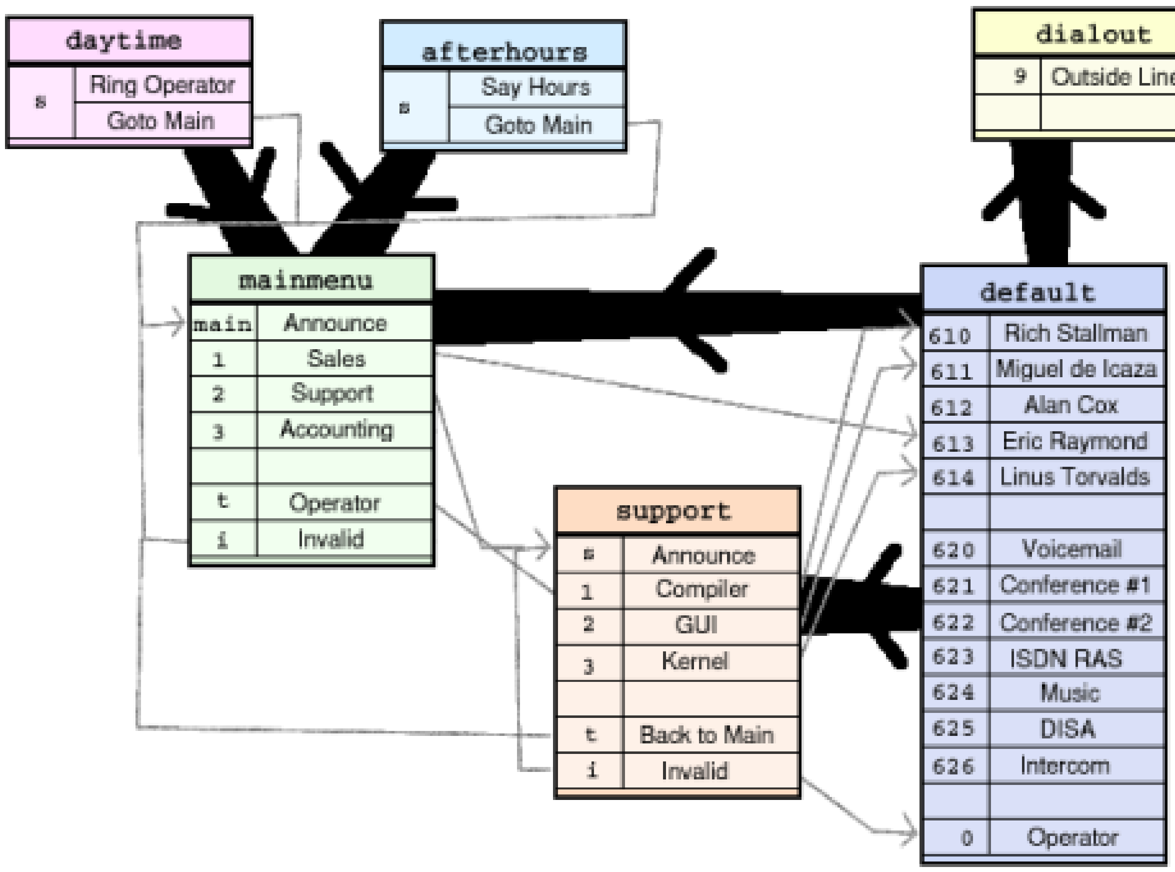


This diagram shows another variation on the Asterisk PBX theme. Here, 8 incoming phone lines, and 16 outgoing stations are connected to a channel bank, which is connected to Asterisk using a T1 cross over cable (remember, in this diagram T1 is just a technology, not a service that you have to pay for). A T100P from LSS then performs all the PBX functionality, using the channel bank as a simple analog to digital converter. Channel banks are fairly commodity items in the telcom industry, and just about any brand and vendor should be compatible with Asterisk.



Asterisk can also be used in high density, redundant applications. In this example, two rackmount servers could be providing conferencing and/or IVR, using a TDMoE span for communication between the two machines. Since Asterisk can perform a graceful shutdown, having redundancy in your Asterisk IVR networks means you can take down one machine at a time and upgrade the software and then bring it back up, while you take down another machine to upgrade its software.

