Asterisk Reference Information
Version SVN-trunk-r59090

Asterisk Development Team
Asterisk.org

March 22, 2007

Contents

Chapter 1

Introduction

This document contains various pieces of information that are useful for reference purposes.

1.1 License Information

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

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

contrib/firmware/iax/iaxy.bin: This file is Copyright (C) Digium, Inc. and is licensed for use with Digium IAXy hardware devices only. It can be distributed freely as long as the distribution is in the original form present in this package (not reformatted or modified).

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

more flexible licenses than GPL.

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

Specific permission is also granted to link Asterisk with OpenSSL and OpenH323.

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

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

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

+1.877.546.8963 (via telephone in the USA) +1.256.428.6000 (via telephone outside the USA) +1.256.864.0464 (via FAX inside or outside the USA) IAX2/misery.digium.com/6000 (via IAX2) licensing@digium.com (via email)

Digium, Inc. 150 West Park Loop Suite 100 Huntsville, AL 35806 USA

1.1.1 Hold Music

Digium has licensed the music included with the Asterisk distribution From FreePlayMusic for use and distribution with Asterisk. It is licensed ONLY for use as hold music within an Asterisk based PBX.

1.2 Security

1.2.1 Introduction

PLEASE READ THE FOLLOWING IMPORTANT SECURITY RELATED INFORMATION. IMPROPER CONFIGURATION OF ASTERISK COULD ALLOW UNAUTHORIZED USE OF YOUR FACILITIES, POTENTIALLY INCURRING SUBSTANTIAL CHARGES.

Asterisk security involves both network security (encryption, authentication) as well as dialplan security (authorization - who can access services in your pbx). If you are setting up Asterisk in production use, please make sure you understand the issues involved.

1.2.2 Network Security

If you install Asterisk and use the "make samples" command to install a demonstration configuration, Asterisk will open a few ports for accepting VoIP calls. Check the channel configuration files for the ports and IP addresses.

If you enable the manager interface in manager.conf, please make sure that you access manager in a safe environment or protect it with SSH or other VPN solutions.

For all TCP/IP connections in Asterisk, you can set ACL lists that will permit or deny network access to Asterisk services. Please check the "permit" and "deny" configuration options in manager.conf and the VoIP channel configurations - i.e. sip.conf and iax.conf.

The IAX2 protocol supports strong RSA key authentication as well as AES encryption of voice and signalling. The SIP channel does not support encryption in this version of Asterisk.

1.2.3 Dialplan Security

First and foremost remember this:

USE THE EXTENSION CONTEXTS TO ISOLATE OUTGOING OR TOLL SERVICES FROM ANY INCOMING CONNECTIONS.

You should consider that if any channel, incoming line, etc can enter an extension context that it has the capability of accessing any extension within that context.

Therefore, you should NOT allow access to outgoing or toll services in contexts that are accessible (especially without a password) from incoming channels, be they IAX channels, FX or other trunks, or even untrusted stations within you network. In particular, never ever put outgoing toll services in the "default" context. To make things easier, you can include the "default" context within other private contexts by using:

```
include => default
```

in the appropriate section. A well designed PBX might look like this:

```
[longdistance]
exten => _91NXXNXXXXXX,1,Dial(Zap/g2/${EXTEN:1})
include => local

[local]
exten => _9NXXNXXX,1,Dial(Zap/g2/${EXTEN:1})
include => default

[default]
exten => 6123,Dial(Zap/1)
```

DON'T FORGET TO TAKE THE DEMO CONTEXT OUT OF YOUR DEFAULT CONTEXT. There isn't really a security reason, it just will keep people from wanting to play with your Asterisk setup remotely.

1.2.4 Log Security

Please note that the Asterisk log files, as well as information printed to the Asterisk CLI, may contain sensitive information such as passwords and call history. Keep this in mind when providing access to these resources.

1.3 Hardware

1.3.1 Introduction

A PBX is only really useful if you can get calls into it. Of course, you can use Asterisk with VoIP calls (SIP, H.323, IAX), but you can also talk to the real PSTN through various cards.

Supported Hardware is divided into two general groups: Zaptel devices and non-zaptel devices. The Zaptel compatible hardware supports pseudo-TDM conferencing and all call features through chan_zap, whereas non-zaptel compatible hardware may have different features.

1.3.2 Zaptel compatible hardware

- Digium, Inc. (Primary Developer of Asterisk) http://www.digium.com
  - Analog Interfaces
    - TDM400P - The TDM400P is a half-length PCI 2.2-compliant card that supports FXS and FXO station interfaces for connecting analog telephones and analog POTS lines through a PC.
    - TDM800P - The TDM800P is a half-length PCI 2.2-compliant, 8 port card using Digium's VoiceBus technology that supports FXS and FXO station interfaces for connecting analog telephones and analog POTS lines through a PC.
    - TDM2400P - The TDM2400P is a full-length PCI 2.2-compliant card for connecting analog telephones and analog POTS lines through a PC. It supports a combination of up to 6 FXS and/or FXO modules for a total of 24 lines.
  - Digital Interfaces
    - TE412P - The TE412P offers an on-board DSP-based echo cancellation module. It supports E1, T1, and J1 environments and is selectable on a per-card or per-port basis.
    - TE410P - The TE410P improves performance and scalability through bus mastering architecture. It supports E1, T1, and J1 environments and is selectable on a per-card or per-port basis.
    - TE407P - The TE407P offers an on-board DSP-based echo cancellation module. It supports E1, T1, and J1 environments and is selectable on a per-card or per-port basis.
    - TE405P - The TE405P improves performance and scalability through bus mastering architecture. It supports both E1, T1, J1 environments and is selectable on a per-card or per-port basis.

* TE212P - The TE212P offers an on-board DSP-based echo cancellation module. It supports E1, T1, and J1 environments and is selectable on a per-card or per-port basis.
* TE210P - The TE210P improves performance and scalability through bus mastering architecture. It supports E1, T1, and J1 environments and is selectable on a per-card or per-port basis.
* TE207P - The TE207P offers an on-board DSP-based echo cancellation module. It supports E1, T1, and J1 environments and is selectable on a per-card or per-port basis.
* TE205P - The TE205P improves performance and scalability through bus mastering architecture. It supports both E1 and T1/J1 environments and is selectable on a per-card or per-port basis.
* TE120P - The TE120P is a single span, selectable T1, E1, or J1 card and utilizes Digium's VoiceBus technology. It supports both voice and data modes.
* TE110P - The TE110P brings a high-performance, cost-effective, and flexible single span togglable T1, E1, J1 interface to the Digium line-up of telephony interface devices.

1.3.3 Non-zaptel compatible hardware

- QuickNet, Inc. http://www.quicknet.net
  - Internet PhoneJack - Single FXS interface. Supports Linux telephony interface. DSP compression built-in.
  - Internet LineJack - Single FXS or FXO interface. Supports Linux telephony interface.

1.3.4 mISDN compatible hardware

mISDN homepage: http://www.isdn4linux.de/mISDN/

Any adapter with an mISDN driver should be compatible with chan_misdn. See the mISDN section for more information.

- Digium, Inc. (Primary Developer of Asterisk) http://www.digium.com

  - B410P - 4 Port BRI card (TE/NT)

- beroNet http://www.beronet.com
  - BN4S0 - 4 Port BRI card (TE/NT)
  - BN8S0 - 8 Port BRI card (TE/NT)
  - Billion Card - Single Port BRI card (TE (/NT with crossed cable))

1.3.5 Miscellaneous other interfaces

- Digium, Inc. (Primary Developer of Asterisk)
  - TC400B - The TC400B is a half-length, low-profile PCI 2.2-compliant card for transforming complex VoIP codecs (G.729) into simple codecs.

- ALSA http://www.alsa-project.org
  - Any ALSA compatible full-duplex sound card

- OSS http://www.opensound.com
  - Any OSS compatible full-duplex sound card

Chapter 2

Configuration

2.1 General Configuration Information

2.1.1 Configuration Parser

Introduction

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

These changes are general to the configuration parser, and works in all configuration files.

General syntax

Asterisk configuration files are defined as follows:

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

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

```
[section]
label1 = value1
```

```
label2 = value2
object => name

label3 = value3
label2 = value4
object2 => name2
```

In this syntax, we create objects with the settings defined above the object creation. Note that settings are inherited from the top, so in the example above object2 has inherited the setting for "label1" from the first object.

For template configurations, the syntax for defining a section is changed to:

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

The options field is used to define templates, refer to templates and hide templates. Any object can be used as a template.

No whitespace is allowed between the closing "]" and the parenthesis "(".

Comments

All lines that starts with semi-colon ";" is treated as comments and is not parsed.

The ";–" is a marker for a multi-line comment. Everything after that marker will be treated as a comment until the end-marker "–;" is found. Parsing begins directly after the end-marker.

```
;This is a comment
label = value
;-- This is
a comment --;

;-- Comment --; exten=> 1000,1,dial(SIP/lisa)
```

Including other files

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

```
#include myusers.conf
```

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

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

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

The exec directive is used like this:

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

Adding to an existing section

```
[section]
label = value

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

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

Defining a template-only section

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

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

Using templates (or other configuration sections)

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

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

```
[foo]
permit=192.168.0.2
host=asdf
deny=192.168.0.1

[bar]
permit=192.168.1.2
host=jkl
deny=192.168.1.1

[baz](foo,bar)
permit=192.168.3.1
host=bnm
```

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

```
[baz]
permit=192.168.0.2
host=asdf
deny=192.168.0.1
permit=192.168.1.2
host=jkl
deny=192.168.1.1
permit=192.168.3.1
host=bnm
```

Additional Examples

(in top-level sip.conf)

```
[defaults](!)
type=friend
nat=yes
qualify=on
dtmfmode=rfc2833
disallow=all
allow=alaw

#include accounts/*/sip.conf
```

(in accounts/customer1/sip.conf)

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

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

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

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

2.1.2 Asterisk.conf

Asterisk Main Configuration File

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

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

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

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

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

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

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

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

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


[options]
;Under "options" you can enter configuration options
;that you also can set with command line options

; Verbosity level for logging (-v)
verbose = 0

; Debug: "No" or value (1-4)
debug = 3
```

```
; Background execution disabled (-f)
nofork=yes | no

; Always background, even with -v or -d (-F)
alwaysfork=yes | no

; Console mode (-c)
console= yes | no

; Execute with high priority (-p)
highpriority = yes | no

; Initialize crypto at startup (-i)
initcrypto = yes | no

; Disable ANSI colors (-n)
nocolor = yes | no

; Dump core on failure (-g)
dumpcore = yes | no

; Run quietly (-q)
quiet = yes | no

; Force timestamping in CLI verbose output (-T)
timestamp = yes | no

; User to run asterisk as (-U) NOTE: will require changes to
; directory and device permissions
runuser = asterisk

; Group to run asterisk as (-G)
rungroup = asterisk

; Enable internal timing support (-I)
internal_timing = yes | no

; These options have no command line equivalent
```

```
; Cache record() files in another directory until completion
cache_record_files = yes | no
record_cache_dir = <dir>

; Build transcode paths via SLINEAR
transcode_via_sln = yes | no

; send SLINEAR silence while channel is being recorded
transmit_silence_during_record = yes | no

; The maximum load average we accept calls for
maxload = 1.0

; The maximum number of concurrent calls you want to allow
maxcalls = 255

; Allow #exec entries in configuration files
execincludes = yes | no

; Don't over-inform the Asterisk sysadm, he's a guru
dontwarn = yes | no

; System name. Used to prefix CDR uniqueid and to fill \${SYSTEMNAME}
systemname = <a_string>

; Should language code be last component of sound file name or first?
; when off, sound files are searched as <path>/<lang>/<file>
; when on, sound files are search as <lang>/<path>/<file>
; (only affects relative paths for sound files)
languageprefix = yes | no


[files]
; Changing the following lines may compromise your security
; Asterisk.ctl is the pipe that is used to connect the remote CLI
; (asterisk -r) to Asterisk. Changing these settings change the
; permissions and ownership of this file.
```

```
; The file is created when Asterisk starts, in the "astrundir" above.

;astctlpermissions = 0660
;astctlowner = root
;astctlgroup = asterisk
;astctl = asterisk.ctl
```

2.1.3 CLI Prompt

Changing the CLI Prompt

The CLI prompt is set with the ASTERISK_PROMPT UNIX environment variable that you set from the Unix shell before starting Asterisk

You may include the following variables, that will be replaced by the current value by Asterisk:

```
%d Date (year-month-date)
%s Asterisk system name (from asterisk.conf)
%h Full hostname
%H Short hostname
%t Time
%% Percent sign
%# '#' if Asterisk is run in console mode, '>' if running as remote console
%Cn[;n] Change terminal foreground (and optional background) color to specified
```

A full list of colors may be found in include/asterisk/term.h

On Linux systems, you may also use:

```
%l1     Load average over past minute
%l2     Load average over past 5 minutes
%l3     Load average over past 15 minutes
%l4     Process fraction (processes running / total processes)
%l5     The most recently allocated pid
```

2.1.4 Extensions

The Asterisk dialplan

The Asterisk dialplan is divided into contexts. A context is simply a group of extensions. For each "line" that should be able to be called, an extension

must be added to a context. Then, you configure the calling "line" to have access to this context.

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

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

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

When each action completes, it generally moves to the next priority (except for some modules which use explicitly GOTO's.

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

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

Example dialplan

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

Special extensions

There are some extensions with important meanings:

- s
  - What to do when an extension context is entered (unless overridden by the low level channel interface) This is used in macros, and some special cases. "s" is not a generic catch-all wildcard extension.
- i

  - What to do if an invalid extension is entered

- h
  - The hangup extension, executed at hangup

- t
  - What to do if nothing is entered in the requisite amount of time.

- T
  - This is the extension that is executed when the 'absolute' timeout is reached. See "show function TIMEOUT" for more information on setting timeouts.

And finally, the extension context "default" is used when either a) an extension context is deleted while an extension is in use, or b) a specific starting extension handler has not been defined (unless overridden by the low level channel interface).

2.1.5 IP Type of Service

Introduction

Asterisk can set the Type of Service (TOS) byte on outgoing IP packets for various protocols. The TOS byte is used by the network to provide some level of Quality of Service (QoS) even if the network is congested with other traffic.

SIP

In sip.conf, there are three parameters that control the TOS settings: "tos_sip", "tos_audio", and "tos_video". tos_sip controls what TOS SIP call signalling packets are set to. tos_audio controls what TOS RTP audio packets are set to. tos_video controls what TOS RTP video packets are set to.

There is a "tos" parameter that is supported for backwards compatibility. The tos parameter should be avoided in sip.conf because it sets all three tos settings in sip.conf to the same value.

IAX2

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

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

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

The tos* parameters also take numeric values.

The lowdelay, throughput, reliability, mincost, and none values are deprecated because they set the IP TOS using the outdated "IP precedence" model as defined in RFC 791 and RFC 1349. They still work in this version of Asterisk, but will be removed in future releases.

```
==========================================
Configuration Parameter Recommended
File Setting
------------------------------------------
sip.conf tos\_sip cs3
sip.conf tos\_audio ef
sip.conf tos\_video af41
------------------------------------------
iax.conf tos ef
------------------------------------------
iaxprov.conf tos ef
==========================================
```

Reference

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

IANA Assignments, DSCP registry Differentiated Services Field Codepoints http://www.iana.org/assignments/dscp-registry

To get the most out of setting the TOS on packets generated by Asterisk, you will need to ensure that your network handles packets with a TOS properly. For Cisco devices, see the previously mentioned "Enterprise QoS Solution Reference Network Design Guide". For Linux systems see the "Linux Advanced Routing & Traffic Control HOWTO" at ¡http://www.lartc.org/¿.

For more information on Quality of Service for VoIP networks see the "Enterprise QoS Solution Reference Network Design Guide" version 3.3 from Cisco at:

¡http://www.cisco.com/application/pdf/en/us/guest/netsol/ns432/c649/ccmigration_09186a00

2.1.6 MP3 Support

MP3 Music On Hold

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

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

2.1.7 ICES

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

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

2.2 Database Support

2.2.1 Realtime Database Configuration

Introduction

The Asterisk Realtime Architecture is a new set of drivers and functions implemented in Asterisk.

The benefits of this architecture are many, both from a code management standpoint and from an installation perspective.

Additional information on the configuration of Realtime with Asterisk can be found in doc/extconfig.txt

The ARA is designed to be independent of storage. Currently, most drivers are based on SQL, but the architecture should be able to handle other storage methods in the future, like LDAP.

The main benefit comes in the database support. In Asterisk v1.0 some functions supported MySQL database, some PostgreSQL and other ODBC. With the ARA, we have a unified database interface internally in Asterisk, so if one function supports database integration, all databases that has a realtime driver will be supported in that function.

Currently there are three realtime database drivers:

- ODBC: Support for UnixODBC, integrated into Asterisk The UnixODBC subsystem supports many different databases, please check www.unixodbc.org for more information.
- MySQL: Found in the asterisk-addons subversion repository on svn.digium.com
- PostgreSQL: Native support for Postgres, integrated into Asterisk

Two modes: Static and Realtime

The ARA realtime mode is used to dynamically load and update objects. This mode is used in the SIP and IAX2 channels, as well as in the voicemail system. For SIP and IAX2 this is similar to the v1.0 MYSQL_FRIENDS functionality. With the ARA, we now support many more databases for dynamic configuration of phones.

The ARA static mode is used to load configuration files. For the Asterisk modules that read configurations, there's no difference between a static file in the file system, like extensions.conf, and a configuration loaded from a database.

Realtime SIP friends

The SIP realtime objects are users and peers that are loaded in memory when needed, then deleted. This means that Asterisk currently can't handle voicemail notification and NAT keepalives for these peers. Other than that,

most of the functionality works the same way for realtime friends as for the ones in static configuration.

With caching, the device stays in memory for a specified time. More information about this is to be found in the sip.conf sample file.

Realtime H.323 friends

Like SIP realtime friends, H.323 friends also can be configured using dynamic realtime objects.

New function in the dial plan: The Realtime Switch

The realtime switch is more than a port of functionality in v1.0 to the new architecture, this is a new feature of Asterisk based on the ARA. The realtime switch lets your Asterisk server do database lookups of extensions in realtime from your dial plan. You can have many Asterisk servers sharing a dynamically updated dial plan in real time with this solution.

Note that this switch does NOT support Caller ID matching, only extension name or pattern matching.

Capabilities

The realtime Architecture lets you store all of your configuration in databases and reload it whenever you want. You can force a reload over the AMI, Asterisk Manager Interface or by calling Asterisk from a shell script with

asterisk -rx "reload"

You may also dynamically add SIP and IAX devices and extensions and making them available without a reload, by using the realtime objects and the realtime switch.

Configuration in extconfig.conf

You configure the ARA in extconfig.conf (yes, it's a strange name, but is was defined in the early days of the realtime architecture and kind of stuck). Please see doc/extconfig.txt for database schemas.

The part of Asterisk that connects to the ARA use a well defined family name to find the proper database driver. The syntax is easy:

```
<family> => <realtime driver>,<db name>[,<table>]
```

The options following the realtime driver identified depends on the driver. Defined well-known family names are:

- sippeers, sipusers - SIP peers and users
- iaxpeers, iaxusers - IAX2 peers and users
- voicemail - Voicemail accounts
- queues - Queues
- queue_members - Queue members
- extensions - Realtime extensions (switch)

There is documentation of the SQL database in the file doc/extconfig.txt in your Asterisk source code tree.

For voicemail storage with the support of ODBC, there is a doc/odbcstorage.txt documentation file.

Limitations

Currently, realtime extensions do not support realtime hints. There is a workaround available by using func_odbc. See the sample func_odbc.conf for more information.

FreeTDS supported with connection pooling

In order to use a FreeTDS-based database with realtime, you need to turn connection pooling on in res_odbc.conf. This is due to a limitation within the FreeTDS protocol itself. Please note that this includes databases such as MS SQL Server and Sybase. This support is new in the current release.

2.2.2 FreeTDS

The cdr_tds module is NOT compatible with version 0.63 of FreeTDS.

The cdr_tds module is known to work with FreeTDS version 0.62.1; it should also work with 0.62.2, 0.62.3 and 0.62.4, which are bug fix releases.

The cdr_tds module uses the raw "libtds" API of FreeTDS. It appears that from 0.63 onwards, this is not considered a published API of FreeTDS and is subject to change without notice.

Between 0.62.x and 0.63 of FreeTDS, many incompatible changes have been made to the libtds API.

For newer versions of FreeTDS, it is recommended that you use the ODBC driver.

2.3 Privacy

So, you want to avoid talking to pesky telemarketers/charity seekers/poll takers/magazine renewers/etc?

2.3.1 First of all

Try the FTC "Don't call" database, this alone will reduce your telemarketing call volume considerably. (see: https://www.donotcall.gov/default.aspx) But, this list won't protect from the Charities, previous business relationships, etc.

2.3.2 Next, Fight against autodialers!!

Zapateller detects if callerid is present, and if not, plays the da-da-da tones that immediately precede messages like, "I'm sorry, the number you have called is no longer in service."

Most humans, even those with unlisted/callerid-blocked numbers, will not immediately slam the handset down on the hook the moment they hear the three tones. But autodialers seem pretty quick to do this.

I just counted 40 hangups in Zapateller over the last year in my CDR's. So, that is possibly 40 different telemarketers/charities that have hopefully slashed my back-waters, out-of-the-way, humble home phone number from their lists.

I highly advise Zapateller for those seeking the nirvana of "privacy".

2.3.3 Next, Fight against the empty CALLERID!

A considerable percentage of the calls you don't want, come from sites that do not provide CallerID.

Null callerid's are a fact of life, and could be a friend with an unlisted number, or some charity looking for a handout. The PrivacyManager appli-

cation can help here. It will ask the caller to enter a 10-digit phone number. They get 3 tries(configurable), and this is configurable, with control being passed to priority+101 if they won't supply one.