The Asterisk Dialplan



It is Asterisk's dialplan itself which provides much of its flexibility. The dialplan is what determines how calls are routed through an Asterisk system. It is composed of a group of (generally connected) *extension contexts*, which are simply collections of extensions. An extension may appear in more than one context, and one context may include another (with an optional date/time limitation). Contexts may also reference external switches (e.g. The IAX switch), permitting the dialplan to be externally augmented.

The above diagram represents a fictitious PBX for the GNU/Linux community. Each context is in its own table, listing extensions and what those extensions represent. The fine, gray lines represent Links from an extension in one context to another. The thick lines represent the inclusion of one context in another (The arrow points **from** the context which is being included).

Creative use of contexts can be used to implement many features, some of which are illustrated in the above diagram:

Security - Dialplans can be segmented for allowing certain extensions to only be accessible from certain interfaces or VoIP users. In the example dialplan, local phones would be placed in the "dialout" context, permitting them to make outgoing calls, which inbound callers would be placed in the "mainmenu" context, precluding them from accessing the "9" extension for an outbound line.

Voice Menus - Since extensions can be any length and can be reused in different contexts, voice menus can be created. Notice how extensions 1,2, and 3 are used in the "mainmenu" context for sales, support, and accounting, and then again in the "support" menu for compiler, GUI, and kernel. By including the "default" context in both the mainmenu and support menu contexts, the calling user can at any point enter the extension of the person they are trying to reach.

Authenticated Services - Contexts can be used to verify a password or passcode, by permitting one valid extension to a given service and a pattern match going to failure.

PBX Multihosting - Obviously multiple company PBX's could be hosted at one site, with the DID (Direct Inward Dialing) number deciding which context the call would begin in.

Callback Services - Extension contexts, external scripts, and app_qcall can be combined to implement callback services.

Day/night Modes - Since an include can be time/day dependent, behavior can be modified for different times and days. For example, in the above diagram, the "daytime" context might be included from 9 to 5 mon-fri, and then the "afterhours" included afterwards, so that during the day the operator phone can ring, and in the evening an "after hours" message played and immediately sent to enter an extension.

Dialplan Centralization - Using the "IAX" internal switch, the dialplan of a remote Asterisk server can be polled, so that the dialplan for a complex system can be centralized on a single server.

Asterisk Extensions

Extension contexts are made of extensions. Extensions themselves consist of a set of applications (optionally with arguments) and priorities. Beginning with priority one, the applications are executed generally in order until the call is hung up. Asterisk applications and AGI scripts may modify the call flow, however. Extensions may route based on both called and caller number.

Extension names may contain numbers, letters, *, and #. They can be numbers, or text expressions (like "main"). If preceded by a "_", the number is considered a pattern match, where 'N' represents any digit 2-9, 'X' represents any digit 0-9, and '.' represents a match everything.

Certain extension names are reserved for specialized purposes. Specifically:

s: The "start" extension. A call which does not have digits associated with it (for example, a loopstart analog line) begins at the "s" extension.

t: The "timeout" extension. When a caller is in a voice menu and does not enter the proper number of digits (or enters no digits at all), the timeout extension is executed. If the timeout extension is omitted, then the caller is disconnected.

i: The "invalid" extension. When a caller enters an invalid extension number, this extension is executed. If omitted, the caller is disconnected.

o: The "operator" extension. When a caller zero-s out of voicemail to get an operator, this extension is executed. If not present, the caller is not permitted to zero out of voice mail.

h: The "hangup" extension. If present, this extension is executed at the end of call (i.e. when the caller hangs up or is hung up upon). Applications executed in this extension should *not* try to do anything to the channel itself since it has been hung up already, but can use it for logging, executing commands, etc.

Extension logic is best illustrated by examples:

Simple extension with Voicemail

```
exten => 600,1,Dial,Zap/9|15
exten => 600,2,Voicemail,u600
exten => 600,102,Voicemail,b600
```

The above lines implement a simple extension with voicemail. When a caller was directed to extension 600, first, Asterisk would try dialing the Zap/9 interface (channel 9 of the zaptel system) for up to 15 seconds. If there was no answer the user would be dumped to unavailable voicemail. The "Dial" application is somewhat special in that it allows you to optionally setup a separate rule for a "busy" (n + 101 instead of n+1), so in this case, if the user gets a busy, they are sent to voicemail with a "busy" message played instead of "unavailable".