PrivacyManager can't guarantee that the number they supply is any good, tho, as there is no way to find out, short of hanging up and calling them back. But some answers are obviously wrong. For instance, it seems a common practice for telemarketers to use your own number instead of giving you theirs. A simple test can detect this. More advanced tests would be to look for -555- numbers, numbers that count up or down, numbers of all the same digit, etc.

My logs show that 39 have hung up in the PrivacyManager script over the last year.

(Note: Demanding all unlisted incoming callers to enter their CID may not always be appropriate for all users. Another option might be to use call screening. See below.)

2.3.4 Next, use a WELCOME MENU !

Experience has shown that simply presenting incoming callers with a set of options, no matter how simple, will deter them from calling you. In the vast majority of situations, a telemarketer will simply hang up rather than make a choice and press a key.

This will also immediately foil all autodialers that simply belch a message in your ear and hang up.

Example usage of Zapateller and PrivacyManager

```
[homeline]
exten => s,1,Answer
exten => s,2,SetVar,repeatcount=0
exten => s,3,Zapateller,nocallerid
exten => s,4,PrivacyManager
exten => s,105,Background(tt-allbusy)       ;; do this if they don't enter a numbe
exten => s,106,Background(tt-somethingwrong)
exten => s,107,Background(tt-monkeysintro)
exten => s,108,Background(tt-monkeys)
exten => s,109,Background(tt-weasels)
exten => s,110,Hangup
```

```
exten => s,5,GotoIf($[ "${CALLERIDNUM}"  = "7773334444" &  "${CALLERIDNAME}" : "P
```

I suggest using Zapateller at the beginning of the context, before anything else, on incoming calls.This can be followed by the PrivacyManager App.

Make sure, if you do the PrivacyManager app, that you take care of the error condition! or their non-compliance will be rewarded with access to the system. In the above, if they can't enter a 10-digit number in 3 tries, they get the humorous "I'm sorry, but all household members are currently helping other telemarketers...", "something is terribly wrong", "monkeys have carried them away...", various loud monkey screechings, "weasels have...", and a hangup. There are plenty of other paths to my torture scripts, I wanted to have some fun.

In nearly all cases now, the telemarketers/charity-seekers that usually get thru to my main intro, hang up. I guess they can see it's pointless, or the average telemarketer/charity-seeker is instructed not to enter options when encountering such systems. Don't know.

2.3.5 Next: Torture Them!

I have developed an elaborate script to torture Telemarketers, and entertain friends. (See http://www.voip-info.org/wiki-Asterisk+Telemarketer+Torture)

While mostly those that call in and traverse my teletorture scripts are those we know, and are doing so out of curiosity, there have been these others from Jan 1st,2004 thru June 1st, 2004: (the numbers may or may not be correct.)

603890zzzz hung up telemarket options. "Integrated Sale" called a couple times. hung up in telemarket options "UNITED STATES GOV" (– maybe a military recruiter, trying to lure one of my sons). 800349zzzz – hung up in charity intro 800349zzzz – hung up in charity choices, intro, about the only one who actually travelled to the bitter bottom of the scripts! 216377zzzz – hung up the magazine section 626757zzzz = "LIR " (pronounced "Liar"?) hung up in telemarket intro, then choices 757821zzzz – hung up in new magazine subscription options.

That averages out to maybe 1 a month. That puts into question whether the ratio of the amount of labor it took to make the scripts versus the benefits of lower call volumes was worth it, but, well, I had fun, so what the heck.

but, that's about it. Not a whole lot. But I haven't had to say "NO" or "GO AWAY" to any of these folks for about a year now ...!

2.3.6 Using Call Screening

Another option is to use call screening in the Dial command. It has two main privacy modes, one that remembers the CID of the caller, and how the callee wants the call handled, and the other, which does not have a "memory".

Turning on these modes in the dial command results in this sequence of events, when someone calls you at an extension:

1. The caller calls the Asterisk system, and at some point, selects an option or enters an extension number that would dial your extension.
2. Before ringing your extension, the caller is asked to supply an introduction. The application asks them: "After the tone, say your name". They are allowed 4 seconds of introduction.
3. After that, they are told "Hang on, we will attempt to connect you to your party. Depending on your dial options, they will hear ringing indications, or get music on hold. I suggest music on hold.
4. Your extension is then dialed. When (and if) you pick up, you are told that a caller presenting themselves as ¡their recorded intro is played¿ is calling, and you have options, like being connected, sending them to voicemail, torture, etc.
5. You make your selection, and the call is handled as you chose.

There are some variations, and these will be explained in due course.
To use these options, set your Dial to something like:

```
exten => 3,3,Dial(Zap/5r3&Zap/6r3|35|tmPA(beep))

or

exten => 3,3,Dial(Zap/5r3&Zap/6r3|35|tmP(something)A(beep))

or

exten => 3,3,Dial(Zap/5r3&Zap/6r3|35|tmpA(beep))
```

The 't' allows the dialed party to transfer the call using '#'. It's optional.

The 'm' is for music on hold. I suggest it. Otherwise, the calling party gets to hear all the ringing, and lack thereof. It is generally better to use Music On Hold. Lots of folks hang up after the 3rd or 4th ring, and you might lose the call before you can enter an option!

The 'P' option alone will database everything using the extension as a default 'tree'. To get multiple extensions sharing the same database, use P(some-shared-key). Also, if the same person has multiple extensions, use P(unique-id) on all their dial commands.

Use little 'p' for screening. Every incoming call will include a prompt for the callee's choice.

the A(beep), will generate a 'beep' that the callee will hear if they choose to talk to the caller. It's kind of a prompt to let the callee know that he has to say 'hi'. It's not required, but I find it helpful.

When there is no CallerID, P and p options will always record an intro for the incoming caller. This intro will be stored temporarily in the /var/lib/asterisk/sounds/priv-callerintros dir, under the name NOCALLERID_¡extension¿¡channelı and will be erased after the callee decides what to do with the call.

Of course, NOCALLERID is not stored in the database. All those with no CALLERID will be considered "Unknown".

2.3.7 The 'N' and 'n' options

Two other options exist, that act as modifiers to the privacy options 'P' and 'p'. They are 'N' and 'n'. You can enter them as dialing options, but they only affect things if P or p are also in the options.

'N' says, "Only screen the call if no CallerID is present". So, if a callerID were supplied, it will come straight thru to your extension.

'n' says, "Don't save any introductions". Folks will be asked to supply an introduction ("At the tone, say your name") every time they call. Their introductions will be removed after the callee makes a choice on how to handle the call. Whether the P option or the p option is used, the incoming caller will have to supply their intro every time they call.

2.3.8 Recorded Introductions

Philosophical Side Note

The 'P' option stores the CALLERID in the database, along with the callee's choice of actions, as a convenience to the CALLEE, whereas introductions are stored and re-used for the convenience of the CALLER.]

Introductions

Unless instructed to not save introductions (see the 'n' option above), the screening modes will save the recordings of the caller's names in the directory /var/lib/asterisk/sounds/priv-callerintros, if they have a CallerID. Just the 10-digit callerid numbers are used as filenames, with a ".gsm" at the end.

Having these recordings around can be very useful, however...

First of all, if a callerid is supplied, and a recorded intro for that number is already present, the caller is spared the inconvenience of having to supply their name, which shortens their call a bit.

Next of all, these intros can be used in voicemail, played over loudspeakers, and perhaps other nifty things. For instance:

```
exten => s,7,System(/usr/bin/play /var/lib/asterisk/sounds/priv-callerintros/${CAI
```

When a call comes in at the house, the above priority gets executed, and the callers intro is played over the phone systems speakers. This gives us a hint who is calling.

(Note: the —0 option at the end of the System command above, is a local mod I made to the System command. It forces a 0 result code to be returned, whether the play command successfully completed or not. Therefore, I don't have to ensure that the file exists or not. While I've turned this mod into the developers, it hasn't been incorporated yet. You might want to write an AGI or shell script to handle it a little more intelligently)

And one other thing. You can easily supply your callers with an option to listen to, and re-record their introductions. Here's what I did in the home system's extensions.conf. (assume that a Goto(home-introduction—s—1) exists somewhere in your main menu as an option):

```
[home-introduction]
exten => s,1,Background,intro-options   ;; Script: To hear your Introduction, dial
```

```
                                        ;;         to record a new introduction, c
                                        ;;         to return to the main menu, dia
                                        ;;         to hear what this is all about,
exten => 1,1,Playback,priv-callerintros/${CALLERIDNUM}
exten => 1,2,Goto(s,1)
exten => 2,1,Goto(home-introduction-record,s,1)
exten => 3,1,Goto(homeline,s,7)
exten => 4,1,Playback,intro-intro     ;; Script:
                                ;; This may seem a little strange, but it really i
                                ;; thing, both for you and for us. I've taped a sh
                                ;; for many of the folks who normally call us. Usi
                                ;; from each incoming call, the system plays the i
                                ;; for that phone number over a speaker, just as t
                                ;; This helps the folks
                                ;; here in the house more quickly determine who is
                                ;; and gets the right ones to gravitate to the pho
                                ;; You can listen to, and record a new intro for y
                                ;; using this menu.
exten => 4,2,Goto(s,1)
exten => t,1,Goto(s,1)
exten => i,1,Background,invalid
exten => i,2,Goto(s,1)
exten => o,1,Goto(s,1)

[home-introduction-record]
exten => s,1,Background,intro-record-choices    ;; Script:
                                ;;      If you want some advice about recording yo
                                ;;      introduction, dial 1.
                                ;;      otherwise, dial 2, and introduce yourself
                                ;;      the beep.
exten => 1,1,Playback,intro-record
                                ;;      Your introduction should be short and swee
                                ;;      Your introduction will be limited to 4 sec
                                ;;      This is NOT meant to be a voice mail messa
                                ;;      please, don't say anything about why you a
                                ;;      After we are done making the recording, yo
                                ;;      will be saved for playback.
                                ;;      If you are the only person that would call
```

```
                                    ;;      please state your name.  Otherwise, state
                                    ;;      or residence name instead. For instance, i
                                    ;;      friend of the family, say, Olie McPherson,
                                    ;;      you and your kids might call here a lot, y
                                    ;;      say: "This is the distinguished Olie McPhe
                                    ;;      If you are the only person calling, you mi
                                    ;;      "This is the illustrious Kermit McFrog! Pi
                                    ;;      If you are calling from a business, you mi
                                    ;;      "Fritz from McDonalds calling.", or perhap
                                    ;;      "John, from the Park County Morgue. You st
                                    ;;      Just one caution: the kids will hear what
                                    ;;      you call. So watch your language!
                                    ;;      I will begin recording after the tone.
                                    ;;      When you are done, hit the # key. Gather y
                                    ;;      ready. Remember, the # key will end the re
                                    ;;      your intro. Good Luck, and Thank you!"
exten => 1,2,Goto(2,1)
exten => 2,1,Background,intro-start
                                    ;;  OK, here we go! After the beep, please give yo
exten => 2,2,Background,beep
exten => 2,3,Record,priv-callerintros/${CALLERIDNUM}:gsm|4
exten => 2,4,Background,priv-callerintros/${CALLERIDNUM}
exten => 2,5,Goto(home-introduction,s,1)
exten => t,1,Goto(s,1)
exten => i,1,Background,invalid
exten => i,2,Goto(s,1)
exten => o,1,Goto(s,1)
```

In the above, you'd most likely reword the messages to your liking, and maybe do more advanced things with the 'error' conditions (i,o,t priorities), but I hope it conveys the idea.

Chapter 3

Channel Variables

3.1 Introduction

There are two levels of parameter evaluation done in the Asterisk dial plan in extensions.conf.

1. The first, and most frequently used, is the substitution of variable references with their values.

2. Then there are the evaluations of expressions done in $[..]. This will be discussed below.

Asterisk has user-defined variables and standard variables set by various modules in Asterisk. These standard variables are listed at the end of this document.

3.2 Parameter Quoting

```
exten => s,5,BackGround,blabla
```

The parameter (blabla) can be quoted ("blabla"). In this case, a comma does not terminate the field. However, the double quotes will be passed down to the Background command, in this example.

Also, characters special to variable substitution, expression evaluation, etc (see below), can be quoted. For example, to literally use a $ on the string "$1231", quote it with a preceding

. Special characters that must be quoted to be used, are [ ] $ ”
. (to write
itself, use
).

These Double quotes and escapes are evaluated at the level of the asterisk config file parser.

Double quotes can also be used inside expressions, as discussed below.

3.3 Variables

Parameter strings can include variables. Variable names are arbitrary strings. They are stored in the respective channel structure.

To set a variable to a particular value, do :

```
exten => 1,2,Set(varname=value)
```

You can substitute the value of a variable everywhere using $variablename. For example, to stringwise append $lala to $blabla and store result in $koko, do:

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

There are two reference modes - reference by value and reference by name. To refer to a variable with its name (as an argument to a function that requires a variable), just write the name. To refer to the variable's value, enclose it inside $. For example, Set takes as the first argument (before the =) a variable name, so:

```
exten => 1,2,Set(koko=lala)
exten => 1,3,Set(${koko}=blabla)
```

stores to the variable ”koko” the value ”lala” and to variable ”lala” the value ”blabla”.

In fact, everything contained $here is just replaced with the value of the variable ”here”.

3.4 Variable Inheritance

Variable names which are prefixed by "_" will be inherited to channels that are created in the process of servicing the original channel in which the variable was set. When the inheritance takes place, the prefix will be removed in the channel inheriting the variable. If the name is prefixed by "__" in the channel, then the variable is inherited and the "__" will remain intact in the new channel.

In the dialplan, all references to these variables refer to the same variable, regardless of having a prefix or not. Note that setting any version of the variable removes any other version of the variable, regardless of prefix.

3.4.1 Example

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

However,

NoOp(${__FOO}) is identical to NoOp(${FOO})
```

3.5 Selecting Characters from Variables

The format for selecting characters from a variable can be expressed as:

```
${variable_name[:offset[:length]]}
```

If you want to select the first N characters from the string assigned to a variable, simply append a colon and the number of characters to skip from the beginning of the string to the variable name.

```
;Remove the first character of extension, save in "number" variable
exten => _9X.,1,Set(number=${EXTEN:1})
```

Assuming we've dialed 918005551234, the value saved to the 'number' variable would be 18005551234. This is useful in situations when we require users to dial a number to access an outside line, but do not wish to pass the first digit.

If you use a negative offset number, Asterisk starts counting from the end of the string and then selects everything after the new position. The following example will save the numbers 1234 to the 'number' variable, still assuming we've dialed 918005551234.

```
;Remove everything before the last four digits of the dialed string
exten => _9X.,1,Set(number=${EXTEN:-4})
```

We can also limit the number of characters from our offset position that we wish to use. This is done by appending a second colon and length value to the variable name. The following example will save the numbers 555 to the 'number' variable.

```
;Only save the middle numbers 555 from the string 918005551234
exten => _9X.,1,Set(number=${EXTEN:5:3})
```

The length value can also be used in conjunction with a negative offset. This may be useful if the length of the string is unknown, but the trailing digits are. The following example will save the numbers 555 to the 'number' variable, even if the string starts with more characters than expected (unlike the previous example).

```
;Save the numbers 555 to the 'number' variable
exten => _9X.,1,Set(number=${EXTEN:-7:3})
```

If a negative length value is entered, Asterisk will remove that many characters from the end of the string.

```
;Set pin to everything but the trailing #.
exten => _XXXX#,1,Set(pin=${EXTEN:0:-1})
```

3.6 Expressions

Everything contained inside a bracket pair prefixed by a $ (like $[this]) is considered as an expression and it is evaluated. Evaluation works similar to (but is done on a later stage than) variable substitution: the expression (including the square brackets) is replaced by the result of the expression evaluation.

For example, after the sequence:

```
exten => 1,1,Set(lala=$[1 + 2])
exten => 1,2,Set(koko=$[2 * ${lala}])
```

the value of variable koko is "6".

and, further:

```
exten => 1,1,Set,(lala=$[  1 +    2   ]);
```

will parse as intended. Extra spaces are ignored.

3.6.1 Spaces Inside Variables Values

If the variable being evaluated contains spaces, there can be problems.

For these cases, double quotes around text that may contain spaces will force the surrounded text to be evaluated as a single token. The double quotes will be counted as part of that lexical token.

As an example:

```
exten => s,6,GotoIf($[ "${CALLERIDNAME}" : "Privacy Manager" ]?callerid-liar|s|1:s
```

The variable CALLERIDNAME could evaluate to "DELOREAN MOTORS" (with a space) but the above will evaluate to:

```
"DELOREAN MOTORS" : "Privacy Manager"
```

and will evaluate to 0.

The above without double quotes would have evaluated to:

```
DELOREAN MOTORS : Privacy Manager
```

and will result in syntax errors, because token DELOREAN is immediately followed by token MOTORS and the expression parser will not know how to evaluate this expression, because it does not match its grammar.

3.6.2 Operators

Operators are listed below in order of increasing precedence. Operators with equal precedence are grouped within symbols.

```
expr1 | expr2
        Return the evaluation of expr1 if it is neither an empty string
        nor zero; otherwise, returns the evaluation of expr2.

expr1 & expr2
        Return the evaluation of expr1 if neither expression evaluates to
        an empty string or zero; otherwise, returns zero.

expr1 {=, >, >=, <, <=, !=} expr2
        Return the results of integer comparison if both arguments are
        integers; otherwise, returns the results of string comparison
        using the locale-specific collation sequence.  The result of each
        comparison is 1 if the specified relation is true, or 0 if the
        relation is false.

expr1 {+, -} expr2
        Return the results of addition or subtraction of integer-valued
        arguments.

expr1 {*, /, %} expr2
        Return the results of multiplication, integer division, or
        remainder of integer-valued arguments.

- expr1
       Return the result of subtracting expr1 from 0.
       This, the unary minus operator, is right associative, and
       has the same precedence as the ! operator.

! expr1
       Return the result of a logical complement of expr1.
       In other words, if expr1 is null, 0, an empty string,
       or the string "0", return a 1. Otherwise, return a 0.
       It has the same precedence as the unary minus operator, and
       is also right associative.
```

```
expr1 : expr2
        The ‘:’ operator matches expr1 against expr2, which must be a
        regular expression.  The regular expression is anchored to the
        beginning of  the string with an implicit ‘^’.

        If the match succeeds and the pattern contains at least one regu-
        lar expression subexpression ‘\(...\)’, the string correspond-
        ing to ‘\1’ is returned; otherwise the matching operator
        returns the number of characters matched.  If the match fails and
        the pattern contains a regular expression subexpression the null
        string is returned; otherwise 0.

        Normally, the double quotes wrapping a string are left as part
        of the string. This is disastrous to the : operator. Therefore,
        before the regex match is made, beginning and ending double quote
        characters are stripped from both the pattern and the string.

 expr1 =~ expr2
        Exactly the same as the ’:’ operator, except that the match is
        not anchored to the beginning of the string. Pardon any similarity
        to seemingly similar operators in other programming languages!
        The ":" and "=~" operators share the same precedence.

 expr1 ? expr2 :: expr3
        Traditional Conditional operator. If expr1 is a number
that evaluates to 0 (false), expr3 is result of the this
expression evaluation.  Otherwise, expr2 is the result.
If expr1 is a string, and evaluates to an empty string,
or the two characters (""), then expr3 is the
result. Otherwise, expr2 is the result.  In Asterisk, all
3 exprs will be "evaluated"; if expr1 is "true", expr2
will be the result of the "evaluation" of this
expression.  expr3 will be the result otherwise. This
operator has the lowest precedence.
```

Parentheses are used for grouping in the usual manner.

Operator precedence is applied as one would expect in any of the C or C derived languages.

3.6.3 Examples

```
 "One Thousand Five Hundred" =~ "(T[^ ]+)"
returns: Thousand

 "One Thousand Five Hundred" =~ "T[^ ]+"
returns: 8

 "One Thousand Five Hundred" : "T[^ ]+"
returns: 0

 "8015551212" : "(...)"
returns: 801

 "3075551212":"...(...)"
returns: 555

 ! "One Thousand Five Hundred" =~ "T[^ ]+"
returns: 0 (because it applies to the string, which is non-null,
                   which it turns to "0", and then looks for the pattern
                   in the "0", and doesn't find it)

 !( "One Thousand Five Hundred" : "T[^ ]+" )
returns: 1  (because the string doesn't start with a word starting
                    with T, so the match evals to 0, and the ! operator
                    inverts it to 1 ).

 2 + 8 / 2
returns 6. (because of operator precedence; the division is done first, then the a

 2+8/2
returns 6. Spaces aren't necessary.

(2+8)/2
returns 5, of course.
```

```
\begin{verbatim}

Of course, all of the above examples use constants, but would work the
same if any of the numeric or string constants were replaced with a
variable reference \${CALLERIDNUM}, for instance.


\subsection{Numbers Vs. Strings}

Tokens consisting only of numbers are converted to 64-bit numbers for
most of the operators. This means that overflows can occur when the
numbers get above 18 digits.  Warnings will appear in the logs in this
case.

\subsection{Conditionals}

There is one conditional application - the conditional goto :

\begin{verbatim}
exten => 1,2,gotoif(condition?label1:label2)
```

If condition is true go to label1, else go to label2. Labels are interpreted exactly as in the normal goto command.

"condition" is just a string. If the string is empty or "0", the condition is considered to be false, if it's anything else, the condition is true. This is designed to be used together with the expression syntax described above, eg :

```
exten => 1,2,gotoif($[${CALLERID} = 123456]?2|1:3|1)
```

Example of use :

```
exten => s,2,Set(vara=1)
exten => s,3,Set(varb=$[${vara} + 2])
exten => s,4,Set(varc=$[${varb} * 2])
exten => s,5,GotoIf($[${varc} = 6]?99|1:s|6)
```

3.6.4 Parse Errors

Syntax errors are now output with 3 lines.

If the extensions.conf file contains a line like:

```
exten => s,6,GotoIf($[ "${CALLERIDNUM}"  = "3071234567" & &  "${CALLERIDNAME}" : '
```

You may see an error in /var/log/asterisk/messages like this:

```
Jul 15 21:27:49 WARNING[1251240752]: ast_yyerror(): syntax error: parse error, une
"3072312154"  = "3071234567" & & "Steves Extension" : "Privacy Manager"
                                ^
```

The log line tells you that a syntax error was encountered. It now also tells you (in grand standard bison format) that it hit an "AND" (&) token unexpectedly, and that was hoping for for a MINUS (-), LP (left parenthesis), or a plain token (a string or number).

The next line shows the evaluated expression, and the line after that, the position of the parser in the expression when it became confused, marked with the "$\hat{}$" character.

3.6.5 NULL Strings

Testing to see if a string is null can be done in one of two different ways:

```
exten => _XX.,1,GotoIf($["${calledid}" != ""]?3)

exten => _XX.,1,GotoIf($[foo${calledid} != foo]?3)
```

The second example above is the way suggested by the WIKI. It will work as long as there are no spaces in the evaluated value.

The first way should work in all cases, and indeed, might now be the safest way to handle this situation.

3.6.6 Warning

If you need to do complicated things with strings, asterisk expressions is most likely NOT the best way to go about it. AGI scripts are an excellent option to this need, and make available the full power of whatever language you desire, be it Perl, C, C++, Cobol, RPG, Java, Snobol, PL/I, Scheme, Common Lisp, Shell scripts, Tcl, Forth, Modula, Pascal, APL, assembler, etc.

3.6.7 Incompatabilities

The asterisk expression parser has undergone some evolution. It is hoped that the changes will be viewed as positive.

The "original" expression parser had a simple, hand-written scanner, and a simple bison grammar. This was upgraded to a more involved bison grammar, and a hand-written scanner upgraded to allow extra spaces, and to generate better error diagnostics. This upgrade required bison 1.85, and part of the user community felt the pain of having to upgrade their bison version.

The next upgrade included new bison and flex input files, and the makefile was upgraded to detect current version of both flex and bison, conditionally compiling and linking the new files if the versions of flex and bison would allow it.

If you have not touched your extensions.conf files in a year or so, the above upgrades may cause you some heartburn in certain circumstances, as several changes have been made, and these will affect asterisk's behavior on legacy extension.conf constructs. The changes have been engineered to minimize these conflicts, but there are bound to be problems.

The following list gives some (and most likely, not all) of areas of possible concern with "legacy" extension.conf files:

1. Tokens separated by space(s). Previously, tokens were separated by spaces. Thus, ' 1 + 1 ' would evaluate to the value '2', but '1+1' would evaluate to the string '1+1'. If this behavior was depended on, then the expression evaluation will break. '1+1' will now evaluate to '2', and something is not going to work right. To keep such strings from being evaluated, simply wrap them in double quotes: ' "1+1" '

2. The colon operator. In versions previous to double quoting, the colon operator takes the right hand string, and using it as a regex pattern, looks for it in the left hand string. It is given an implicit ôperator at the beginning, meaning the pattern will match only at the beginning of the left hand string. If the pattern or the matching string had double quotes around them, these could get in the way of the pattern match. Now, the wrapping double quotes are stripped from both the pattern and the left hand string before applying the pattern. This was done because it recognized that the new way of scanning the expression doesn't use spaces to separate tokens, and the average regex expression is full of

operators that the scanner will recognize as expression operators. Thus, unless the pattern is wrapped in double quotes, there will be trouble. For instance, $VAR1 : (Who—What*)+ may have have worked before, but unless you wrap the pattern in double quotes now, look out for trouble! This is better: "$VAR1" : "(Who—What*)+" and should work as previous.

3. Variables and Double Quotes Before these changes, if a variable's value contained one or more double quotes, it was no reason for concern. It is now!

4. LE, GE, NE operators removed. The code supported these operators, but they were not documented. The symbolic operators, ¡=, ¿=, and != should be used instead.

5. Added the unary '-' operator. So you can 3+ -4 and get -1.

6. Added the unary '!' operator, which is a logical complement. Basically, if the string or number is null, empty, or '0', a '1' is returned. Otherwise a '0' is returned.

7. Added the '= ' operator, just in case someone is just looking for match anywhere in the string. The only diff with the ':' is that match doesn't have to be anchored to the beginning of the string.

8. Added the conditional operator 'expr1 ? true_expr :: false_expr' First, all 3 exprs are evaluated, and if expr1 is false, the 'false_expr' is returned as the result. See above for details.

9. Unary operators '-' and '!' were made right associative.

3.6.8 Debugging Hints

There are two utilities you can build to help debug the $[ ] in your extensions.conf file.

The first, and most simplistic, is to issue the command:

make testexpr2

in the top level asterisk source directory. This will build a small executable, that is able to take the first command line argument, and run it

thru the expression parser. No variable substitutions will be performed. It might be safest to wrap the expression in single quotes...

testexpr2 '2*2+2/2'

is an example.

And, in the utils directory, you can say:

make check_expr

and a small program will be built, that will check the file mentioned in the first command line argument, for any expressions that might be have problems when you move to flex-2.5.31. It was originally designed to help spot possible incompatibilities when moving from the pre-2.5.31 world to the upgraded version of the lexer.

But one more capability has been added to check_expr, that might make it more generally useful. It now does a simple minded evaluation of all variables, and then passes the $[] exprs to the parser. If there are any parse errors, they will be reported in the log file. You can use check_expr to do a quick sanity check of the expressions in your extensions.conf file, to see if they pass a crude syntax check.

The "simple-minded" variable substitution replaces $varname variable references with '555'. You can override the 555 for variable values, by entering in var=val arguments after the filename on the command line. So...

check_expr /etc/asterisk/extensions.conf CALLERIDNUM=3075551212 DIALSTATUS=TORTURE EXTEN=121

will substitute any $CALLERIDNUM variable references with 3075551212, any $DIALSTATUS variable references with 'TORTURE', and any $EXTEN references with '121'. If there is any fancy stuff going on in the reference, like $EXTEN:2, then the override will not work. Everything in the $... has to match. So, to substitute $EXTEN:2 references, you'd best say:

check_expr /etc/asterisk/extensions.conf CALLERIDNUM=3075551212 DIALSTATUS=TORTURE EXTEN:2=121

on stdout, you will see something like:

OK – $["$DIALSTATUS" = "TORTURE" — "$DIALSTATUS" = "DONT-CALL"] at line 416

In the expr2_log file that is generated, you will see:

line 416, evaluation of $["TORTURE" = "TORTURE" — "TORTURE" = "DONTCALL"] result: 1

check_expr is a very simplistic algorithm, and it is far from being guaranteed to work in all cases, but it is hoped that it will be useful.

3.7 Asterisk standard channel variables

There are a number of variables that are defined or read by Asterisk. Here is a list of them. More information is available in each application's help text. All these variables are in UPPER CASE only.

Variables marked with a * are builtin functions and can't be set, only read in the dialplan. Writes to such variables are silently ignored.

```
${ACCOUNTCODE}    * Account code (if specified) (Deprecated; use ${CDR(accountcode
${BLINDTRANSFER}  The name of the channel on the other side of a blind transfer
${BRIDGEPEER}   Bridged peer
${CALLERANI}   * Caller ANI (PRI channels) (Deprecated; use ${CALLERID(ani)})
${CALLERID}   * Caller ID (Deprecated; use ${CALLERID(all)})
${CALLERIDNAME}   * Caller ID Name only (Deprecated; use ${CALLERID(name)})
${CALLERIDNUM}   * Caller ID Number only (Deprecated; use ${CALLERID(num)})
${CALLINGANI2}   * Caller ANI2 (PRI channels)
${CALLINGPRES}   * Caller ID presentation for incoming calls (PRI channels)
${CALLINGTNS}    * Transit Network Selector (PRI channels)
${CALLINGTON}     * Caller Type of Number (PRI channels)
${CHANNEL}   * Current channel name
${CONTEXT}        * Current context
${DATETIME}   * Current date time in the format: DDMMYYYY-HH:MM:SS (Deprecated; us
${DB_RESULT} Result value of DB_EXISTS() dial plan function
${DNID}           * Dialed Number Identifier (Deprecated; use ${CALLERID(dnid)})
${EPOCH}   * Current unix style epoch
${EXTEN}   * Current extension
${ENV(VAR)}   Environmental variable VAR
${GOTO_ON_BLINDXFR} Transfer to the specified context/extension/priority
after a blind transfer (use ^ characters in place of
| to separate context/extension/priority when setting
this variable from the dialplan)
${HANGUPCAUSE}   * Asterisk cause of hangup (inbound/outbound)
${HINT}           * Channel hints for this extension
${HINTNAME}       * Suggested Caller*ID name for this extension
${INVALID_EXTEN}  The invalid called extension (used in the "i" extension)
${LANGUAGE}   * Current language (Deprecated; use ${LANGUAGE()})
${LEN(VAR)}   * String length of VAR (integer)
${PRIORITY}   * Current priority in the dialplan
```

```
${PRIREDIRECTREASON}  Reason for redirect on PRI, if a call was directed
${RDNIS}          * Redirected Dial Number ID Service (Deprecated; use ${CALLERID(
${TIMESTAMP}   * Current date time in the format: YYYYMMDD-HHMMSS (Deprecated; use
${TRANSFER_CONTEXT}  Context for transferred calls
${FORWARD_CONTEXT}     Context for forwarded calls
${UNIQUEID}   * Current call unique identifier
${SYSTEMNAME} * value of the systemname option of asterisk.conf
```

3.7.1 Application return values

———————- In Asterisk 1.2, many applications return the result in a variable instead of, as in Asterisk 1.0, changing the dial plan priority (+101). For the various status values, see each application's help text.

```
${AGISTATUS} * agi()
${AQMSTATUS} * addqueuemember()
${AVAILSTATUS} * chanisavail()
${CHECKGROUPSTATUS} * checkgroup()
${CHECKMD5STATUS} * checkmd5()
${CPLAYBACKSTATUS} * controlplayback()
${DIALSTATUS}    * dial()
${DBGETSTATUS} * dbget()
${ENUMSTATUS} * enumlookup()
${HASVMSTATUS} * hasnewvoicemail()
${LOOKUPBLSTATUS} * lookupblacklist()
${OSPAUTHSTATUS} * ospauth()
${OSPLOOKUPSTATUS} * osplookup()
${OSPNEXTSTATUS} * ospnext()
${OSPFINISHSTATUS} * ospfinish()
${PARKEDAT} * parkandannounce()
${PLAYBACKSTATUS} * playback()
${PQMSTATUS} * pausequeuemember()
${PRIVACYMGRSTATUS} * privacymanager()
${QUEUESTATUS} * queue()
${RQMSTATUS} * removequeuemember()
${SENDIMAGESTATUS} * sendimage()
${SENDTEXTSTATUS} * sendtext()
${SENDURLSTATUS} * sendurl()
```

```
${SYSTEMSTATUS} * system()
${TRANSFERSTATUS} * transfer()
${TXTCIDNAMESTATUS} * txtcidname()
${UPQMSTATUS} * unpausequeuemember()
${VMSTATUS} * voicmail()
${VMBOXEXISTSSTATUS} * vmboxexists()
${WAITSTATUS} * waitforsilence()
```

3.7.2 Various application variables

```
${CURL} * Resulting page content for curl()
${ENUM} * Result of application EnumLookup
${EXITCONTEXT} Context to exit to in IVR menu (app background())
or in the RetryDial() application
${MONITOR} * Set to "TRUE" if the channel is/has been monitored (app monitor())
${MONITOR_EXEC} Application to execute after monitoring a call
${MONITOR_EXEC_ARGS} Arguments to application
${MONITOR_FILENAME}  File for monitoring (recording) calls in queue
${QUEUE_PRIO} Queue priority
${QUEUE_MAX_PENALTY} Maximum member penalty allowed to answer caller
${QUEUESTATUS}  Status of the call, one of:
(TIMEOUT | FULL | JOINEMPTY | LEAVEEMPTY | JOINUNAVAIL | LEAVEUNAVAIL)
${RECORDED_FILE}  * Recorded file in record()
${TALK_DETECTED}  * Result from talkdetect()
${TOUCH_MONITOR}  The filename base to use with Touch Monitor (auto record)
${TOUCH_MONITOR_FORMAT} The audio format to use with Touch Monitor (auto record)
${TOUCH_MONITOR_OUTPUT} * Recorded file from Touch Monitor (auto record)
${TXTCIDNAME} * Result of application TXTCIDName
${VPB_GETDTMF} chan_vpb
```

3.7.3 The MeetMe Conference Bridge

```
${MEETME_RECORDINGFILE}  Name of file for recording a conference with
the "r" option
${MEETME_RECORDINGFORMAT}  Format of file to be recorded
${MEETME_EXIT_CONTEXT}  Context for exit out of meetme meeting
${MEETME_AGI_BACKGROUND}  AGI script for Meetme (zap only)
${MEETMESECS}  * Number of seconds a user participated in a MeetMe conference
```

3.7.4 The VoiceMail() application

```
${VM_CATEGORY} Sets voicemail category
${VM_NAME} * Full name in voicemail
${VM_DUR} * Voicemail duration
${VM_MSGNUM} * Number of voicemail message in mailbox
${VM_CALLERID} * Voicemail Caller ID (Person leaving vm)
${VM_CIDNAME} * Voicemail Caller ID Name
${VM_CIDNUM} * Voicemail Caller ID Number
${VM_DATE} * Voicemail Date
${VM_MESSAGEFILE} * Path to message left by caller
```

3.7.5 The VMAuthenticate() application

```
${AUTH_MAILBOX} * Authenticated mailbox
${AUTH_CONTEXT} * Authenticated mailbox context
```

3.7.6 DUNDiLookup()

```
${DUNDTECH} * The Technology of the result from a call to DUNDiLookup()
${DUNDDEST} * The Destination of the result from a call to DUNDiLookup()
```

3.7.7 chan_zap

```
${ANI2} * The ANI2 Code provided by the network on the incoming call.
(ie, Code 29 identifies call as a Prison/Inmate Call)
${CALLTYPE} * Type of call (Speech, Digital, etc)
${CALLEDTON} * Type of number for incoming PRI extension
i.e. 0=unknown, 1=international, 2=domestic, 3=net_specific,
4=subscriber, 6=abbreviated, 7=reserved
${CALLINGSUBADDR} * Called PRI Subaddress
${FAXEXTEN}   * The extension called before being redirected to "fax"
${PRIREDIRECTREASON} * Reason for redirect, if a call was directed
${SMDI_VM_TYPE} * When an call is received with an SMDI message, the 'type'
of message 'b' or 'u'
```

3.7.8 chan_sip

```
${SIPCALLID}  * SIP Call-ID: header verbatim (for logging or CDR matching)
```

```
${SIPDOMAIN}     * SIP destination domain of an inbound call (if appropriate)
${SIPUSERAGENT}  * SIP user agent (deprecated)
${SIPURI} * SIP uri
${SIP_CODEC}  Set the SIP codec for a call
${SIP_URI_OPTIONS} * additional options to add to the URI for an outgoing call
${RTPAUDIOQOS} RTCP QoS report for the audio of this call
${RTPVIDEOQOS} RTCP QoS report for the video of this call
```

3.7.9 chan_agent

```
${AGENTMAXLOGINTRIES} Set the maximum number of failed logins
${AGENTUPDATECDR} Whether to update the CDR record with Agent channel data
${AGENTGOODBYE} Sound file to use for "Good Bye" when agent logs out
${AGENTACKCALL}  Whether the agent should acknowledge the incoming call
${AGENTAUTOLOGOFF} Auto logging off for an agent
${AGENTWRAPUPTIME} Setting the time for wrapup between incoming calls
${AGENTNUMBER} * Agent number (username) set at login
${AGENTSTATUS} * Status of login ( fail | on | off )
${AGENTEXTEN} * Extension for logged in agent
```

3.7.10 The Dial() application

```
${DIALEDPEERNAME}  * Dialed peer name
${DIALEDPEERNUMBER}  * Dialed peer number
${DIALEDTIME}    * Time for the call (seconds)
${ANSWEREDTIME}  * Time from dial to answer (seconds)
${DIALSTATUS}    * Status of the call, one of:
                 (CHANUNAVAIL | CONGESTION | BUSY | NOANSWER
| ANSWER | CANCEL | DONTCALL | TORTURE)
${DYNAMIC_FEATURES} * The list of features (from the [applicationmap] section of
  features.conf) to activate during the call, with feature
  names separated by '#' characters
${LIMIT_PLAYAUDIO_CALLER} Soundfile for call limits
${LIMIT_PLAYAUDIO_CALLEE} Soundfile for call limits
${LIMIT_WARNING_FILE} Soundfile for call limits
${LIMIT_TIMEOUT_FILE} Soundfile for call limits
${LIMIT_CONNECT_FILE} Soundfile for call limits
${OUTBOUND_GROUP} Default groups for peer channels (as in SetGroup)
```

```
* See "show application dial" for more information
```

3.7.11 The chanisavail() application

```
${AVAILCHAN} * the name of the available channel if one was found
${AVAILORIGCHAN}  * the canonical channel name that was used to create the channel
${AVAILSTATUS} * Status of requested channel
```

3.7.12 Dialplan Macros

```
${MACRO_EXTEN} * The calling extensions
${MACRO_CONTEXT} * The calling context
${MACRO_PRIORITY} * The calling priority
${MACRO_OFFSET} Offset to add to priority at return from macro
```

3.7.13 The ChanSpy() application

```
${SPYGROUP} * A ':' (colon) separated list of group names.
  (To be set on spied on channel and matched against the g(grp) option)
```

3.7.14 OSP

```
${OSPINHANDLE} OSP handle of in_bound call
${OSPINTIMELIMIT} Duration limit for in_bound call
${OSPOUTHANDLE} OSP handle of out_bound call
${OSPTECH} OSP technology
${OSPDEST} OSP destination
${OSPCALLING} OSP calling number
${OSPOUTTOKEN} OSP token to use for out_bound call
${OSPOUTTIMELIMIT} Duration limit for out_bound call
${OSPRESULTS} Number of remained destinations
```

Chapter 4

AEL, Asterisk Extension Language

4.1 Introduction

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

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

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

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

- The first and most obvious is the AEL syntax itself. A BNF is provided near the end of this document.
- The second syntax is the Expression Syntax, which is normally handled by Asterisk extension engine, as expressions enclosed in $[...]. The right hand side of assignments are wrapped in $[...] by AEL, and so are the if and while expressions, among others.
- The third syntax is the Variable Reference Syntax, the stuff enclosed in $.. curly braces. It's a bit more involved than just putting a variable name in there. You can include one of dozens of 'functions', and their arguments, and there are even some string manipulation notation in there.

- The last syntax that underlies AEL, and is not used directly in AEL, is the Extension Language Syntax. The extension language is what you see in extensions.conf, and AEL compiles the higher level AEL language into extensions and priorities, and passes them via function calls into Asterisk. Embedded in this language is the Application/AGI commands, of which one application call per step, or priority can be made. You can think of this as a "macro assembler" language, that AEL will compile into.

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

4.2 Asterisk in a Nutshell

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

4.2.1 Contexts

Contexts are a grouping of extensions.

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

4.2.2 Extensions and priorities

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

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

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

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

4.2.3 Macros

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

4.2.4 Applications

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

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

4.3 Getting Started

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

Reloading extensions.ael

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

*CLI¿ ael reload

4.4 Debugging

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

Enable AEL contexts debug *CLI¿ ael debug contexts

Enable AEL macros debug *CLI¿ ael debug macros

Enable AEL read debug *CLI¿ ael debug read

Enable AEL tokens debug *CLI¿ ael debug tokens

Disable AEL debug messages *CLI¿ ael no debug

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

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

4.5 About "aelparse"

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

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

aelparse has two optional arguments:

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

4.6 General Notes about Syntax

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

You can just as easily say,

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

as you can say:

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

or:

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

or:

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

4.7 Keywords

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

The following are keywords in the AEL language:

- abstract
- context
- macro
- globals
- ignorepat
- switch
- if

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

4.8 Procedural Interface and Internals

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

This tree is then handed to the semantic check routine.

Then the tree is handed to the compiler.

After that, it is freed from memory.

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

4.8.1 AEL version 2 BNF

(hopefully, something close to bnf).

First, some basic objects

```
------------------------
<word>    a lexical token consisting of characters matching this pattern: [-a-zA-Z

<word3-list>  a concatenation of up to 3 <word>s.