Extension with anti-ex-girlfriend

```
exten => 600/2565551212,1,Congestion
exten => 600,1,Dial,Zap/9&IAX/marko/s|15
exten => 600,2,Voicemail,u600
exten => 600,102,Voicemail,b600
```

[Note: This example is derived from an actual extension used by one of my friends]

This example is similar to the previous one, except it exercises two additional features of Asterisk -- Caller*ID matching, and the ability to call multiple interfaces at the same time.

If the incoming caller has the Caller*ID of 256-555-1212, then the caller is immediately given "congestion tone", as if the number were invalid or having trouble. Otherwise, the Dial application is used to try to call both Zap/9 and another IAX host, "marko", for up to 15 seconds. If neither answers, then the call is sent to voicemail, preceded by an "unavailable" message. If the interfaces are busy, they are sent to the "busy" voicemail instead.

Simple Call Queue

```
exten => 600,1,Dial,Zap/9|15
exten => 600,2,Voicemail,u600
exten => 600,102,WaitMusicOnHold,5
exten => 600,103,Goto,1
```

This example demonstrates use of Asterisk extension logic to create a simple call queue, where the user is put on hold (with music in this case) until the called party becomes available, or is unable to answer the phone.

When a caller enters this extension, it immediately tries to dial on the Zap/9 interface for up to 15 seconds. If there is no answer, the caller is sent to voicemail immediately. However, if the interface is busy, they are played hold music for 5 seconds, and then sent back to the first extension to try Zap/9 again. Note that this is not the same as true call queuing since there is no prioritization of the calls (FIFO) implemented.

Operator Extension for a Small Business

```
exten => 0,1,Dial,Zap/9|15
exten => 0,2,Dial,Zap/10&Zap/11&Zap/12|15
exten => 0,3,Playback,companymailbox
exten => 0,4,Voicemail,600
exten => 0,5,Hangup
```

This example implements an operator extension for a small business. Here, when the "0" extension is executed, first the "real" operator is called on Zap/9. Then, if there is no answer, or the person is busy, then three other people are dialed for up to 15 seconds. If there is no answer or they are all busy, the user is asked to leave a message in the company mailbox, and then dumped into the operator's voicemail, without their normal announcement being played at all.

Outgoing Line with Least Cost Routing

```
exten => _9NXXXXXX,1,Dial,Zap/g2/BYEXTENSION
exten => _9NXXXXXX,2,Dial,IAX/oh/BYEXTENSION
exten => _9NXXXXXX,3,Congestion
```

This example demonstrates the use of pattern matching and helps show how everything in Asterisk is treated as an extension, be it an outgoing line, incoming interface, or voice menu. Here, we first try using any interface in "group 2" first to connect the call. If that is unavailable, we try calling through another IAX host called "oh". If that fails, the congestion tone is executed.

Main Voice Menu

```
exten => s,1,Wait,1
exten => s,2,Answer
exten => s,3,DigitTimeout,5
exten => s,4,ResponseTimeout,10
exten => s,5,Background,intro
exten => s,6,Background,instructions

exten => 1,1,Goto,sales|s|1
exten => 2,1,Goto,support|s|1

exten => i,1,Playback,pbx-invalid
exten => i,2,Goto,s|6

exten => t,1,Goto,0|1
```

This example implements a voice menu. When an incoming call is received, Asterisk waits for one second (to give a little bit of ring to the caller), answers the line, sets the digit and response timeouts to reasonable numbers, and then plays an introduction along the lines of "Thank you for calling GNU/Linux headquarters." in the background, while waiting for the user to enter an extension. After the introduction, instructions are played, for example "If you know the extension of the party you wish to dial you may enter it at anytime, otherwise press 1 for sales or 2 for support." If the party does not enter any extension, they are sent to the operator (whose extension is omitted for brevity). If they enter an invalid extension, they are played a message to inform them the extension is invalid and then presented with the options again.

AGI integration for routing

```
exten => s,1,AGI,agi-lookup.agi
exten => s,2,Background,intro

exten => 100,1,AGI,agi-save.agi
exten => 100,2,Dial,Zap/9|15
exten => 100,3,Voicemail,u100
...
exten => 120,1,AGI,agi-save.agi
exten => 120,2,Dial,Zap/24|15
exten => 120,3,Voicemail,u101
```

This set of extensions is designed to allow a caller to speak with whomever the last person was that they spoke with, without having to remember their extension. An incoming caller is first sent to the agi-lookup script, which checks their Caller*ID against a database. If the caller has already spoken with someone before, the "agi-lookup" script sends them to the last extension they called. Otherwise, the call progresses normally. When the user connects to someone's extension, the "agi-save" script saves the person they are about to speak with, for future reference by the agi-lookup script the next time they call.