<collected-word>  all characters encountered until the character that follows the
-------------------------

<file> :== <objects>

<objects> :== <object>
           | <objects> <object>


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

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


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


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


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


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


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


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


<element> :== <extension>
         | <includes>
```

```
          | <switches>
          | <eswitches>
          | <ignorepat>
          | <word> '='  <collected-word> ';'
          | ';'


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


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


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

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

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

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


<word3-list> :== <word>
       | <word> <word>
       | <word> <word> <word>

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


<statement> :== '{' <statements> '}'
       | <word> '='  <collected-word> ';'
       | 'goto' <target> ';'
       | 'jump' <jumptarget> ';'
```

```
        | <word> ':'
        | 'for' '('  <collected-word> ';'  <collected-word> ';' <collected-word> ')
        | 'while' '('  <collected-word> ')' <statement>
        | <switch_head> '}'
        | <switch_head> <case_statements> '}'
        | '&' macro_call ';'
        | <application_call> ';'
        | <application_call> '='  <collected-word> ';'
        | 'break' ';'
        | 'return' ';'
        | 'continue' ';'
        | <random_head> <statement>
        | <random_head> <statement> 'else' <statement>
        | <if_head> <statement>
        | <if_head> <statement> 'else' <statement>
        | <ifTime_head> <statement>
        | <ifTime_head> <statement> 'else' <statement>
        | ';'

<target> :== <word>
        | <word> '|' <word>
        | <word> '|' <word> '|' <word>
        | 'default' '|' <word> '|' <word>
        | <word> ',' <word>
        | <word> ',' <word> ',' <word>
        | 'default' ',' <word> ',' <word>

<jumptarget> :== <word>
                | <word> ',' <word>
                | <word> ',' <word> '@' <word>
                | <word> '@' <word>
                | <word> ',' <word> '@' 'default'
                | <word> '@' 'default'

<macro_call> :== <word> '(' <eval_arglist> ')'
        | <word> '(' ')'

<application_call_head> :== <word>  '('
```

```
<application_call> :== <application_call_head> <eval_arglist> ')'
       | <application_call_head> ')'

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

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


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

<macro_statements> :== <macro_statement>
       | <macro_statements> <macro_statement>

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

<switches> :== 'switches' '{' <switchlist> '}'
       | 'switches' '{' '}'

<eswitches> :== 'eswitches' '{' <switchlist> '}'
       | 'eswitches' '{'  '}'

<switchlist> :== <word> ';'
       | <switchlist> <word> ';'

<includeslist> :== <includedname> ';'
       | <includedname> '|' <word3-list> ':' <word3-list> ':' <word3-list> '|' <wo
       | <includedname> '|' <word> '|' <word3-list> '|' <word3-list> '|' <word3-li
```

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

<includedname> :== <word>
        | 'default'

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

4.9 AEL Example USAGE

4.9.1 Comments

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

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

4.9.2 Context

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

```
context default {

}


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

\begin{verbatim}
```

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

4.9.3 Extensions

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

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

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

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

```
context default {

    regexten _5XXX => NoOp(it's a pattern!);
}

context default {

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

context default {

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

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

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

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

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

4.9.4 Includes

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

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

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

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

4.9.5 #include

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

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

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

4.9.6 Dialplan Switches

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

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

4.9.7 Ignorepat

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

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

4.9.8 Variables

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

Global variables are set in their own block.

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

Variables can be set within extensions as well.

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

NOTE: AEL wraps the right hand side of an assignment with $[ ] to allow expressions to be used If this is unwanted, you can protect the right hand side from being wrapped by using the Set() application. Read the README.variables about the requirements and behavior of $[ ] expressions.

NOTE: These things are wrapped up in a $[ ] expression: The while() test; the if() test; the middle expression in the for(x; y; z) statement (the y expression); Assignments - the right hand side, so a = b -¿ Set(a=$[b])

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

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

4.9.9 Loops

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

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

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

4.9.10 Conditionals

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

```
context conditional {
    _8XXX => {
         Dial(SIP/${EXTEN});
         if ("${DIALSTATUS}" = "BUSY")
         {
              NoOp(yessir);
              Voicemail(${EXTEN}|b);
         }
```

```
     else
          Voicemail(${EXTEN}|u);
     ifTime (14:00-25:00|sat-sun|*|*)
          Voicemail(${EXTEN}|b);
     else
     {
          Voicemail(${EXTEN}|u);
          NoOp(hi, there!);
     }
     random(51) NoOp(This should appear 51% of the time);

     random( 60 )
     {
                  NoOp( This should appear 60% of the time );
     }
     else
     {
                  random(75)
                  {
                          NoOp( This should appear 30% of the time! );
                  }
                  else
                  {
                          NoOp( This should appear 10% of the time! );
                  }
      }
}
_777X => {
     switch (${EXTEN}) {
          case 7771:
               NoOp(You called 7771!);
               break;
          case 7772:
               NoOp(You called 7772!);
               break;
          case 7773:
               NoOp(You called 7773!);
               // fall thru-
```

```
                pattern 777[4-9]:
                      NoOp(You called 777 something!);
                default:
                     NoOp(In the default clause!);
          }
     }
}
```

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

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

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

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

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

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

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

4.9.11 Break, Continue, and Return

Three keywords, break, continue, and return, are included in the syntax to provide flow of control to loops, and switches.

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

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

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

4.9.12 goto, jump, and labels

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

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

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

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

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

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

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

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

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

NOTE: A NEW addition to AEL: you can now use ',' instead of '—' to separate the items in the target address. You can't have a mix, though, of '—' and ',' in the target. It's either one, or the other.

4.9.13 Macros

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

same name. A catch block can be specified to catch special extensions.

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

       }
       catch a {
               VoiceMailMain(${ext});
               return;
       }
}
```

A macro is then called by preceding the macro name with an ampersand. Empty arguments can be passed simply with nothing between comments(0.11).

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

4.10 Examples

```
context demo {
    s => {
         Wait(1);
         Answer();
         TIMEOUT(digit)=5;
         TIMEOUT(response)=10;
restart:
```

```
        Background(demo-congrats);
instructions:
        for (x=0; ${x} < 3; x=${x} + 1) {
             Background(demo-instruct);
             WaitExten();
        }
    }
    2 => {
         Background(demo-moreinfo);
         goto s|instructions;
    }
    3 => {
         LANGUAGE()=fr;
         goto s|restart;
    }

    500 => {
         Playback(demo-abouttotry);
         Dial(IAX2/guest@misery.digium.com);
         Playback(demo-nogo);
         goto s|instructions;
    }
    600 => {
         Playback(demo-echotest);
         Echo();
         Playback(demo-echodone);
         goto s|instructions;
    }
    # => {
hangup:
         Playback(demo-thanks);
         Hangup();
    }
    t => goto #|hangup;
    i => Playback(invalid);
}
```

4.11 Semantic Checks

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

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

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

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

Differences with the original version of AEL ==========

1. The $[...] expressions have been enhanced to include the ==, ——, and && operators. These operators are exactly equivalent to the =, —, and & operators, respectively. Why? So the C, Java, C++ hackers feel at home here.
2. It is more free-form. The newline character means very little, and is pulled out of the white-space only for line numbers in error messages.
3. It generates more error messages – by this I mean that any difference between the input and the grammar are reported, by file, line number, and column.

4. It checks the contents of $[ ] expressions (or what will end up being $[ ] expressions!) for syntax errors. It also does matching paren/bracket counts.

5. It runs several semantic checks after the parsing is over, but before the compiling begins, see the list above.

6. It handles #include "filepath" directives. – ALMOST anywhere, in fact. You could easily include a file in a context, in an extension, or at the root level. Files can be included in files that are included in files, down to 50 levels of hierarchy...

7. Local Goto's inside Switch statements automatically have the extension of the location of the switch statement appended to them.

8. A pretty printer function is available within pbx_ael.so.

9. In the utils directory, two standalone programs are supplied for debugging AEL files. One is called "aelparse", and it reads in the /etc/asterisk/extensions.ael file, and shows the results of syntax and semantic checking on stdout, and also shows the results of compilation to stdout. The other is "aelparse1", which uses the original ael compiler to do the same work, reading in "/etc/asterisk/extensions.ael", using the original 'pbx_ael.so' instead.

10. AEL supports the "jump" statement, and the "pattern" statement in switch constructs. Hopefully these will be documented in the AEL README.

11. Added the "return" keyword, which will jump to the end of an extension/Macro.

12. Added the ifTime (¡time range¿—¡days of week¿—¡days of month¿—¡months¿) [else] construct, which executes much like an if () statement, but the decision is based on the current time, and the time spec provided in the ifTime. See the example above. (Note: all the other time-dependent Applications can be used via ifTime)

13. Added the optional time spec to the contexts in the includes construct. See examples above.

14. You don't have to wrap a single "true" statement in curly braces, as in the original AEL. An "else" is attached to the closest if. As usual, be careful about nested if statements! When in doubt, use curlies!

15. Added the syntax [regexten] [hint(channel)] to precede an extension declaration. See examples above, under "Extension". The regexten keyword will cause the priorities in the extension to begin with 2 instead of 1. The hint keyword will cause its arguments to be inserted in the extension under the hint priority. They are both optional, of course, but the order is fixed at the moment– the regexten must come before the hint, if they are both present.

16. Empty case/default/pattern statements will "fall thru" as expected. (0.6)

17. A trailing label in an extension, will automatically have a NoOp() added, to make sure the label exists in the extension on Asterisk. (0.6)

18. (0.9) the semicolon is no longer required after a closing brace! (i.e. "];" ===¿ "}". You can have them there if you like, but they are not necessary. Someday they may be rejected as a syntax error, maybe.

19. (0.9) the // comments are not recognized and removed in the spots where expressions are gathered, nor in application call arguments. You may have to move a comment if you get errors in existing files.

20. (0.10) the random statement has been added. Syntax: random (¡expr¿) ¡lucky-statement¿ [else ¡unlucky-statement¿]. The probability of the lucky-statement getting executed is ¡expr¿, which should evaluate to an integer between 0 and 100. If the ¡lucky-statement¿ isn't so lucky this time around, then the ¡unlucky-statement¿ gets executed, if it is present.

4.12 Hints and Bugs

The safest way to check for a null strings is to say $["$x" = ""] The old way would do as shell scripts often do, and append something on both sides, like this: $[$xfoo = foo]. The trouble with the old way, is that, if x contains any spaces, then problems occur, usually syntax errors. It is better practice

and safer wrap all such tests with double quotes! Also, there are now some functions that can be used in a variable reference, ISNULL(), and LEN(), that can be used to test for an empty string: $ISNULL($x) or $[$LEN($x) = 0].

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

4.13 The Full Power of AEL

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

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

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

Chapter 5

SLA (Shared Line Appearances)

5.1 Introduction

The "SLA" functionality in Asterisk is intended to allow a setup that emulates a simple key system. It uses the various abstraction layers already built into Asterisk to emulate key system functionality across various devices, including IP channels.

5.2 Configuration

5.2.1 Summary

An SLA system is built up of virtual trunks and stations mapped to real Asterisk devices. The configuration for all of this is done in three different files: extensions.conf, sla.conf, and the channel specific configuration file such as sip.conf or zapata.conf.

5.2.2 Dialplan

The SLA implementation can automatically generate the dialplan necessary for basic operation if the "autocontext" option is set for trunks and stations in sla.conf. However, for reference, here is an automatically generated dialplan

to help with custom building of the dialplan to include other features, such as voicemail (5.3.2).

However, note that there is a little bit of additional configuration needed if the trunk is an IP channel. This is discussed in the section on trunks (5.2.3).

There are extensions for incoming calls on a specific trunk, which execute the SLATrunk application, as well as incoming calls from a station, which execute SLAStation. Note that there are multiple extensions for incoming calls from a station. This is because the SLA system has to know whether the phone just went off hook, or if the user pressed a specific line button.

Also note that there is a hint for every line on every station. This lets the SLA system control each individual light on every phone to ensure that it shows the correct state of the line. The phones must subscribe to the state of each of their line appearances.

Please refer to the examples section for full dialplan samples for SLA.

5.2.3 Trunks

An SLA trunk is a mapping between a virtual trunk and a real Asterisk device. This device may be an analog FXO line, or something like a SIP trunk. A trunk must be configured in two places. First, configure the device itself in the channel specific configuration file such as zapata.conf or sip.conf. Once the trunk is configured, then map it to an SLA trunk in sla.conf.

```
[line1]
type=trunk
device=Zap/1
```

Be sure to configure the trunk's context to be the same one that is set for the "autocontext" option in sla.conf if automatic dialplan configuration is used. This would be done in the regular device entry in zapata.conf, sip.conf, etc. Note that the automatic dialplan generation creates the SLATrunk() extension at extension 's'. This is perfect for Zap channels that are FXO trunks, for example. However, it may not be good enough for an IP trunk, since the call coming in over the trunk may specify an actual number.

If the dialplan is being built manually, ensure that calls coming in on a trunk execute the SLATrunk() application with an argument of the trunk name, as shown in the dialplan example before.

IP trunks can be used, but they require some additional configuration to work.

For this example, let's say we have a SIP trunk called "mytrunk" that is going to be used as line4. Furthermore, when calls come in on this trunk, they are going to say that they are calling the number "12564286000". Also, let's say that the numbers that are valid for calling out this trunk are NANP numbers, of the form _1NXXNXXXXXX.

In sip.conf, there would be an entry for [mytrunk]. For [mytrunk], set context=line4.

```
[line4]
type=trunk
device=Local/disa@line4_outbound

[line4]
exten => 12564286000,1,SLATrunk(line4)

[line4_outbound]
exten => disa,1,Disa(no-password|line4_outbound)
exten => _1NXXNXXXXXX,1,Dial(SIP/\${EXTEN}@mytrunk)
```

So, when a station picks up their phone and connects to line 4, they are connected to the local dialplan. The Disa application plays dialtone to the phone and collects digits until it matches an extension. In this case, once the phone dials a number like 12565551212, the call will proceed out the SIP trunk.

5.2.4 Stations

An SLA station is a mapping between a virtual station and a real Asterisk device. Currently, the only channel driver that has all of the features necessary to support an SLA environment is chan_sip. So, to configure a SIP phone to use as a station, you must configure sla.conf and sip.conf.

```
[station1]
type=station
device=SIP/station1
trunk=line1
trunk=line2
```

Here are some hints on configuring a SIP phone for use with SLA:

1. Add the SIP channel as a [station] in sla.conf.
2. Configure the phone in sip.conf. If automatic dialplan configuration was used by enabling the "autocontext" option in sla.conf, then this entry in sip.conf should have the same context setting.
3. On the phone itself, there are various things that must be configured to make everything work correctly:

   Let's say this phone is called "station1" in sla.conf, and it uses trunks named "line1" and line2".

   (a) Two line buttons must be configured to subscribe to the state of the following extensions: - station1_line1 - station1_line2

   (b) The line appearance buttons should be configured to dial the extensions that they are subscribed to when they are pressed.

   (c) If you would like the phone to automatically connect to a trunk when it is taken off hook, then the phone should be automatically configured to dial "station1" when it is taken off hook.

5.3 Configuration Examples

5.3.1 Basic SLA

This is an example of the most basic SLA setup. It uses the automatic dialplan generation so the configuration is minimal.

sla.conf:

```
[line1]
type=trunk
device=Zap/1
autocontext=line1

[line2]
type=trunk
device=Zap/2
autocontext=line2
```

```
[station](!)
type=station
trunk=line1
trunk=line2
autocontext=sla_stations

[station1](station)
device=SIP/station1

[station2](station)
device=SIP/station2

[station3](station)
device=SIP/station3
```

With this configuration, the dialplan is generated automatically. The first zap channel should have its context set to "line1" and the second should be set to "line2" in zapata.conf. In sip.conf, station1, station2, and station3 should all have their context set to "sla_stations".

For reference, here is the automatically generated dialplan for this situation:

```
[line1]
exten => s,1,SLATrunk(line1)

[line2]
exten => s,2,SLATrunk(line2)

[sla_stations]
exten => station1,1,SLAStation(station1)
exten => station1_line1,hint,SLA:station1_line1
exten => station1_line1,1,SLAStation(station1_line1)
exten => station1_line2,hint,SLA:station1_line2
exten => station1_line2,1,SLAStation(station1_line2)

exten => station2,1,SLAStation(station2)
exten => station2_line1,hint,SLA:station2_line1
```

```
exten => station2_line1,1,SLAStation(station2_line1)
exten => station2_line2,hint,SLA:station2_line2
exten => station2_line2,1,SLAStation(station2_line2)

exten => station3,1,SLAStation(station3)
exten => station3_line1,hint,SLA:station3_line1
exten => station3_line1,1,SLAStation(station3_line1)
exten => station3_line2,hint,SLA:station3_line2
exten => station3_line2,1,SLAStation(station3_line2)
```

5.3.2 SLA and Voicemail

This is an example of how you could set up a single voicemail box for the phone system. The voicemail box number used in this example is 1234, which would be configured in voicemail.conf.

For this example, assume that there are 2 trunks and 3 stations. The trunks are Zap/1 and Zap/2. The stations are SIP/station1, SIP/station2, and SIP/station3.

In zapata.conf, channel 1 has context=line1 and channel 2 has context=line2.

In sip.conf, all three stations are configured with context=sla_stations.

When the stations pick up their phones to dial, they are allowed to dial NANP numbers for outbound calls, or 8500 for checking voicemail.

sla.conf:

```
[line1]
type=trunk
device=Local/disa@line1_outbound

[line2]
type=trunk
device=Local/disa@line2_outbound

[station](!)
type=station
trunk=line1
trunk=line2

[station1](station)
```

```
device=SIP/station1

[station2](station)
device=SIP/station2

[station3](station)
device=SIP/station3
```

extensions.conf:

```
[macro-slaline]
exten => s,1,SLATrunk(${ARG1})
exten => s,n,Goto(s-${SLATRUNK_STATUS}|1)
exten => s-FAILURE,1,Voicemail(1234|u)
exten => s-UNANSWERED,1,Voicemail(1234|u)

[line1]
exten => s,1,Macro(slaline|line1)

[line2]
exten => s,2,Macro(slaline|line2)

[line1_outbound]
exten => disa,1,Disa(no-password|line1_outbound)
exten => _1NXXNXXXXXX,1,Dial(Zap/1/${EXTEN})
exten => 8500,1,VoicemailMain(1234)

[line2_outbound]
exten => disa,1,Disa(no-password|line2_outbound)
exten => _1NXXNXXXXXX,1,Dial(Zap/2/${EXTEN})
exten => 8500,1,VoicemailMain(1234)

[sla_stations]

exten => station1,1,SLAStation(station1)
exten => station1_line1,hint,SLA:station1_line1
exten => station1_line1,1,SLAStation(station1_line1)
exten => station1_line2,hint,SLA:station1_line2
```

```
exten => station1_line2,1,SLAStation(station1_line2)

exten => station2,1,SLAStation(station2)
exten => station2_line1,hint,SLA:station2_line1
exten => station2_line1,1,SLAStation(station2_line1)
exten => station2_line2,hint,SLA:station2_line2
exten => station2_line2,1,SLAStation(station2_line2)