Ringback Example

```
[ringback]
exten => s,1,Ringing
exten => s,2,Wait,1
exten => s,3,Congestion
exten => h,1,System,callback

[default]
exten => 1234,1,DISA,4321|trusted
exten => 1235,1,Goto,ringback|s|1
```

[Note: This is the same format as what I use when I'm in France to call back to the U.S.]

This example shows how to create a ringback system. When extension 1235 is dialed, the caller is sent to the ringback extension, which plays ringtone for one second, followed by congestion tone. Note that the call is *never* answered at this point. Finally, when the caller hangs up, the "callback" script is executed, which would create an appropriate entry in /var/spool/asterisk/qcall for the outbound call.

Then, app_qcall would call a pre-specified number, and when the line was answered, would dump the caller into extension 1234, which is the DISA (Direct Inward System Access). After entering a password (4321), the called party could now make outbound calls through the "trusted" extension context.

Configuration Files

Asterisk is configured through a number of configuration files located in /etc/asterisk. Config files use a consitant format, which consists of one or more sections (delimited by the section title in brackets, like "[section]"), and a number of "variable = value", or "object => parameters." Comments are delimited by ';' instead of '#' since # could often be used in extensions. Here is a sample:

```
[section1]
variable1=value1  ; assign variable1 the value of value1
variable2=value2  ; assign variable2 the value of value2

[section2] ; New section now
variable3=value3 ; assign variable 3 the value of value3
object1 => param2 ; create an object1 type with param2 parameters
```

When declaring objects or assigning variables, the "=" and "=>" may be used interchangeably. They are parsed identically and their use is only designed to make configuration files more readable. This format, coincidentally, fairly closely resembles the "win.ini" format. It was designed to be easily machine and human parsable.

The grammar of Asterisk varies from one config file to another, however the configuration files can generally be divided into three categories which will be detailed more closely in the following sections:

Interface Configuration - These types of files typically configure channel interfaces which bind directly to physical hardware. Examples include adtranvofr.conf, alsa.conf, modem.conf, oss.conf, phone.conf, and zapata.conf.

Simple Groups - These types of files define the existence of various simple entities, like mailboxes, conference rooms, etc. Examples include extensions.conf, logger.conf, meetme.conf, musiconhold.conf, parking.conf, and voicemail.conf.

Individual Entities - These configs detail a number of typically unrelated entities, like clients and servers and are used most often for VoIP services. Examples include iax.conf, oh323.conf, and sip.conf.

An example of one of each of these types of config files can familiarize a user with their use, and the details of what options and fields are configurable in each one is generally best determined by looking at the sample configurations that are included with Asterisk.

Interface Configuration Example: zapata.conf

The zapata.conf file is one of the most adjustable interface configuration files. It controls all zaptel based interfaces that Asterisk is to be aware of, setting up their protocol, security and features.

At the time of this writing, the zapata.conf file has only one section, the "channels" section. Numerous options can be set for each channel to configure the caller-id, features, and starting context of each channel. When a channel is instantiated with the "channel => <x>" syntax, it is created with all the parameters specified *above* it. This allows you to setup parameters which are global to many channels, and then tweak some of them individually as you go through. For example:

```
[channels]
signalling = fxo_ks
callwaiting = yes
threewaycalling = yes
callerid = "Linus Torvalds" <(256) 555-1000>
channel => 1

callerid = "Richard Stallman" <(256) 555-2000>
callwaiting = no
channel => 2

callwaiting = yes
callerid = "Eric Raymond" <(256) 555-3000>
channel => 3

signalling = fxs_ks
callerid = asreceived
channel => 4-8
```

In this example, Three FXO signalled channels and four FXS signalled channels are created. The first three represent phones for Linus, Richard, and Eric respectively. We enable callwaiting and threeway calling before instantiating Linus's phone on channel 1. Then, we change the value of the callerid, and disable call waiting, then create Richard's phone. Note that the "callwaiting = no" only affects Richard and not Linus because it is specified *after* Linus's channel has been instantiated. After creating Richard's line, we re-enable call waiting, change the Caller*ID to be Eric, and create the third channel.

Next, we set the signalling to fxs_ks, set the callerid to be the value that is received from the line, and create four new channels. Theoretically these channels have callwaiting and threeway calling enabled, although those features only apply to station interfaces (that we signal with FXO signalling).

Simple Group Configuration Example: extensions.conf

In simple group configuration files, each section represents a group which contains individual lines, each instantiating an object, completely independantly of the other objects in the group. Sometimes the "general" section is reserved for global parameters that apply to whatever it is configuring in general. The most widely used example of this sort of configuration file is of course the extensions.conf file. In this file, the "general" section isn't actually used at this time, but may be in the future. All other sections represent contexts, and each line of each section either a) generates a single piece of an extension, b) declares an external switch, or c) includes another context.

```
[general]
;
; Rarely used, totally unimplemented general options done here.
;
[context1]
;exten => extension,priority,application ;[,args]
exten => s,1,Wait,3
exten => s,2,Answer
exten => s,3,Voicemail,u600
exten => 100,1,Dial,Zap/g2

[context2]
exten => 9,1,Dial,Zap/g2/9
include => context1
include => context3

[context3]
switch => IAX/myuser@otherhost/local
```

In the above example, we have three declared contexts. Context 1 has two extensions, "s" which has three steps, and "100" which has only one step. Context 2 includes a "9" extension as well as including everything in context1 and everything in context3 (note that includes are processed in the order listed, so if an extension's step were in both context1 and context3, the one in context 1 would be executed).

VoIP connections typically use the "individual entity" config grammar. The most commonly used file of this type is the "iax" configuration file. It contains a "general" section, with global parameters like the bind address and port, and then a number of entities, each of which has a "type" (user, peer, or friend) and its own set of parameters. Each entity is entirely independent of the other entities surrounding it.

```
[general]
portno = 5036

[mark]
type=friend
secret=mypass
callerid="Mark Spencer" <(256) 428-6000>
deny=0.0.0.0/0.0.0.0
permit=216.207.245.0/255.255.255.0
```

```
[greg]
type=friend
host=dynamic
```

In this case, the binding port is set to 5036 (the default), and two entities are declared -- a friend named "greg" and a friend named "mark". The "mark" user has a number of parameters declared like callerid, a secret, and a permit/deny list. Note that with some options, like "secret", there can be only one logical value, and so adding another simply replaces the first. But with some options like "permit" and "deny", multiple ones can be listed, each of which is applied in order.

Programmability

Although the details are beyond the scope of this document, it is important to at least give brief mention to Asterisk's programmability. In general, there are three levels at which to program Asterisk. From highest to lowest, they are:

Extension logic: Use the dialplan in extensions.conf to create simple applications, authorizations, etc. For more on extension logic, see the above examples.

Asterisk Gateway Interface (AGI): For more sophisticated or complex tasks, AGI presents a way for a user to write scripts in their language of choice (i.e. Perl, PHP, etc). An AGI script is launched from the Asterisk dialplan. It receives an environment setup on stdin, then can issue commands to stdout and read the results from stdin. In this way, no special bindings are required and any language that can write to stdout and read from stdin (i.e. all languages) will work.

C-level API: For hard core applications, channel drivers, file formats, etc there is the Asterisk C API, which gives full access to all the Asterisk internals.

Extra Resources

The information in this white paper is designed to provide an overview and introduction to the Asterisk system. However, it does leave out a great deal of detail. Fortunately, there are many resources available to the user to assist in planning, configuring, and developing with the Asterisk PBX. Details on asterisk commands and applications (including the format of their parameters) can be found at the Asterisk command line, by typing the following commands:

`help` (to show a list of commands)
`help <command>` (to explain how to use a particular command)
`show applications` (to show a list of Asterisk applications)
`show application <foo>` (to explain the purpose and arguments for a particular app)

Also some URL's are helpful in finding Asterisk documentation, news, and hardware:

http://www.asterisk.org - The main Asterisk web site contains links to documentation and architecture specifications *http://www.linux-support.net* - Linux Support Services, Inc. is the primary sponsor of Asterisk and has documentation, provides commercial support/development services related to Asterisk, and also sells low-cost zaptel compatible telephony hardware.

http://asterisk.drunkencoder.com - An independently maintained Asterisk FAQ

http://www.markocam.com - My web cam, where you can often watch me code on Asterisk in real-time.

Those interested in Asterisk or its development may also wish to sign up on the Asterisk mailing list, *asterisk@marko.net* by sending the word "subscribe" in the **body** of a message to *asterisk-request@marko.net*.