exten => station3,1,SLAStation(station3)
exten => station3_line1,hint,SLA:station3_line1
exten => station3_line1,1,SLAStation(station3_line1)
exten => station3_line2,hint,SLA:station3_line2
exten => station3_line2,1,SLAStation(station3_line2)
```

5.4 Call Handling

5.4.1 Summary

This section is intended to describe how Asterisk handles calls inside of the SLA system so that it is clear what behavior is expected.

5.4.2 Station goes off hook (not ringing)

When a station goes off hook, it should initiate a call to Asterisk with the extension that indicates that the phone went off hook without specifying a specific line. In the examples in this document, for the station named "station1", this extension is simply named, "station1".

Asterisk will attempt to connect this station to the first available trunk that is not in use. Asterisk will check the trunks in the order that they were specified in the station entry in sla.conf. If all trunks are in use, the call will be denied.

If Asterisk is able to acquire an idle trunk for this station, then trunk is connected to the station and the station will hear dialtone. The station can then proceed to dial a number to call. As soon as a trunk is acquired, all appearances of this line on stations will show that the line is in use.

5.4.3 Station goes off hook (ringing)

When a station goes off hook while it is ringing, it should simply answer the call that had been initiated to it to make it ring. Once the station has answered, Asterisk will figure out which trunk to connect it to. It will connect it to the highest priority trunk that is currently ringing. Trunk priority is determined by the order that the trunks are listed in the station entry in sla.conf.

5.4.4 Line button on a station is pressed

When a line button is pressed on a station, the station should initiate a call to Asterisk with the extension that indicates which line button was pressed. In the examples given in this document, for a station named "station1" and a trunk named "line1", the extension would be "station1_line1".

If the specified trunk is not in use, then the station will be connected to it and will hear dialtone. All appearances of this trunk will then show that it is now in use.

If the specified trunk is on hold by this station, then this station will be reconnected to the trunk. The line appearance for this trunk on this station will now show in use. If this was the only station that had the call on hold, then all appearances of this trunk will now show that it is in use. Otherwise, all stations that are not currently connected to this trunk will show it on hold.

If the specified trunk is on hold by a different station, then this station will be connected to the trunk only if the trunk itself and the station(s) that have it on hold do not have private hold enabled. If connected, the appeareance of this trunk on this station will then show in use. All stations that are not currently connected to this trunk will show it on hold.

Chapter 6

Channel Drivers

6.1 IAX2

6.1.1 Introduction

This section is intended as an introduction to the Inter-Asterisk eXchange v2 (or simply IAX2) protocol. It provides both a theoretical background and practical information on its use.

6.1.2 Why IAX2?

The first question most people are thinking at this point is "Why do you need another VoIP protocol? Why didn't you just use SIP or H.323?"

Well, the answer is a fairly complicated one, but in a nutshell it's like this... Asterisk is intended as a very flexible and powerful communications tool. As such, the primary feature we need from a VoIP protocol is the ability to meet our own goals with Asterisk, and one with enough flexibility that we could use it as a kind of laboratory for inventing and implementing new concepts in the field. Neither H.323 or SIP fit the roles we needed, so we developed our own protocol, which, while not standards based, provides a number of advantages over both SIP and H.323, some of which are:

- Interoperability with NAT/PAT/Masquerade firewalls
  - IAX seamlessly interoperates through all sorts of NAT and PAT and other firewalls, including the ability to place and receive calls, and transfer calls to other stations.

- High performance, low overhead protocol
  - When running on low-bandwidth connections, or when running large numbers of calls, optimized bandwidth utilization is imperative. IAX uses only 4 bytes of overhead
- Internationalization support
  - IAX transmits language information, so that remote PBX content can be delivered in the native language of the calling party.
- Remote dialplan polling
  - IAX allows a PBX or IP phone to poll the availability of a number from a remote server. This allows PBX dialplans to be centralized.
- Flexible authentication
  - IAX supports cleartext, md5, and RSA authentication, providing flexible security models for outgoing calls and registration services.
- Multimedia protocol
  - IAX supports the transmission of voice, video, images, text, HTML, DTMF, and URL's. Voice menus can be presented in both audibly and visually.
- Call statistic gathering
  - IAX gathers statistics about network performance (including latency and jitter, as well as providing end-to-end latency measurement.
- Call parameter communication
  - Caller*ID, requested extension, requested context, etc are all communicated through the call.
- Single socket design
  - IAX's single socket design allows up to 32768 calls to be multiplexed.

While we value the importance of standards based (i.e. SIP) call handling, hopefully this will provide a reasonable explanation of why we developed IAX rather than starting with SIP.

6.1.3 Configuration

For examples of a configuration, please see the iax.conf.sample in your the /configs directory of you source code distribution.

6.1.4 IAX2 Jitterbuffer

The new jitterbuffer

You must add "jitterbuffer=yes" to either the [general] part of iax.conf, or to a peer or a user. (just like the old jitterbuffer). Also, you can set "maxjitterbuffer=n", which puts a hard-limit on the size of the jitterbuffer of "n milliseconds". It is not necessary to have the new jitterbuffer on both sides of a call; it works on the receive side only.

PLC

The new jitterbuffer detects packet loss. PLC is done to try to recreate these lost packets in the codec decoding stage, as the encoded audio is translated to slinear. PLC is also used to mask jitterbuffer growth.

This facility is enabled by default in iLBC and speex, as it has no additional cost. This facility can be enabled in adpcm, alaw, g726, gsm, lpc10, and ulaw by setting genericplc =¿ true in the [plc] section of codecs.conf.

Trunktimestamps

To use this, both sides must be using Asterisk v1.2 or later. Setting "trunktimestamps=yes" in iax.conf will cause your box to send 16-bit timestamps for each trunked frame inside of a trunk frame. This will enable you to use jitterbuffer for an IAX2 trunk, something that was not possible in the old architecture.

The other side must also support this functionality, or else, well, bad things will happen. If you don't use trunktimestamps, there's lots of ways the jitterbuffer can get confused because timestamps aren't necessarily sent through the trunk correctly.

Communication with Asterisk v1.0.x systems

You can set up communication with v1.0.x systems with the new jitterbuffer, but you can't use trunks with trunktimestamps in this communication.

If you are connecting to an Asterisk server with earlier versions of the software (1.0.x), do not enable both jitterbuffer and trunking for the involved peers/users in order to be able to communicate. Earlier systems will not support trunktimestamps.

You may also compile chan_iax2.c without the new jitterbuffer, enabling the old backwards compatible architecture. Look in the source code for instructions.

Testing and monitoring

You can test the effectiveness of PLC and the new jitterbuffer's detection of loss by using the new CLI command "iax2 test losspct ¡n¿". This will simulate n percent packet loss coming _in_ to chan_iax2. You should find that with PLC and the new JB, 10 percent packet loss should lead to just a tiny amount of distortion, while without PLC, it would lead to silent gaps in your audio.

"iax2 show netstats" shows you statistics for each iax2 call you have up. The columns are "RTT" which is the round-trip time for the last PING, and then a bunch of s tats for both the local side (what you're receiving), and the remote side (what the other end is telling us they are seeing). The remote stats may not be complete if the remote end isn't using the new jitterbuffer.

The stats shown are:

- Jit: The jitter we have measured (milliseconds)
- Del: The maximum delay imposed by the jitterbuffer (milliseconds)
- Lost: The number of packets we've detected as lost.
- %: The percentage of packets we've detected as lost recently.
- Drop: The number of packets we've purposely dropped (to lower latency).
- OOO: The number of packets we've received out-of-order
- Kpkts: The number of packets we've received / 1000.

Reporting problems

There's a couple of things that can make calls sound bad using the jitterbuffer:

1. The JB and PLC can make your calls sound better, but they can't fix everything. If you lost 10 frames in a row, it can't possibly fix that. It really can't help much more than one or two consecutive frames.

2. Bad timestamps: If whatever is generating timestamps to be sent to you generates nonsensical timestamps, it can confuse the jitterbuffer. In particular, discontinuities in timestamps will really upset it: Things like timestamps sequences which go 0, 20, 40, 60, 80, 34000, 34020, 34040, 34060... It's going to think you've got about 34 seconds of jitter in this case, etc.. The right solution to this is to find out what's causing the sender to send us such nonsense, and fix that. But we should also figure out how to make the receiver more robust in cases like this.

   chan_iax2 will actually help fix this a bit if it's more than 3 seconds or so, but at some point we should try to think of a better way to detect this kind of thing and resynchronize.

   Different clock rates are handled very gracefully though; it will actually deal with a sender sending 20% faster or slower than you expect just fine.

3. Really strange network delays: If your network "pauses" for like 5 seconds, and then when it restarts, you are sent some packets that are 5 seconds old, we are going to see that as a lot of jitter. We already throw away up to the worst 20 frames like this, though, and the "maxjitterbuffer" parameter should put a limit on what we do in this case.

6.2 mISDN

6.2.1 Introduction

This package contains the mISDN Channel Driver for the Asterisk PBX. It supports every mISDN Hardware and provides an interface for asterisk.

6.2.2 Features

- NT and TE mode
- PP and PMP mode
- BRI and PRI (with BNE1 and BN2E1 Cards)
- Hardware Bridging
- DTMF Detection in HW+mISDNdsp
- Display Messages on Phones (on those that support display msg)
- app_SendText
- HOLD/RETRIEVE/TRANSFER on ISDN Phones :)
- Screen/ Not Screen User Number
- EchoCancellation
- Volume Control
- Crypting with mISDNdsp (Blowfish)
- Data (HDLC) callthrough
- Data Calling (with app_ptyfork +pppd)
- Echo cancellation
- CallDeflection
- Some other

6.2.3 Fast Installation Guide

It is easy to install mISDN and mISDNuser. Just fetch the newest head of the cvs, this can be done by:

```
cvs -d:pserver:anonymous:readonly@cvs.isdn4linux.de:/i4ldev co mISDN mISDNuser
```

then compile and install both with:

```
cd mISDN ;
make && make install
```

(you will need at least your kernel headers to compile mISDN).

```
cd mISDNuser ;
make && make install
```

Now you can compile chan_misdn, just by making asterisk:

```
cd asterisk ;
./configure && make && make install
```

That's all!

Follow the instructions in the mISDN Package for how to load the Kernel Modules.

6.2.4 Pre-Requisites

To compile and install this driver, you'll need at least one mISDN Driver and the mISDNuser package. Chan_misdn works with both, the current release version and the development (svn trunk) version of Asterisk. mISDNuser and mISDN must be fetched from cvs.isdn4linux.de.

You should use Kernels ¿= 2.6.9

6.2.5 Configuration

First of all you must configure the mISDN drivers, please follow the instructions in the mISDN package to do that, the main config file and config script is:

```
/etc/init.d/misdn-init  and
/etc/misdn-init.conf
```

Now you will want to configure the misdn.conf file which resides in the asterisk config directory (normally /etc/asterisk).

misdn.conf: [general]

The misdn.conf file contains a "general" subsection, and user subsections which contain misdn port settings and different Asterisk contexts.

In the general subsection you can set options that are not directly port related. There is for example the very important debug variable which you can set from the Asterisk cli (command line interface) or in this configuration file, bigger numbers will lead to more debug output. There's also a tracefile option, which takes a path+filename where debug output is written to.

misdn.conf: [default] subsection

The default subsection is another special subsection which can contain all the options available in the user/port subsections. the user/port subsection inherit their parameters from the default subsection.

misdn.conf: user/port subsections

The user subsections have names which are unequal to "general". Those subsections contain the ports variable which mean the mISDN Ports. Here you can add multiple ports, comma separated.

Espacially for TE-Mode Ports there is a msns option. This option tells the chan_misdn driver to listen for incoming calls with the given msns, you can insert a '*' as single msn, which leads in getting every incoming call (if you want to share on PMP TE S0 with a asterisk and a phone or isdn card you should insert here the msns which you'll like to give the Asterisk). Finally a context variable resides in the user subsections, which tells chan_misdn where to send incoming calls to in the Asterisk dial plan (extension.conf).

Dial and Options String

The dial string of chan_misdn got more complex, because we added more features, so the generic dial string looks like:

```
mISDN/<port>|g:<group>/<extension>[/<OPTIONSSTRING>]

The Optionsstring looks Like:
:<optchar1><OptParam1>:<optchar2><OptParam2>
```

```
the ":" character is the delimiter.

The available Optchars are:
d - Send display text on called phone, text is the optparam
n - don't detect dtmf tones on called channel
h - make digital outgoing call
c - make crypted outgoing call, param is keyindex
e - perform echo cancellation on this channel,
    takes taps as arguments (32,64,128,256)
s - send Non Inband DTMF as inband
vr - rxgain control
vt - txgain control
```

chan_misdn registers a new dial plan application "misdn_set_opt" when loaded. This application takes the Optionsstring as argument. The Syntax is:

```
misdn_set_opt(<OPTIONSSTRING>)
```

When you set options in the dialstring, the options are set in the external channel. When you set options with misdn_set_opt, they are set in the current incoming channel. So if you like to use static encryption, the scenario looks as follows:

```
Phone1 --> * Box 1 --> PSTN_TE
PSTN_TE --> * Box 2 --> Phone2
```

The Encryption must be done on the PSTN sides, so the dialplan on the boxes are:

```
* Box 1:
exten => _${CRYPT_PREFIX}X.,1,Dial(mISDN/g:outbound/:c1)

* Box 2:
exten => ${CRYPT_MSN},1,misdn_set_opt(:c1)
exten => ${CRYPT_MSN},2,dial(${PHONE2})
```

6.2.6 mISDN CLI commands

At the Asterisk cli you can try to type in:

```
misdn <tab> <tab>
```

Now you should see the misdn cli commands:

```
- clean
-> pid  (cleans a broken call, use with care, leads often
 to a segmentation fault)
- send
-> display (sends a Text Message to a Asterisk channel,
 this channel must be an misdn channel)
- set
-> debug (sets debug level)
- show
-> config (shows the configuration options)
-> channels (shows the current active misdn channels)
-> channel (shows details about the given misdn channels)
-> stacks (shows the current ports, their protocols and states)
-> fullstacks (shows the current active and inactive misdn channels)

- restart
-> port (restarts given port (L2 Restart) )

- reload  (reloads misdn.conf)
```

You can only use "misdn send display" when an Asterisk channel is created and isdn is in the correct state. "correct state" means that you have established a call to another phone (mustn't be isdn though).

Then you use it like this:

misdn send display mISDN/1/101 "Hello World!"

where 1 is the Port of the Card where the phone is plugged in, and 101 is the msn (callerid) of the Phone to send the text to.

6.2.7 mISDN Variables

mISDN Exports/Imports a few Variables:

- MISDN_ADDRESS_COMPLETE : Is either set to 1 from the Provider, or you can set it to 1 to force a sending complete.

6.2.8 Debugging and sending bug reports

If you encounter problems, you should set up the debugging flag, usually debug=2 should be enough. the messages are divided in asterisk and misdn parts. Misdn Debug messages begin with an 'I', asterisk messages begin with an '*', the rest is clear I think.

Please take a trace of the problem and open a report in the Asterisk issue tracker at http://bugs.digium.com in the "channel drivers" project, "chan_misdn" category. Read the bug guidelines to make sure you provide all the information needed.

6.2.9 Examples

Here are some examples of how to use chan_misdn in the dialplan (extensions.conf):

```
[globals]
OUT_PORT=1 ; The physical Port of the Card
OUT_GROUP=ExternE1 ; The Group of Ports defined in misdn.conf

[misdnIn]
exten => _X.,1,Dial(mISDN/${OUT_PORT}/${EXTEN})
exten => _0X.,1,Dial(mISDN/g:${OUT_GROUP}/${EXTEN:1})
exten => _1X.,1,Dial(mISDN/g:${OUT_GROUP}/${EXTEN:1}/:dHello)
exten => _1X.,1,Dial(mISDN/g:${OUT_GROUP}/${EXTEN:1}/:dHello Test:n)
```

On the last line, you will notice the last argument (Hello); this is sent as Display Message to the Phone.

6.2.10 Known Problems

* I cannot hear any tone after a successful CONNECT to the other end

-> you forgot to load mISDNdsp, which is now needed by chan_misdn for switching and dtmf tone detection

6.3 Local

6.3.1 Introduction

chan_local is a pseudo-channel. Use of this channel simply loops calls back into the dialplan in a different context. Useful for recursive routing.

6.3.2 Syntax

```
Local/extension@context[/n]
```

Adding "/n" at the end of the string will make the Local channel not do a native transfer (the "n" stands for "n"o release) upon the remote end answering the line. This is an esoteric, but important feature if you expect the Local channel to handle calls exactly like a normal channel. If you do not have the "no release" feature set, then as soon as the destination (inside of the Local channel) answers the line, the variables and dial plan will revert back to that of the original call, and the Local channel will become a zombie and be removed from the active channels list. This is desirable in some circumstances, but can result in unexpected dialplan behavior if you are doing fancy things with variables in your call handling.

6.3.3 Purpose

The Local channel construct can be used to establish dialing into any part of the dialplan.

Imagine you have a TE410P in your box. You want to do something for which you must use a Dial statement (for instance when dropping files in /var/spool/outgoing) but you do want to be able to use your dialplans least-cost-routes or other intelligent stuff. What you could do before we had chan_local was create a cross-link between two ports of the TE410P and then Dial out one port and in the other. This way you could control where the call was going.

Of course, this was a nasty hack, and to make it more sensible, chan_local was built.

The "Local" channel driver allows you to convert an arbitrary extension into a channel. It is used in a variety of places, including agents, etc.

This also allows us to hop to contexts like a GoSub routine; See examples below.

6.3.4 Examples

```
[inbound] ; here falls all incoming calls
exten => s,1,Answer
exten => s,2,Dial(local/200@internal,30,r)
exten => s,3,Playback(sorrynoanswer)
exten => s,4,Hangup

[internal] ; here where our phones falls for default
exten => 200,1,Dial(sip/blah)
exten => 200,102,VoiceMail(${EXTEN}@default)

exten => 201,1,Dial(zap/1)
exten => 201,102,VoiceMail(${EXTEN}@default)

exten => _0.,1,Dial(Zap/g1/${EXTEN:1}) ; outgoing calls with 0+number
```

6.3.5 Caveats

If you use chan_local from a call-file and you want to pass channel variables into your context, make sure you append the '/n', because otherwise chan_local will 'optimize' itself out of the call-path, and the variables will get lost. i.e.

```
Local/00531234567@pbx becomes Local/00531234567@pbx/n
```

Chapter 7

Distributed Universal Number Discovery (DUNDi)

7.1 Introduction

http://www.dundi.com Mark Spencer, Digium, Inc.

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

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

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

7.2 Peering Agreement

DIGIUM GENERAL PEERING AGREEMENT (TM)
Version 1.0.0, September 2004

150 West Park Loop Suite 100, Huntsville, AL 35806 USA

DIGIUM GENERAL PEERING AGREEMENT

PREAMBLE

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

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

The Participants wish to protect competition, promote innovation and

value added services and make this service valuable both commercially and non-commercially. To that end, this GPA provides special terms and conditions outlining some permissible and non-permissible revenue sources.

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

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

The parties to this GPA agree as follows:

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

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

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

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

(d) The term "Egress Gateway" refers an Internet facility that provides a communications path to a Service or Services that may

not be directly addressable via the Internet.

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

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

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

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

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

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

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

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

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

1. PEERING. The undersigned Participants agree to Propagate Routes with each other and any other member of the Peering System and further agree not to Propagate DUNDi Routes with a third party unless they have first have executed this GPA (in its unmodified form) with such third party. The Participants further agree only to Propagate Routes with Participants whom they reasonably believe to be honoring the terms of the GPA. Participants may not insert, remove, amend, or otherwise modify any of the terms of the GPA.

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

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

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

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

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

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

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

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

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

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

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

(c) A Participant may provide Routes for those Services whose

Subscriber has authorized the Participant to do so, provided that the Participant is able to confirm that the Authorizing Individual is the Subscriber through:

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

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

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

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

(a) receipt of a revocation request;

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

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

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

6. TOLL SERVICES. No Participant may provide Routes for any Services that require payment from the calling party or their customer for communication with the Service. Nothing in this section shall prohibit

a Participant from providing routes for Services where the calling party may later enter into a financial transaction with the called party (e.g., a Participant may provide Routes for calling cards services).

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

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

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

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

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

12. WARRANTY. EACH PARTICIPANT WARRANTS TO THE OTHER PARTICIPANTS THAT IT MADE, AND WILL CONTINUE TO MAKE, A GOOD FAITH EFFORT TO AUTHENTICATE OTHERS IN THE PEERING SYSTEM AND TO PROVIDE ACCURATE INFORMATION IN

ACCORDANCE WITH THE TERMS OF THIS GPA. THIS WARRANTY IS MADE BETWEEN THE PARTICIPANTS, AND THE PARTICIPANTS MAY NOT EXTEND THIS WARRANTY TO ANY NON-PARTICIPANT INCLUDING END-USERS.

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

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

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

16. INDEMNIFICATION. Each Participant agrees to defend, indemnify and

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

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

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

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

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

(a) Prior to mediation or arbitration the disputing Participants shall seek informal resolution of disputes. The process shall be initiated with written notice of one Participant to the other

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

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

(c) If the dispute has not been resolved pursuant to (a) or (b) above, the dispute shall be promptly referred, no later than one (1) year from the date of original notice and subject to applicable statute of limitations, to binding arbitration in accordance with the UNCITRAL Arbitration Rules in effect on the date of this contract. The appointing authority shall be the International Centre for Dispute Resolution. The case shall be administered by the International Centre for Dispute Resolution under its Procedures for Cases under the UNCITRAL Arbitration Rules. Each Participant shall bear its own expenses and shall share equally in fees of the arbitrator. All arbitrators shall have substantial experience in information technology and/or in the telecommunications business and shall be selected by the disputing participants in accordance with UNCITRAL Arbitration Rules. If any arbitrator, once selected is unable or unwilling to continue for any reason, replacement shall be filled via the

process described above and a re-hearing shall be conducted. The disputing Participants will provide each other with all requested documents and records reasonably related to the dispute in a manner that will minimize the expense and inconvenience of both parties. Discovery will not include depositions or interrogatories except as the arbitrators expressly allow upon a showing of need. If disputes arise concerning discovery requests, the arbitrators shall have sole and complete discretion to resolve the disputes. The parties and arbitrator shall be guided in resolving discovery disputes by the Federal Rules of Civil Procedure. The Participants agree that time of the essence principles shall guide the hearing and that the arbitrator shall have the right and authority to issue monetary sanctions in the event of unreasonable delay. The arbitrator shall deliver a written opinion setting forth findings of fact and the rationale for the award within thirty (30) days following conclusion of the hearing. The award of the arbitrator, which may include legal and equitable relief, but which may not include punitive damages, will be final and binding upon the disputing Participants, and judgment may be entered upon it in accordance with applicable law in any court having jurisdiction thereof. In addition to award the arbitrator shall have the discretion to award the prevailing Participant all or part of its attorneys' fees and costs, including fees associated with arbitrator, if the arbitrator determines that the positions taken by the other Participant on material issues of the dispute were without substantial foundation. Any conflict between the UNCITRAL Arbitration Rules and the provisions of this GPA shall be controlled by this GPA.

21. INTEGRATED AGREEMENT. This GPA, constitutes the complete integrated agreement between the parties concerning the subject matter hereof. All prior and contemporaneous agreements, understandings, negotiations or representations, whether oral or in writing, relating to the subject matter of this GPA are superseded and canceled in their entirety.

22. WAIVER. No waiver of any of the provisions of this GPA shall be

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

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

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

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

Exhibit A

Weight Range	Requirements
0-99	May only be used under authorization of Owner

100-199	May only be used by the Owner's service provider, regardless of authorization.
200-299	Reserved -- do not use for e164 context.
300-399	May only be used by the owner of the code under which the Owner's number is a part of.
400-499	May be used by any entity providing access via direct connectivity to the Public Switched Telephone Network.
500-599	May be used by any entity providing access via indirect connectivity to the Public Switched Telephone Network (e.g. Via another VoIP provider)
600-	Reserved-- do not use for e164 context.

Participant	Participant
Company:	
Address:	
Email:	
_____	_____
Authorized Signature	Authorized Signature
Name:	

END OF GENERAL PEERING AGREEMENT

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

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

Chapter 8

ENUM

8.1 The ENUMLOOKUP dialplan function

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

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

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

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

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

8.1.1 Arguments

- number
  - telephone number or search string. Only numeric values within this string are parsed; all other digits are ignored for search, but are re-written during NAPTR regexp expansion.
- service_type
  - tel, sip, h323, iax2, mailto, ...[any other string], ALL. Default type is "sip". Special name of "ALL" will create a list of method types across all NAPTR records for the search number, and then put the results in an ordinal list starting with 1. The position ¡number¿ specified will then be returned, starting with 1 as the first record (lowest value) in the list. The service types are not hardcoded in Asterisk except for the default (sip) if no other service type specified; any method type string (IANA-approved or not) may be used except for the string "ALL".
- options
  - c
    - count. Returns the number of records of this type are returned (regardless of order or priority.) If "ALL" is the specified service_type, then a count of all methods will be returned for the DNS record.
- record#

  - which record to present if multiple answers are returned ¡integer¿ = The record in priority/order sequence based on the total count of records passed back by the query. If a service_type is specified, all entries of that type will be sorted into an ordinal list starting with 1 (by order first, then priority). The default of ¡options¿ is "1"

- zone_suffix
  - allows customization of the ENUM zone. Default is e164.arpa.

8.1.2 Examples

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

```
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 10 100 "u"
     "E2U+tel" "!^\\+13015611020$!tel:+12125551212!" .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 21 100 "u"
     "E2U+tel" "!^\\+13015611020$!tel:+14155551212!" .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 25 100 "u"
     "E2U+sip" "!^\\+13015611020$!sip:2203@sip.fox-den.com!" .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 26 100 "u"
     "E2U+sip" "!^\\+13015611020$!sip:1234@sip-2.fox-den.com!" .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 30 100 "u"
     "E2U+sip" "!^\\+*([^\\*]*)!sip:\\1@sip-3.fox-den.com!" .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 55 100 "u"
     "E2U+mailto" "!^\\+13015611020$!mailto:jtodd@fox-den.com!" .
```

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

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,,,,loligo.com)})
  returns: ${foo}="2203@sip.fox-den.com"
```

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

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,tel,,,loligo.com)})
  returns: ${foo}="+12125551212"
```

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

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,sip,c,,loligo.com)})
  returns: ${foo}=3
```

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

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,tel,,2,loligo.com)})
  returns: ${foo}="+14155551212"
```

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

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,ALL,c,,loligo.com)})
  returns: ${foo}=6
```

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

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+13015611020,ALL,,2,loligo.com)})
  returns: ${foo}="tel:+14155551212"  [note the "tel:" prefix in the string]
```

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

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+437203001721)})
  returns: ${foo}="enum-test@sip.nemox.net"  [note: this result is
  subject to change as it is "live" DNS and not under my control]
```

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

```
exten => 100,1,Set(foo=${ENUMLOOKUP(1234*256,,,,freenum.org)})
  returns: ${foo}="1234@204.91.156.10"  [note: this result is subject
  to change as it is "live" DNS]
```

Example 9: Give back the first SIP pointer for a number in the

```
enum.yoydynelabs.com zone (invalid lookup)
exten => 100,1,Set(foo=${ENUMLOOKUP(1234567890,sip,,1,enum.yoyodynelabs.com)})
  returns: ${foo}=""
```

8.1.3 Usage notes and subtle features

- The use of "+" in lookups is confusing, and warrants further explanation. All E.164 numbers ("global phone numbers") by definition need a leading "+" during ENUM lookup. If you neglect to add a leading "+", you may discover that numbers that seem to exist in the DNS aren't getting matched by the system or are returned with a null string result. This is due to the NAPTR reply requiring a "+" in the regular expression matching sequence. Older versions of Asterisk add a "+" from within the code, which may confuse administrators converting to the new function. Please ensure that all ENUM (e164.arpa) lookups contain a leading "+" before lookup, so ensure your lookup includes the leading plus sign. Other DNS trees may or may not require a leading "+" - check before using those trees, as it is possible the parsed NAPTRs will not provide correct results unless you have the correct dialed string. If you get console messages like "WARNING[24907]: enum.c:222 parse_naptr: NAPTR Regex match failed." then it is very possible that the returned NAPTR expects a leading "+" in the search string (or the returned NAPTR is mis-formed.)

- If a query is performed of type "c" ("count") and let's say you get back 5 records and then some seconds later a query is made against record 5 in the list, it may not be the case that the DNS resolver has the same answers as it did a second or two ago - maybe there are only 4 records in the list in the newest query. The resolver should be the canonical storage location for DNS records, since that is the intent of ENUM. However, some obscure future cases may have wildly changing NAPTR records within several seconds. This is a corner case, and probably only worth noting as a very rare circumstance. (note: I do

not object to Asterisk's dnsmgr method of locally caching DNS replies, but this method needs to honor the TTL given by the remote zone master. Currently, the ENUMLOOKUP function does not use the dnsmgr method of caching local DNS replies.)

- If you want strict NAPTR value ordering, then it will be necessary to use the "ALL" method to incrementally step through the different returned NAPTR pointers. You will need to use string manipulation to strip off the returned method types, since the results will look like "sip:12125551212" in the returned value. This is a non-trivial task, though it is required in order to have strict RFC compliance and to comply with the desires of the remote party who is presenting NAPTRs in a particular order for a reason.

- Default behavior for the function (even in event of an error) is to move to the next priority, and the result is a null value. Most ENUM lookups are going to be failures, and it is the responsibility of the dialplan administrator to manage error conditions within their dialplan. This is a change from the old app_enumlookup method and it's arbitrary priority jumping based on result type or failure.

- Anything other than digits will be ignored in lookup strings. Example: a search string of "+4372030blah01721" will turn into 1.2.7.1.0.0.3.0.2.7.3.4.e164.arpa. for the lookup. The NAPTR parsing may cause unexpected results if there are strings inside your NAPTR lookups.

- If there exist multiple records with the same weight and order as a result of your query, the function will RANDOMLY select a single NAPTR from those equal results.

- Currently, the function ignores the settings in enum.conf as the search zone name is now specified within the function, and the H323 driver can be chosen by the user via the dialplan. There were no other values in this file, and so it becomes deprecated.

- The function will digest and return NAPTRs which use older (deprecated) style, reversed method strings such as "sip+E2U" instead of the more modern "E2U+sip"

- There is no provision for multi-part methods at this time. If there are multiple NAPTRs with (as an example) a method of "E2U+voice:sip" and then another NAPTR in the same DNS record with a method of ""E2U+sip", the system will treat these both as method "sip" and they will be separate records from the perspective of the function. Of course, if both records point to the same URI and have equal priority/weight (as is often the case) then this will cause no serious difficulty, but it bears mentioning.

- ISN (ITAD Subscriber Number) usage: If the search number is of the form ABC*DEF (where ABC and DEF are at least one numeric digit) then perform an ISN-style lookup where the lookup is manipulated to C.B.A.DEF.domain.tld (all other settings and options apply.) See http://www.freenum.org/ for more details on ISN lookups. In the unlikely event you wish to avoid ISN re-writes, put an "n" as the first digit of the search string - the "n" will be ignored for the search.

8.1.4 Some more Examples

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

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

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

; example 3
;
; This example expects an ${EXTEN} that is an e.164 number (like
;  14102241145 or 437203001721)
; Search through e164.arpa and then also search through e164.org
;  to see if there are any valid SIP or IAX termination capabilities.
;  If none, send call out via Zap channel 1.
;
; Start first with e164.arpa zone...
;
exten => _X.,1,Set(sipcount=${ENUMLOOKUP(+${EXTEN},sip,c)}|counter=0)
exten => _X.,2,GotoIf($["${counter}"<"${sipcount}"]?3:6)
exten => _X.,3,Set(counter=$[${counter}+1])
exten => _X.,4,Dial(SIP/${ENUMLOOKUP(+${EXTEN},sip,,${counter})})
exten => _X.,5,GotoIf($["${counter}"<"${sipcount}"]?3:6)
;
exten => _X.,6,Set(iaxcount=${ENUMLOOKUP(+${EXTEN},iax2,c)}|counter=0)
exten => _X.,7,GotoIf($["${counter}"<"${iaxcount}"]?8:11)
exten => _X.,8,Set(counter=$[${counter}+1])
exten => _X.,9,Dial(IAX2/${ENUMLOOKUP(+${EXTEN},iax2,,${counter})})
exten => _X.,10,GotoIf($["${counter}"<"${iaxcount}"]?8:11)
```

```
;
exten => _X.,11,NoOp("No valid entries in e164.arpa for ${EXTEN} - checking in e16
;
; ...then also try e164.org, and look for SIP and IAX NAPTRs...
;
exten => _X.,12,Set(sipcount=${ENUMLOOKUP(+${EXTEN},sip,c,,e164.org)}|counter=0)
exten => _X.,13,GotoIf($["${counter}"<"${sipcount}"]?14:17)
exten => _X.,14,Set(counter=$[${counter}+1])
exten => _X.,15,Dial(SIP/${ENUMLOOKUP(+${EXTEN},sip,,${counter},e164.org)})
exten => _X.,16,GotoIf($["${counter}"<"${sipcount}"]?14:17)
;
exten => _X.,17,Set(iaxcount=${ENUMLOOKUP(+${EXTEN},iax2,c,,e164.org)}|counter=0)
exten => _X.,18,GotoIf($["${counter}"<"${iaxcount}"]?19:22)
exten => _X.,19,Set(counter=$[${counter}+1])
exten => _X.,20,Dial(IAX2/${ENUMLOOKUP(+${EXTEN},iax2,,${counter},e164.org)})
exten => _X.,21,GotoIf($["${counter}"<"${iaxcount}"]?19:22)
;
; ...then send out PRI.
;
exten => _X.,22,NoOp("No valid entries in e164.org for ${EXTEN} - sending out via
exten => _X.,23,Dial(Zap/g1/${EXTEN})
;
; end example 3
```

Chapter 9

AMI: Asterisk Manager Interface

9.1 The Asterisk Manager TCP/IP API

The manager is a client/server model over TCP. With the manager interface, you'll be able to control the PBX, originate calls, check mailbox status, monitor channels and queues as well as execute Asterisk commands.

AMI is the standard management interface into your Asterisk server. You configure AMI in manager.conf. By default, AMI is available on TCP port 5038 if you enable it in manager.conf.

AMI receive commands, called "actions". These generate a "response" from Asterisk. Asterisk will also send "Events" containing various information messages about changes within Asterisk. Some actions generate an initial response and data in the form list of events. This format is created to make sure that extensive reports do not block the manager interface fully.

Management users are configured in the configuration file manager.conf and are given permissions for read and write, where write represents their ability to perform this class of "action", and read represents their ability to receive this class of "event".

If you develop AMI applications, treat the headers in Actions, Events and Responses as local to that particular message. There is no cross-message standardization of headers.

If you develop applications, please try to reuse existing manager headers and their interpretation. If you are unsure, discuss on the asterisk-dev

mailing list.

9.2 Device status reports

Manager subscribes to extension status reports from all channels, to be able to generate events when an extension or device changes state. The level of details in these events may depend on the channel and device configuration. Please check each channel configuration file for more information. (in sip.conf, check the section on subscriptions and call limits)

9.3 Command Syntax

Management communication consists of tags of the form "header: value", terminated with an empty newline (
r
n) in the style of SMTP, HTTP, and other headers.

The first tag MUST be one of the following:

- Action: An action requested by the CLIENT to the Asterisk SERVER. Only one "Action" may be outstanding at any time.
- Response: A response to an action from the Asterisk SERVER to the CLIENT.
- Event: An event reported by the Asterisk SERVER to the CLIENT

9.4 Manager commands

To see all of the available manager commands, use the "manager show commands" CLI command.

You can get more information about a manager command with the "manager show command ¡command¿" CLI command in Asterisk.

9.5 Examples

```
Login - Log a user into the manager interface.
```

```
   Action: Login
   Username: testuser
   Secret: testsecret

Originate - Originate a call from a channel to an extension.

   Action: Originate
   Channel: sip/12345
   Exten: 1234
   Context: default

Originate - Originate a call from a channel to an extension without waiting
for call to complete.

   Action: Originate
   Channel: sip/12345
   Exten: 1234
   Context: default
   Async: yes


Redirect with ExtraChannel:
  Attempted goal:
  Have a 'robot' program Redirect both ends of an already-connected call
  to a meetme room using the ExtraChannel feature through the management interface

  Action: Redirect
  Channel: Zap/1-1
  ExtraChannel: SIP/3064-7e00 (varies)
  Exten: 680
  Priority: 1

Where 680 is an extension that sends you to a MeetMe room.
```

There are a number of GUI tools that use the manager interface, please search the mailing list archives and the documentation page on the http://www.asterisk.org web site for more information.

9.6 Some standard AMI headers

```
 Account:  -- Account Code (Status)
 AccountCode: -- Account Code (cdr_manager)
 ACL: <Y | N>-- Does ACL exist for object ?
 Action: <action>-- request or notification of a particular action
 Address-IP: -- IPaddress
 Address-Port: -- IP port number
 Agent: <string>-- Agent name
 AMAflags: -- AMA flag (cdr_manager, sippeers)
 AnswerTime: -- Time of answer (cdr_manager)
 Append: <bool>-- CDR userfield Append flag
 Application: -- Application to use
 Async: -- Whether or not to use fast setup
 AuthType:  -- Authentication type (for login or challenge)
"md5"
 BillableSeconds: -- Billable seconds for call (cdr_manager)
 CallerID: -- Caller id (name and number in Originate & cdr_manager)
 CallerID: -- CallerID number
   Number or "<unknown>" or "unknown"
(should change to "<unknown>" in app_queue)
 CallerID1:                      -- Channel 1 CallerID (Link event)
 CallerID2:                      -- Channel 2 CallerID (Link event)
 CallerIDName: -- CallerID name
   Name or "<unknown>" or "unknown"
(should change to "<unknown>" in app_queue)
 Callgroup: -- Call group for peer/user
 CallsTaken: <num>-- Queue status variable
 Cause: <value>-- Event change cause - "Expired"
 Cause: <value>-- Hangupcause (channel.c)
 CID-CallingPres: -- Caller ID calling presentation
 Channel: <channel>-- Channel specifier
 Channel: <dialstring>-- Dialstring in Originate
 Channel: <tech/[peer/username]> -- Channel in Registry events (SIP, IAX2)
 Channel: <tech>-- Technology (SIP/IAX2 etc) in Registry events
 ChannelType: -- Tech: SIP, IAX2, ZAP, MGCP etc
 Channel1: -- Link channel 1
 Channel2: -- Link channel 2
```

```
ChanObjectType: -- "peer", "user"
Codecs: -- Codec list
CodecOrder: -- Codec order, separated with comma ","
Command: -- Cli command to run
Context: -- Context
Count: <num>-- Number of callers in queue
Data: -- Application data
Default-addr-IP: -- IP address to use before registration
Default-Username: -- Username part of URI to use before registration
Destination: -- Destination for call (Dialstring ) (dial, cdr_manager)
DestinationContext: -- Destination context (cdr_manager)
DestinationChannel: -- Destination channel (cdr_manager)
DestUniqueID: -- UniqueID of destination (dial event)
Disposition: -- Call disposition (CDR manager)
Domain: <domain>-- DNS domain
Duration: <secs>-- Duration of call (cdr_manager)
Dynamic: <Y |  N>-- Device registration supported?
Endtime: -- End time stamp of call (cdr_manager)
EventList: <flag>-- Flag being "Start", "End", "Cancelled" or "ListObject"
Events: <eventmask>-- Eventmask filter ("on", "off", "system", "call", "log")
Exten: -- Extension (Redirect command)
Extension: -- Extension (Status)
Family: <string>-- ASTdb key family
File: <filename>-- Filename (monitor)
Format: <format>-- Format of sound file (monitor)
From: <time>--  Parking time (ParkedCall event)
Hint: -- Extension hint
Incominglimit: -- SIP Peer incoming limit
Key:
Key: -- ASTdb Database key
LastApplication: -- Last application executed (cdr_manager)
LastCall: <num>-- Last call in queue
LastData: -- Data for last application (cdr_manager)
Link: -- (Status)
ListItems: <number>-- Number of items in Eventlist (Optionally sent in "end" pack
Location: -- Interface (whatever that is -maybe tech/name in app_queue )
Loginchan: -- Login channel for agent
Logintime: <number>-- Login time for agent
```

```
 Mailbox: -- VM Mailbox (id@vmcontext) (mailboxstatus, mailboxcount)
 MD5SecretExist: <Y | N>-- Whether secret exists in MD5 format
 Membership: <string>-- "Dynamic" or "static" member in queue
 Message: <text>-- Text message in ACKs, errors (explanation)
 Mix: <bool>  -- Boolean parameter (monitor)
 NewMessages: <count>  -- Count of new Mailbox messages (mailboxcount)
 Newname:
 ObjectName: -- Name of object in list
 OldName: -- Something in Rename (channel.c)
 OldMessages: <count>-- Count of old mailbox messages (mailboxcount)
 Outgoinglimit: -- SIP Peer outgoing limit
 Paused: <num>-- Queue member paused status
 Peer: <tech/name>-- "channel" specifier :-)
 PeerStatus: <tech/name>-- Peer status code
"Unregistered", "Registered", "Lagged", "Reachable"
 Penalty: <num>-- Queue penalty
 Priority: -- Extension priority
 Privilege: <privilege>-- AMI authorization class (system, call, log, verbose, com
 Pickupgroup: -- Pickup group for peer
 Position: <num>-- Position in Queue
 Queue: -- Queue name
 Reason: -- "Autologoff"
 Reason: -- "Chanunavail"
 Response: <response>-- response code, like "200 OK"
"Success", "Error", "Follows"
 Restart: -- "True", "False"
 RegExpire: -- SIP registry expire
 RegExpiry: -- SIP registry expiry
 Reason: -- Originate reason code
 Seconds: -- Seconds (Status)
 Secret: <password>-- Authentication secret (for login)
 SecretExist: <Y | N>-- Whether secret exists
 Shutdown: -- "Uncleanly", "Cleanly"
 SIP-AuthInsecure:
 SIP-FromDomain: -- Peer FromDomain
 SIP-FromUser: -- Peer FromUser
 SIP-NatSupport:
 SIPLastMsg:
```

```
 Source: -- Source of call (dial event, cdr_manager)
 SrcUniqueID: -- UniqueID of source (dial event)
 StartTime:  -- Start time of call (cdr_manager)
 State: -- Channel state
 Status: -- Registration status (Registry events SIP)
 Status: -- Extension status (Extensionstate)
 Status: -- Peer status (if monitored)  ** Will change name **
"unknown", "lagged", "ok"
 Status: <num>-- Queue Status
 Status: -- DND status (DNDState)
 Time: <sec>-- Roundtrip time (latency)
 Timeout: -- Parking timeout time
 Timeout: -- Timeout for call setup (Originate)
 Timeout: <seconds>-- Timeout for call
 Uniqueid: -- Channel Unique ID
 Uniqueid1: -- Channel 1 Unique ID (Link event)
 Uniqueid2: -- Channel 2 Unique ID (Link event)
 User: -- Username (SIP registry)
 UserField: -- CDR userfield (cdr_manager)
 Val: -- Value to set/read in ASTdb
 Variable: -- Variable AND value to set (multiple separated with | in Originate)
 Variable: <name>-- For channel variables
 Value: <value>-- Value to set
 VoiceMailbox: -- VM Mailbox in SIPpeers
 Waiting: -- Count of mailbox messages (mailboxstatus)
```

** Please try to re-use existing headers to simplify manager message parsing in clients.

Read the CODING-GUIDELINES if you develop new manager commands or events.

9.7 Asynchronous Javascript Asterisk Manger (AJAM)

AJAM is a new technology which allows web browsers or other HTTP enabled applications and web pages to directly access the Asterisk Manger Interface

(AMI) via HTTP. Setting up your server to process AJAM involves a few steps:

9.7.1 Setup the Asterisk HTTP server

1. Uncomment the line "enabled=yes" in /etc/asterisk/http.conf to enable Asterisk's builtin micro HTTP server.

2. If you want Asterisk to actually deliver simple HTML pages, CSS, javascript, etc. you should uncomment "enablestatic=yes"

3. Adjust your "bindaddr" and "bindport" settings as appropriate for your desired accessibility

4. Adjust your "prefix" if appropriate, which must be the beginning of any URI on the server to match. The default is "asterisk" and the rest of these instructions assume that value.

9.7.2 Allow Manager Access via HTTP

1. Make sure you have both "enabled = yes" and "webenabled = yes" setup in /etc/asterisk/manager.conf

2. You may also use "httptimeout" to set a default timeout for HTTP connections.

3. Make sure you have a manager username/secret

Once those configurations are complete you can reload or restart Asterisk and you should be able to point your web browser to specific URI's which will allow you to access various web functions. A complete list can be found by typing "show http" at the Asterisk CLI.

examples:

http://localhost:8088/asterisk/manager?action=login&username=foo&secret=bar

This logs you into the manager interface's "HTML" view. Once you're logged in, Asterisk stores a cookie on your browser (valid for the length of httptimeout) which is used to connect to the same session.

http://localhost:8088/asterisk/rawman?action=status

Assuming you've already logged into manager, this URI will give you a "raw" manager output for the "status" command.

http://localhost:8088/asterisk/mxml?action=status

This will give you the same status view but represented as AJAX data, theoretically compatible with RICO (http://www.openrico.org).

http://localhost:8088/asterisk/static/ajamdemo.html

If you have enabled static content support and have done a make install, Asterisk will serve up a demo page which presents a live, but very basic, "astman" like interface. You can login with your username/secret for manager and have a basic view of channels as well as transfer and hangup calls. It's only tested in Firefox, but could probably be made to run in other browsers as well.

A sample library (astman.js) is included to help ease the creation of manager HTML interfaces.

Note that for the demo, there is no need for *any* external web server.

9.7.3 Integration with other web servers

Asterisk's micro HTTP server is *not* designed to replace a general purpose web server and it is intentionally created to provide only the minimal interfaces required. Even without the addition of an external web server, one can use Asterisk's interfaces to implement screen pops and similar tools pulling data from other web servers using iframes, div's etc. If you want to integrate CGI's, databases, PHP, etc. you will likely need to use a more traditional web server like Apache and link in your Asterisk micro HTTP server with something like this:

ProxyPass /asterisk http://localhost:8088/asterisk

Chapter 10

CDR: Call Detail Records

10.1 Applications

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

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

10.2 Fields of the CDR in Asterisk

- accountcode: What account number to use, (string, 20 characters)
- src: Caller*ID number (string, 80 characters)

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

In some cases, uniqueid is appended:

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

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

10.3 CDR Variables

If the channel has a cdr, that cdr record has its own set of variables which can be accessed just like channel variables. The following builtin variables are available.

```
${CDR(clid)} Caller ID
${CDR(src)} Source
${CDR(dst)} Destination
${CDR(dcontext)} Destination context
${CDR(channel)} Channel name
${CDR(dstchannel)} Destination channel
${CDR(lastapp)} Last app executed
${CDR(lastdata)} Last app's arguments
${CDR(start)} Time the call started.
${CDR(answer)} Time the call was answered.
${CDR(end)} Time the call ended.
${CDR(duration)} Duration of the call.
${CDR(billsec)} Duration of the call once it was answered.
${CDR(disposition)} ANSWERED, NO ANSWER, BUSY
${CDR(amaflags)} DOCUMENTATION, BILL, IGNORE etc
${CDR(accountcode)} The channel's account code.
${CDR(uniqueid)} The channel's unique id.
${CDR(userfield)} The channels uses specified field.
```

In addition, you can set your own extra variables by using Set(CDR(name)=value). These variables can be output into a text-format CDR by using the cdr_custom CDR driver; see the cdr_custom.conf.sample file in the configs directory for an example of how to do this. Call data records can be stored in many different databases or even CSV text.

10.4 MSSQL

Asterisk can currently store CDRs into an MSSQL database in two different ways: cdr_odbc or cdr_tds

Call Data Records can be stored using unixODBC (which requires the FreeTDS package) [cdr_odbc] or directly by using just the FreeTDS package

[cdr_tds] The following provide some examples known to get asterisk working with mssql.

NOTE: Only choose one db connector.

10.4.1 ODBC using cdr_odbc

Compile, configure, and install the latest unixODBC package:

```
tar -zxvf unixODBC-2.2.9.tar.gz &&
cd unixODBC-2.2.9 &&
./configure --sysconfdir=/etc --prefix=/usr --disable-gui &&
make &&
make install
```

Compile, configure, and install the latest FreeTDS package:

```
tar -zxvf freetds-0.62.4.tar.gz &&
cd freetds-0.62.4 &&
./configure --prefix=/usr --with-tdsver=7.0 \
                  --with-unixodbc=/usr/lib &&
make && make install
```

Compile, or recompile, asterisk so that it will now add support for cdr_odbc.

```
make clean && ./configure --with-odbc &&
make update &&
make &&
make install
```

Setup odbc configuration files. These are working examples from my system. You will need to modify for your setup. You are not required to store usernames or passwords here.

```
/etc/odbcinst.ini
   [FreeTDS]
   Description    = FreeTDS ODBC driver for MSSQL
   Driver         = /usr/lib/libtdsodbc.so
   Setup          = /usr/lib/libtdsS.so
   FileUsage      = 1
```

```
/etc/odbc.ini
   [MSSQL-asterisk]
   description         = Asterisk ODBC for MSSQL
   driver              = FreeTDS
   server              = 192.168.1.25
   port                = 1433
   database            = voipdb
   tds_version         = 7.0
   language            = us_english
```

Only install one database connector. Do not confuse asterisk by using both ODBC (cdr_odbc) and FreeTDS (cdr_tds). This command will erase the contents of cdr_tds.conf

```
[ -f /etc/asterisk/cdr_tds.conf ] > /etc/asterisk/cdr_tds.conf
```

NOTE: unixODBC requires the freeTDS package, but asterisk does not call freeTDS directly.

Now set up cdr_odbc configuration files. These are working samples from my system. You will need to modify for your setup. Define your usernames and passwords here, secure file as well.

```
/etc/asterisk/cdr_odbc.conf
   [global]
   dsn=MSSQL-asterisk
   username=voipdbuser
   password=voipdbpass
   loguniqueid=yes
```

And finally, create the 'cdr' table in your mssql database.

```
CREATE TABLE cdr (
        [calldate]      [datetime]              NOT NULL ,
        [clid]          [varchar] (80)          NOT NULL ,
        [src]           [varchar] (80)          NOT NULL ,
        [dst]           [varchar] (80)          NOT NULL ,
        [dcontext]      [varchar] (80)          NOT NULL ,
        [channel]       [varchar] (80)          NOT NULL ,
        [dstchannel]    [varchar] (80)          NOT NULL ,
        [lastapp]       [varchar] (80)          NOT NULL ,
```

```
        [lastdata]      [varchar] (80)          NOT NULL ,
        [duration]      [int]                   NOT NULL ,
        [billsec]       [int]                   NOT NULL ,
        [disposition]   [varchar] (45)          NOT NULL ,
        [amaflags]      [int]                   NOT NULL ,
        [accountcode]   [varchar] (20)          NOT NULL ,
        [uniqueid]      [varchar] (32)          NOT NULL ,
        [userfield]     [varchar] (255)         NOT NULL
)
```

Start asterisk in verbose mode, you should see that asterisk logs a connection to the database and will now record every call to the database when it's complete.

10.4.2 TDS, using cdr_tds

Compile, configure, and install the latest FreeTDS package:

```
tar -zxvf freetds-0.62.4.tar.gz &&
cd freetds-0.62.4 &&
./configure --prefix=/usr --with-tdsver=7.0
make &&
make install
```

Compile, or recompile, asterisk so that it will now add support for cdr_tds.

```
make clean && ./configure --with-tds &&
make update &&
make &&
make install
```

Only install one database connector. Do not confuse asterisk by using both ODBC (cdr_odbc) and FreeTDS (cdr_tds). This command will erase the contents of cdr_odbc.conf

```
[ -f /etc/asterisk/cdr_odbc.conf ] > /etc/asterisk/cdr_odbc.conf
```

Setup cdr_tds configuration files. These are working samples from my system. You will need to modify for your setup. Define your usernames and passwords here, secure file as well.

```
                /etc/asterisk/cdr_tds.conf
   [global]
   hostname=192.168.1.25
   port=1433
   dbname=voipdb
   user=voipdbuser
   password=voipdpass
   charset=BIG5
```

And finally, create the 'cdr' table in your mssql database.

```
CREATE TABLE cdr (
        [accountcode]   [varchar] (20)          NULL ,
        [src]           [varchar] (80)          NULL ,
        [dst]           [varchar] (80)          NULL ,
        [dcontext]      [varchar] (80)          NULL ,
        [clid]          [varchar] (80)          NULL ,
        [channel]       [varchar] (80)          NULL ,
        [dstchannel]    [varchar] (80)          NULL ,
        [lastapp]       [varchar] (80)          NULL ,
        [lastdata]      [varchar] (80)          NULL ,
        [start]         [datetime]              NULL ,
        [answer]        [datetime]              NULL ,
        [end]           [datetime]              NULL ,
        [duration]      [int]                   NULL ,
        [billsec]       [int]                   NULL ,
        [disposition]   [varchar] (20)          NULL ,
        [amaflags]      [varchar] (16)          NULL ,
        [uniqueid]      [varchar] (32)          NULL
)
```

Start asterisk in verbose mode, you should see that asterisk logs a connection to the database and will now record every call to the database when it's complete.

10.5 MYSQL

Using MySQL for CDR records is supported by using ODBC and the cdr_odbc module.

10.6 PGSQL

If you want to go directly to postgresql database, and have the cdr_pgsql.so compiled you can use the following sample setup. On Debian, before compiling asterisk, just install libpqxx-dev. Other distros will likely have a similiar package.

Once you have the compile done, copy the sample cdr_pgsql.conf file or create your own.

Here is a sample:

```
/etc/asterisk/cdr_pgsql.conf
  ; Sample Asterisk config file for CDR logging to PostgresSQL
  [global]
  hostname=localhost
  port=5432
  dbname=asterisk
  password=password
  user=postgres
  table=cdr
```

Now create a table in postgresql for your cdrs

```
CREATE TABLE cdr (
        calldate      time               NOT NULL ,
        clid          varchar (80)          NOT NULL ,
        src           varchar (80)          NOT NULL ,
        dst           varchar (80)          NOT NULL ,
        dcontext      varchar (80)          NOT NULL ,
        channel       varchar (80)          NOT NULL ,
        dstchannel    varchar (80)          NOT NULL ,
        lastapp       varchar (80)          NOT NULL ,
        lastdata      varchar (80)          NOT NULL ,
        duration      int                   NOT NULL ,
        billsec       int                   NOT NULL ,
        disposition   varchar (45)          NOT NULL ,
        amaflags      int                   NOT NULL ,
        accountcode   varchar (20)          NOT NULL ,
        uniqueid      varchar (32)          NOT NULL ,
        userfield     varchar (255)         NOT NULL
);
```

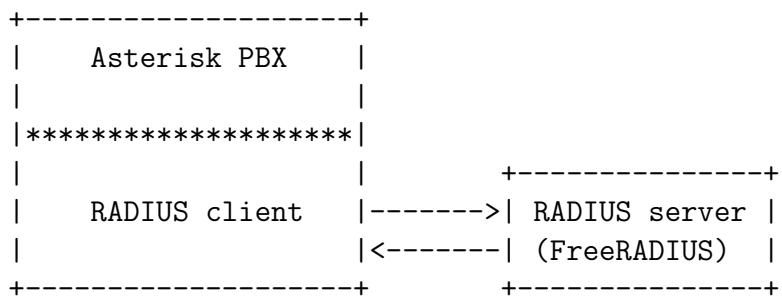
10.7 SQLLITE

SQLite version 2 is supported in cdr_sqlite.

10.8 RADIUS

10.8.1 What is needed

- FreeRADIUS server
- Radiusclient-ng library
- Asterisk PBX



10.8.2 Steps to follow in order to have RADIUS support

Installation of the Radiusclient library

Installation:

```
Download the sources from:

http://developer.berlios.de/projects/radiusclient-ng/

Untar the source tarball.
root@localhost:/usr/local/src# tar xvfz radiusclient-ng-0.5.2.tar.gz

Compile and install the library.
root@localhost:/usr/local/src# cd radiusclient-ng-0.5.2
```

```
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# ./configure
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# make
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# make install
```

Configuration of the Radiusclient library

By default all the configuration files of the radiusclient library will be in /usr/local/etc/radiusclient-ng directory.

File "radiusclient.conf" Open the file and find lines containing the following:

authserver localhost

This is the hostname or IP address of the RADIUS server used for authentication. You will have to change this unless the server is running on the same host as your Asterisk PBX.

acctserver localhost This is the hostname or IP address of the RADIUS server used for accounting. You will have to change this unless the server is running on the same host as your Asterisk PBX.

File "servers"

RADIUS protocol uses simple access control mechanism based on shared secrets that allows RADIUS servers to limit access from RADIUS clients.

A RADIUS server is configured with a secret string and only RADIUS clients that have the same secret will be accepted.

You need to configure a shared secret for each server you have configured in radiusclient.conf file in the previous step. The shared secrets are stored in /usr/local/etc/radiusclient-ng/servers file.

Each line contains hostname of a RADIUS server and shared secret used in communication with that server. The two values are separated by white spaces. Configure shared secrets for every RADIUS server you are going to use.

File "dictionary"

Asterisk uses some attributes that are not included in the dictionary of radiusclient library, therefore it is necessary to add them. A file called dictionary.digium (kept in the contrib dir) was created to list all new attributes used by Asterisk. Add to the end of the main dictionary file /usr/local/etc/radiusclient-ng/dictionary the line:

```
\$INCLUDE /path/to/dictionary.digium
```

Install FreeRADIUS Server (Version 1.1.1)

Download sources tarball from:
http://freeradius.org/
Untar, configure, build, and install the server:

```
root@localhost:/usr/local/src# tar xvfz freeradius-1.1.1.tar.gz
root@localhost:/usr/local/src# cd freeradius-1.1.1
root@localhost"/usr/local/src/freeradius-1.1.1# ./configure
root@localhost"/usr/local/src/freeradius-1.1.1# make
root@localhost"/usr/local/src/freeradius-1.1.1# make install
```

All the configuration files of FreeRADIUS server will be in /usr/local/etc/raddb directory.

Configuration of the FreeRADIUS Server

There are several files that have to be modified to configure the RADIUS server. These are presented next.

File "clients.conf"

File /usr/local/etc/raddb/clients.conf contains description of RADIUS clients that are allowed to use the server. For each of the clients you need to specify its hostname or IP address and also a shared secret. The shared secret must be the same string you configured in radiusclient library.

Example:

```
client myhost {
    secret = mysecret
    shortname = foo
}
```

This fragment allows access from RADIUS clients on "myhost" if they use "mysecret" as the shared secret. The file already contains an entry for localhost (127.0.0.1), so if you are running the RADIUS server on the same host as your Asterisk server, then modify the existing entry instead, replacing the default password.

File "dictionary"

Note : as of version 1.1.2, the dictionary.digium file ships with FreeRADIUS. The following procedure brings the dictionary.digium file to previous versions of FreeRADIUS.

File /usr/local/etc/raddb/dictionary contains the dictionary of FreeRADIUS server. You have to add the same dictionary file (dictionary.digium), which you added to the dictionary of radiusclient-ng library. You can include it into the main file, adding the following line at the end of file '/usr/local/etc/raddb/dictionary':

$INCLUDE /path/to/dictionary.digium

That will include the same new attribute definitions that are used in radiusclient-ng library so the client and server will understand each other.

Asterisk Accounting Configuration

Compilation and installation:

The module will be compiled as long as the radiusclient-ng library has been detected on your system.

By default FreeRADIUS server will log all accounting requests into /usr/local/var/log/radius/ra directory in form of plain text files. The server will create one file for each hostname in the directory. The following example shows how the log files look like.

Asterisk now generates Call Detail Records. See /include/asterisk/cdr.h for all the fields which are recorded. By default, records in comma separated values will be created in /var/log/asterisk/cdr-csv.

The configuration file for cdr_radius.so module is :

/etc/asterisk/cdr.conf This is where you can set CDR related parameters as well as the path to the radiusclient-ng library configuration file.

10.9 Logged Values

```
"Asterisk-Acc-Code",          The account name of detail records
"Asterisk-Src",
"Asterisk-Dst",
"Asterisk-Dst-Ctx",           The destination context
"Asterisk-Clid",
"Asterisk-Chan",              The channel
"Asterisk-Dst-Chan",        (if applicable)
"Asterisk-Last-App",        Last application run on the channel
"Asterisk-Last-Data",         Argument to the last channel
"Asterisk-Start-Time",
"Asterisk-Answer-Time",
```

```
"Asterisk-End-Time",
"Asterisk-Duration",          Duration is the whole length that the entire
                              call lasted. ie. call rx'd to hangup
                              "end time" minus "start time"
"Asterisk-Bill-Sec",          The duration that a call was up after other
                              end answered which will be <= to duration
                              "end time" minus "answer time"
"Asterisk-Disposition",      ANSWERED, NO ANSWER, BUSY
"Asterisk-AMA-Flags",         DOCUMENTATION, BILL, IGNORE etc, specified on
                              a per channel basis like accountcode.
"Asterisk-Unique-ID",         Unique call identifier
"Asterisk-User-Field"         User field set via SetCDRUserField
```

Chapter 11

Voicemail

11.1 ODBC Storage

ODBC Storage allows you to store voicemail messages within a database instead of using a file. This is *not* a full realtime engine and *only* supports ODBC. The table description for the "voicemessages" table is as follows:

```
+---------------+------------+------+-----+---------+-------+
| Field         | Type       | Null | Key | Default | Extra |
+---------------+------------+------+-----+---------+-------+
| msgnum        | int(11)    | YES  |     | NULL    |       |
| dir           | varchar(80)| YES  | MUL | NULL    |       |
| context       | varchar(80)| YES  |     | NULL    |       |
| macrocontext  | varchar(80)| YES  |     | NULL    |       |
| callerid      | varchar(40)| YES  |     | NULL    |       |
| origtime      | varchar(40)| YES  |     | NULL    |       |
| duration      | varchar(20)| YES  |     | NULL    |       |
| mailboxuser   | varchar(80)| YES  |     | NULL    |       |
| mailboxcontext| varchar(80)| YES  |     | NULL    |       |
| recording     | longblob   | YES  |     | NULL    |       |
+---------------+------------+------+-----+---------+-------+
```

The database name (from /etc/asterisk/res_odbc.conf) is in the "odbc-storage" variable in the general section of voicemail.conf.

You may modify the voicemessages table name by using odbctable=??? in voicemail.conf.

11.2 IMAP Storage

By enabling IMAP Storage, Asterisk will use native IMAP as the storage mechanism for voicemail messages instead of using the standard file structure.

Tighter integration of Asterisk voicemail and IMAP email services allows additional voicemail functionality, including:

- Listening to a voicemail on the phone will set its state to "read" in a user's mailbox automatically.
- Deleting a voicemail on the phone will delete it from the user's mailbox automatically.
- Accessing a voicemail recording email message will turn off the message waiting indicator (MWI) on the user's phone.
- Deleting a voicemail recording email will also turn off the message waiting indicator, and delete the message from the voicemail system.

11.2.1 Installation Notes

University of Washington IMAP C-Client

You will need a source distribution of University of Washington's IMAP c-client (http://www.washington.edu/imap/). Asterisk supports both the 2004 and 2006 versions of c-client, however mail_expunge_full is enabled in the 2006 version.

Note that Asterisk only uses the 'client' portion of the UW IMAP toolkit, but building it also builds an IMAP server and various other utilities. Because of this, the build instructions for the IMAP toolkit are somewhat complicated and can lead to confusion about what is needed.

If you are going to be connecting Asterisk to an existing IMAP server, then you don't need to care about the server or utilities in the IMAP toolkit at all. If you want to also install the UW IMAPD server, that is outside the scope of this document.

Building the c-client library is fairly straightforward; for example, on a Debian system there are two possibilities:

```
1) if you will not be using SSL to connect to the IMAP server:
   $ make slx SSLTYPE=none
```

```
2) if you will be using SSL to connect to the IMAP server:
   $ make slx EXTRACFLAGS="-I/usr/include/openssl"
```

Once this completes you can proceed with the Asterisk build; there is no need to run 'make install'.

Compiling Asterisk

Configure with ./configure –with-imap=/usr/src/imap or where ever you built thfe UWashington IMAP Toolkit. When you run 'make menuselect', choose 'Voicemail Build Options' and the IMAP_STORAGE option should be available for selection.

Note that the –with-imap option will NOT search your system for an installed copy of the IMAP Toolkit c-client library; the Asterisk Makefiles and configure script are designed to build against an unpacked and compiled source tree of the IMAP Toolkit, not a binary distribution.

After selecting it, use the 'x' key to exit menuselect and save your changes, and the build/install Asterisk normally.

11.2.2 Modify voicemail.conf

The following directives have been added to voicemail.conf:

```
imapserver=<name or IP address of IMAP mail server>
imapport=<IMAP port, defaults to 143>
imapflags=<IMAP flags, "novalidate-cert" for example>
expungeonhangup=<yes or no>
authuser=<username>
authpassword=<password>
```

The "expungeonhangup" flag is used to determine if the voicemail system should expunge all messages marked for deletion when the user hangs up the phone.

Each mailbox definition should also have imapuser=¡imap username¿. For example:

```
4123=>4123,James Rothenberger,jar@onebiztone.com,,attach=yes|imapuser=jar
```

The directives "authuser" and "authpassword" are not needed when using Kerberos. They are defined to allow Asterisk to authenticate as a single user that has access to all mailboxes as an alternative to Kerberos.

11.2.3 IMAP Folders

Besides INBOX, users should create "Old", "Work", "Family" and "Friends" IMAP folders at the same level of hierarchy as the INBOX. These will be used as alternate folders for storing voicemail messages to mimic the behavior of the current (file-based) voicemail system.

11.2.4 Separate vs. Shared Email Accounts

As administrator you will have to decide if you want to send the voicemail messages to a separate IMAP account or use each user's existing IMAP mailbox for voicemail storage. The IMAP storage mechanism will work either way.

By implementing a single IMAP mailbox, the user will see voicemail messages appear in the same INBOX as other messages. The disadvantage of this method is that if the IMAP server does NOT support UIDPLUS, Asterisk voicemail will expunge ALL messages marked for deletion when the user exits the voicemail system, not just the VOICEMAIL messages marked for deletion.

By implementing separate IMAP mailboxes for voicemail and email, voicemail expunges will not remove regular email flagged for deletion.

11.2.5 IMAP Server Implementations

There are various IMAP server implementations, each supports a potentially different set of features.

UW IMAP-2005 or earlier

UIDPLUS is currently NOT supported on these versions of UW-IMAP. Please note that without UID_EXPUNGE, Asterisk voicemail will expunge ALL messages marked for deletion when a user exits the voicemail system (hangs up the phone).

UW IMAP-2006 Development Branch

This version supports UIDPLUS, which allows UID_EXPUNGE capabilities. This feature allow the system to expunge ONLY pertinent messages, instead of the default behavior, which is to expunge ALL messages marked for deletion when EXPUNGE is called. The IMAP storage mechanism is this version of Asterisk will check if the UID_EXPUNGE feature is supported by the server, and use it if possible.

Cyrus IMAP

Cyrus IMAP server v2.3.3 has been tested using a hierarchy delimiter of '/'.

11.2.6 Quota Support

If the IMAP server supports quotas, Asterisk will check the quota when accessing voicemail. Currently only a warning is given to the user that their quota is exceeded.

11.2.7 Application Notes

Since the primary storage mechanism is IMAP, all message information that was previously stored in an associated text file, AND the recording itself, is now stored in a single email message. This means that the .gsm recording will ALWAYS be attached to the message (along with the user's preference of recording format if different - ie. .WAV). The voicemail message information is stored in the email message headers. These headers include:

```
X-Asterisk-VM-Message-Num
X-Asterisk-VM-Server-Name
X-Asterisk-VM-Context
X-Asterisk-VM-Extension
X-Asterisk-VM-Priority
X-Asterisk-VM-Caller-channel
X-Asterisk-VM-Caller-ID-Num
X-Asterisk-VM-Caller-ID-Name
X-Asterisk-VM-Duration
X-Asterisk-VM-Category
X-Asterisk-VM-Orig-date
```

X-Asterisk-VM-Orig-time

Chapter 12

SMS

12.1 Introduction

The SMS module for Asterisk was developed by Adrian Kennard, and is an implementation of the ETSI specification for landline SMS, ETSI ES 201 912, which is available from www.etsi.org. Landline SMS is starting to be available in various parts of Europe, and is available from BT in the UK. However, Asterisk would allow gateways to be created in other locations such as the US, and use of SMS capable phones such as the Magic Messenger. SMS works using analogue or ISDN lines.

12.2 Background

Short Message Service (SMS), or texting is very popular between mobile phones. A message can be sent between two phones, and normally contains 160 characters. There are ways in which various types of data can be encoded in a text message such as ring tones, and small graphic, etc. Text messaging is being used for voting and competitions, and also SPAM...

Sending a message involves the mobile phone contacting a message centre (SMSC) and passing the message to it. The message centre then contacts the destination mobile to deliver the message. The SMSC is responsible for storing the message and trying to send it until the destination mobile is available, or a timeout.

Landline SMS works in basically the same way. You would normally have a suitable text capable landline phone, or a separate texting box such as a

Magic Messenger on your phone line. This sends a message to a message centre your telco provides by making a normal call and sending the data using 1200 Baud FSK signaling according to the ETSI spec. To receive a message the message centre calls the line with a specific calling number, and the text capable phone answers the call and receives the data using 1200 Baud FSK signaling. This works particularly well in the UK as the calling line identity is sent before the first ring, so no phones in the house would ring when a message arrives.

12.3 Typical use with Asterisk

Sending messages from an Asterisk box can be used for a variety of reasons, including notification from any monitoring systems, email subject lines, etc.

Receiving messages to an Asterisk box is typically used just to email the messages to someone appropriate - we email and texts that are received to our direct numbers to the appropriate person. Received messages could also be used to control applications, manage competitions, votes, post items to IRC, anything.

Using a terminal such as a magic messenger, an Asterisk box could ask as a message centre sending messages to the terminal, which will beep and pop up the message (and remember 100 or so messages in its memory).

12.4 Terminology

- SMS - Short Message Service i.e. text messages
- SMSC - Short Message Service Centre The system responsible for storing and forwarding messages
- MO - Mobile Originated A message on its way from a mobile or landline device to the SMSC
- MT - Mobile Terminated A message on its way from the SMSC to the mobile or landline device
- RX - Receive A message coming in to the Asterisk box
- TX - Transmit A message going out of the Asterisk box

12.5 Sub address

When sending a message to a landline, you simply send to the landline number. In the UK, all of the mobile operators (bar one) understand sending messages to landlines and pass the messages to the BTText system for delivery to the landline.

The specification for landline SMS allows for the possibility of more than one device on a single landline. These can be configured with Sub addresses which are a single digit. To send a message to a specific device the message is sent to the landline number with an extra digit appended to the end. The telco can define a default sub address (9 in the UK) which is used when the extra digit is not appended to the end. When the call comes in, part of the calling line ID is the sub address, so that only one device on the line answers the call and receives the message.

Sub addresses also work for outgoing messages. Part of the number called by the device to send a message is its sub address. Sending from the default sub address (9 in the UK) means the message is delivered with the sender being the normal landline number. Sending from any other sub address makes the sender the landline number with an extra digit on the end.

Using Asterisk, you can make use of the sub addresses for sending and receiving messages. Using DDI (DID, i.e. multiple numbers on the line on ISDN) you can also make use of many different numbers for SMS.

12.6 extensions.conf

The following contexts are recommended.

```
; Mobile Terminated, RX. This is used when an incoming call from the SMS arrive
s, with the queue (called number and sub address) in ${EXTEN}
; Running an app after receipt of the text allows the app to find all messages
in the queue and handle them, e.g. email them.
; The app may be something like   smsq --process=somecommand --queue=${EXTEN}
to run a command for each received message
; See below for usage
[smsmtrx]
exten = _X.,1, SMS(${EXTEN}|a)
exten = _X.,2,System("someapptohandleincomingsms ${EXTEN}")
exten = _X.,3,Hangup
```

```
; Mobile originated, RX. This is receiving a message from a device, e.g. a Magi
c Messenger on a sip extension
; Running an app after receipt of the text allows the app to find all messages
in the queue and handle then, e.g. sending them to the public SMSC
; The app may be something like   smsq --process=somecommand --queue=${EXTEN}
to run a command for each received message
; See below for example usage
[smsmorx]
exten = _X.,1, SMS(${EXTEN}|sa)
exten = _X.,2,System("someapptohandlelocalsms ${EXTEN}")
exten = _X.,3,Hangup

   smsmtrx is normally accessed by an incoming call from the SMSC. In the
   UK this call is from a CLI of 080058752X0 where X is the sub address.
   As such a typical usage in the extensions.conf at the point of
   handling an incoming call is:-
exten = _X./8005875290,1,Goto(smsmtrx,${EXTEN},1)
exten = _X./_80058752[0-8]0,1,Goto(smsmtrx,${EXTEN}-${CALLERIDNUM:8:1},1)

   Alternatively, if you have the correct national prefix on incoming
   CLI, e.g. using zaphfc, you might use:-
exten = _X./08005875290,1,Goto(smsmtrx,${EXTEN},1)
exten = _X./_080058752[0-8]0,1,Goto(smsmtrx,${EXTEN}-${CALLERIDNUM:9:1},1)

   smsmorx is normally accessed by a call from a local sip device
   connected to a Magic Messenger. It could however by that you are
   operating Asterisk as a message centre for calls from outside. Either
   way, you look at the called number and goto smsmorx. In the UK, the
   SMSC number that would be dialed is 1709400X where X is the caller sub
   address. As such typical usage in extension.config at the point of
   handling a call from a sip phone is:-
exten = 17094009,1,Goto(smsmorx,${CALLERIDNUM},1)
exten = _1709400[0-8],1,Goto(smsmorx,${CALLERIDNUM}-{EXTEN:7:1},1)
```

12.7 Using smsq

smsq is a simple helper application designed to make it easy to send messages from a command line. it is intended to run on the Asterisk box and have direct access to the queue directories for SMS and for Asterisk.

In its simplest form you can send an SMS by a command such as smsq 0123456789 This is a test to 0123456789 This would create a queue file for a mobile originated TX message in queue 0 to send the text "This is a test to 0123456789" to 0123456789. It would then place a file in the /var/spool/asterisk/outgoing directory to initiate a call to 17094009 (the default message centre in smsq) attached to application SMS with argument of the queue name (0).

Normally smsq will queue a message ready to send, and will then create a file in the Asterisk outgoing directory causing Asterisk to actually connect to the message centre or device and actually send the pending message(s).

Using –process, smsq can however be used on received queues to run a command for each file (matching the queue if specified) with various environment variables set based on the message (see below); smsq options:-

```
--help
Show help text
--usage
Show usage
--queue
-q
Specify a specific queue
In no specified, messages are queued under queue "0"
--da
-d
Specify destination address
--oa
-o
Specify originating address
This also implies that we are generating a mobile terminated message
--ud
-m
Specify the actual message
--ud-file
```

```
-f
Specify a file to be read for the context of the message
A blank filename (e.g. --ud-file= on its own) means read stdin. Very
useful when using via ssh where command line parsing could mess up the
message.
--mt
-t
Mobile terminated message to be generated
--mo
Mobile originated message to be generated
Default
--tx
Transmit message
Default
--rx
-r
Generate a message in the receive queue
--UTF-8
Treat the file as UTF-8 encoded (default)
--UCS-1
Treat the file as raw 8 bit UCS-1 data, not UTF-8 encoded
--UCS-2
Treat the file as raw 16 bit bigendian USC-2 data
--process
Specific a command to process for each file in the queue
Implies --rx and --mt if not otherwise specified.
Sets environment variables for every possible variable, and also ud,
ud8 (USC-1 hex), and ud16 (USC-2 hex) for each call. Removes files.
--motx-channel
Specify the channel for motx calls
May contain X to use sub address based on queue name or may be full
number
Default is Local/1709400X
--motx-callerid
Specify the caller ID for motx calls
The default is the queue name without -X suffix
--motx-wait
Wait time for motx call
```

```
Default 10
--motx-delay
Retry time for motx call
Default 1
--motx-retries
Retries for motx call
Default 10
--mttx-channel
Specify the channel for mttx calls
Default is Local/ and the queue name without -X suffix
--mtttx-callerid
Specify the callerid for mttx calls
May include X to use sub address based on queue name or may be full
number
Default is 080058752X0
--mttx-wait
Wait time for mttx call
Default 10
--mttx-delay
Retry time for mttx call
Default 30
--mttx-retries
Retries for mttx call
Default 100
--default-sub-address
The default sub address assumed (e.g. for X in CLI and dialled numbers
as above) when none added (-X) to queue
Default 9
--no-dial
-x
Create queue, but do not dial to send message
--no-wait
Do not wait if a call appears to be in progress
This could have a small window where a message is queued but not
sent, so regular calls to smsq should be done to pick up any missed
messages
--concurrent
How many concurrent calls to allow (per queue), default 1
```

```
--mr
-n
Message reference
--pid
-p
Protocol ID
--dcs
Data coding scheme
--udh
Specific hex string of user data header specified (not including the
initial length byte)
May be a blank string to indicate header is included in the user data
already but user data header indication to be set.
--srr
Status report requested
--rp
Return path requested
--vp
Specify validity period (seconds)
--scts
Specify timestamp (YYYY-MM-DDTHH:MM:SS)
--spool-dir
Spool dir (in which sms and outgoing are found)
Default /var/spool/asterisk

Other arguments starting '-' or '--' are invalid and will cause an
error. Any trailing arguments are processed as follows:-
  * If the message is mobile originating and no destination address
    has been specified, then the first argument is assumed to be a
    destination address
  * If the message is mobile terminating and no destination address
    has been specified, then the first argument is assumed to be the
    queue name
  * If there is no user data, or user data file specified, then any
    following arguments are assumed to be the message, which are
    concatenated.
  * If no user data is specified, then no message is sent. However,
    unless --no-dial is specified, smsq checks for pending messages
```

```
and generates an outgoing anyway
```

Note that when smsq attempts to make a file in /var/spool/asterisk/outgoing, it checks if there is already a call queued for that queue. It will try several filenames, up to the –concurrent setting. If these files exist, then this means Asterisk is already queued to send all messages for that queue, and so Asterisk should pick up the message just queued. However, this alone could create a race condition, so if the files exist then smsq will wait up to 3 seconds to confirm it still exists or if the queued messages have been sent already. The –no-wait turns off this behaviour. Basically, this means that if you have a lot of messages to send all at once, Asterisk will not make unlimited concurrent calls to the same message centre or device for the same queue. This is because it is generally more efficient to make one call and send all of the messages one after the other.

smsq can be used with no arguments, or with a queue name only, and it will check for any pending messages and cause an outgoing if there are any. It only sets up one outgoing call at a time based on the first queued message it finds. A outgoing call will normally send all queued messages for that queue. One way to use smsq would be to run with no queue name (so any queue) every minute or every few seconds to send pending message. This is not normally necessary unless –no-dial is selected. Note that smsq does only check motx or mttx depending on the options selected, so it would need to be called twice as a general check.

UTF-8 is used to parse command line arguments for user data, and is the default when reading a file. If an invalid UTF-8 sequence is found, it is treated as UCS-1 data (i.e, as is). The –process option causes smsq to scan the specified queue (default is mtrx) for messages (matching the queue specified, or any if queue not specified) and run a command and delete the file. The command is run with a number of environment variables set as follows. Note that these are unset if not needed and not just taken from the calling environment. This allows simple processing of incoming messages

```
$queue
Set if a queue specified
$?srr
srr is set (to blank) if srr defined and has value 1.
$?rp
rp is set (to blank) if rp defined and has value 1.
$ud
```

```
User data, UTF-8 encoding, including any control characters, but with
nulls stripped out
Useful for the content of emails, for example, as it includes any
newlines, etc.
$ude
User data, escaped UTF-8, including all characters, but control
characters \n, \r, \t, \f, \xxx and \ is escaped as \\
Useful guaranteed one line printable text, so useful in Subject lines
of emails, etc
$ud8
Hex UCS-1 coding of user data (2 hex digits per character)
Present only if all user data is in range U+0000 to U+00FF
$ud16
Hex UCS-2 coding of user data (4 hex digits per character)
other
Other fields set using their field name, e.g. mr, pid, dcs, etc. udh
is a hex byte string
```

12.8 File formats

By default all queues are held in a director /var/spool/asterisk/sms. Within this directory are sub directories mtrx, mttx, morx, motx which hold the received messages and the messages ready to send. Also, /var/log/asterisk/sms is a log file of all messages handled.

The file name in each queue directory starts with the queue parameter to SMS which is normally the CLI used for an outgoing message or the called number on an incoming message, and may have -X (X being sub address) appended. If no queue ID is known, then 0 is used by smsq by default. After this is a dot, and then any text. Files are scanned for matching queue ID and a dot at the start. This means temporary files being created can be given a different name not starting with a queue (we recommend a . on the start of the file name for temp files). Files in these queues are in the form of a simple text file where each line starts with a keyword and an = and then data. udh and ud have options for hex encoding, see below.

UTF-8. The user data (ud) field is treated as being UTF-8 encoded unless the DCS is specified indicating 8 bit format. If 8 bit format is specified then the user data is sent as is. The keywords are as follows:-

```
oa Originating address
The phone number from which the message came
Present on mobile terminated messages and is the CLI for morx messages
da
Destination Address
The phone number to which the message is sent
Present on mobile originated messages
scts
The service centre time stamp
Format YYYY-MM-DDTHH:MM:SS
Present on mobile terminated messages
pid
One byte decimal protocol ID
See GSM specs for more details
Normally 0 or absent
dcs
One byte decimal data coding scheme
If omitted, a sensible default is used (see below)
See GSM specs for more details
mr
One byte decimal message reference
Present on mobile originated messages, added by default if absent
srr
0 or 1 for status report request
Does not work in UK yet, not implemented in app\_sms yet
rp
0 or 1 return path
See GSM specs for details
vp
Validity period in seconds
Does not work in UK yet
udh
Hex string of user data header prepended to the SMS contents,
excluding initial length byte.
Consistent with ud, this is specified as udh# rather than udh=
If blank, this means that the udhi flag will be set but any user data
header must be in the ud field
ud
```

```
User data, may be text, or hex, see below
```

udh is specified as as udh# followed by hex (2 hex digits per byte). If present, then the user data header indicator bit is set, and the length plus the user data header is added to the start of the user data, with padding if necessary (to septet boundary in 7 bit format). User data can hold an USC character codes U+0000 to U+FFFF. Any other characters are coded as U+FEFF

ud can be specified as ud= followed by UTF-8 encoded text if it contains no control characters, i.e. only (U+0020 to U+FFFF). Any invalid UTF-8 sequences are treated as is (U+0080-U+00FF).

ud can also be specified as ud# followed by hex (2 hex digits per byte) containing characters U+0000 to U+00FF only.

ud can also be specified as ud## followed by hex (4 hex digits per byte) containing UCS-2 characters.

When written by app_sms (e.g. incoming messages), the file is written with ud= if it can be (no control characters). If it cannot, the a comment line ;ud= is used to show the user data for human readability and ud# or ud## is used.

12.9 Delivery reports

The SMS specification allows for delivery reports. These are requested using the srr bit. However, as these do not work in the UK yet they are not fully implemented in this application. If anyone has a telco that does implement these, please let me know. BT in the UK have a non standard way to do this by starting the message with *0#, and so this application may have a UK specific bodge in the near future to handle these.

```
The main changes that are proposed for delivery report handling are :-
  * New queues for sent messages, one file for each destination
    address and message reference.
  * New field in message format, user reference, allowing applications
    to tie up their original message with a report.
  * Handling of the delivery confirmation/rejection and connecting to
    the outgoing message - the received message file would then have
    fields for the original outgoing message and user reference
    allowing applications to handle confirmations better.
```

Chapter 13

Queues

13.1 Introduction

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

13.2 Configuring Call Queues

13.2.1 queues.conf

First of all, set up call queues in queue.conf

Here is an example:

```
========== queues.conf ==========
| ; Cool Digium Queues           |
| [general]                      |
| persistentmembers = yes        |
|                                |
| ; General sales queue          |
| [sales-general]                |
| music=default                  |
| context=sales                  |
| strategy=ringall               |
```

```
| joinempty=strict                |
| leavewhenempty=strict           |
|                                 |
| ; Customer service queue        |
| [customerservice]               |
| music=default                   |
| context=customerservice         |
| strategy=ringall                |
| joinempty=strict                |
| leavewhenempty=strict           |
|                                 |
| ; Support dispatch queue        |
| [support-dispatch]              |
| music=default                   |
| context=dispatch                |
| strategy=ringall                |
| joinempty=strict                |
| leavewhenempty=strict           |
===================================
```

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

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

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

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

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

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

13.2.2 Routing incoming Calls to Queues

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

The Main Menu

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

```
context mainmenu {

includes {
digium;
queues-loginout;
}

        0 => goto dispatch|s|1;
        2 => goto sales|s|1;
        3 => goto customerservice|s|1;
        4 => goto dispatch|s|1;

        s => {
                Ringing();
                Wait(1);
                Set(attempts=0);
                Answer();
                Wait(1);
                Background(digium/ThankYouForCallingDigium);
                Background(digium/YourOpenSourceTelecommunicationsSupplier);
                WaitExten(0.3);
        repeat:
                Set(attempts=$[${attempts} + 1]);
                Background(digium/IfYouKnowYourPartysExtensionYouMayDialItAtAnyTi
                WaitExten(0.1);
                Background(digium/Otherwise);
                WaitExten(0.1);
                Background(digium/ForSalesPleasePress2);
                WaitExten(0.2);
                Background(digium/ForCustomerServicePleasePress3);
```

```
                WaitExten(0.2);
                Background(digium/ForAllOtherDepartmentsPleasePress4);
                WaitExten(0.2);
                Background(digium/ToSpeakWithAnOperatorPleasePress0AtAnyTime);
                if( ${attempts} < 2 ) {
                        WaitExten(0.3);
                        Background(digium/ToHearTheseOptionsRepeatedPleaseHold);
                }
                WaitExten(5);
                if( ${attempts} < 2 ) goto repeat;
                Background(digium/YouHaveMadeNoSelection);
                Background(digium/ThisCallWillBeEnded);
                Background(goodbye);
                Hangup();
        }
}
```

The Contexts referenced from the queues.conf file

```
context sales {

        0 => goto dispatch|s|1;
        8 => Voicemail(${SALESVM});

        s => {
                Ringing();
                Wait(2);
                Background(digium/ThankYouForContactingTheDigiumSalesDepartment);
                WaitExten(0.3);
                Background(digium/PleaseHoldAndYourCallWillBeAnsweredByOurNextAvai
                WaitExten(0.3);
                Background(digium/AtAnyTimeYouMayPress0ToSpeakWithAnOperatorOr8ToI
                Set(CALLERID(name)=Sales);
                Queue(sales-general|t);
                Set(CALLERID(name)=EmptySalQ);
                goto dispatch|s|1;
                Playback(goodbye);
                Hangup();
```

```
        }
}
```

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

```
context customerservice {

        0 => {
                SetCIDName(CSVTrans);
                goto dispatch|s|1;
        }
        8 => Voicemail(${CUSTSERVVM});

        s => {
                Ringing();
                Wait(2);
                Background(digium/ThankYouForCallingDigiumCustomerService);
                WaitExten(0.3);
        notracking:
                Background(digium/PleaseWaitForTheNextAvailableCustomerServiceRepr
                WaitExten(0.3);
                Background(digium/AtAnyTimeYouMayPress0ToSpeakWithAnOperatorOr8ToL
                Set(CALLERID(name)=Cust Svc);
                Set(QUEUE_MAX_PENALTY=10);
                Queue(customerservice|t);
                Set(QUEUE_MAX_PENALTY=0);
                Queue(customerservice|t);
                Set(CALLERID(name)=EmptyCSVQ);
                goto dispatch|s|1;
                Background(digium/NoCustomerServiceRepresentativesAreAvailableAtTh
                Background(digium/PleaseLeaveAMessageInTheCustomerServiceVoiceMail
                Voicemail(${CUSTSERVVM});
                Playback(goodbye);
                Hangup();
        }
}
```

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

```
context dispatch
{

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

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

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

First, you set QUEUE_MAX_PENALTY to a value, then you call Queue(¡queue-name¿,option,... (see the documentation for the Queue application));

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

The purpose of specifying the QUEUE_MAX_PENALTY is to develop a set of priorities amongst agents. By the above usage, agents with lower

number priorities will be given the calls first, and then, if no-one picks up the call, the QUEUE_MAX_PENALTY will be incremented, and the queue tried again. Hopefully, along the line, someone will pick up the call, and the Queue application will end with a hangup.

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

13.2.3 Assigning agents to Queues

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

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

Agents Log In and Out

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

        6093 => {
                        Answer();
                        Read(AGENT_NUMBER,agent-enternum);
                        Set(queue-announce-success=1);
                        goto queues-manip,O${AGENT_NUMBER},1;
                }
```

```
}
```

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

```
context queues-manip {

        // Raquel Squelch
        _[IO]6121 => {
                &queue-addremove(dispatch,10);
                &queue-success();
        }

        // Brittanica Spears
        _[IO]6165 => {
                &queue-addremove(dispatch,20);
                &queue-success();
        }

        // Rock Hudson
        _[IO]6170 => {
                &queue-addremove(sales-general,10);
                &queue-addremove(customerservice,20);
                &queue-addremove(dispatch,30);
                &queue-success();
        }

        // Saline Dye-on
        _[IO]6070 => {
                &queue-addremove(sales-general,20);
                &queue-addremove(customerservice,30);
                &queue-addremove(dispatch,30);
                &queue-success();
        }
}
```

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

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

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

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

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

The queue-addremove macro is defined in this manner:

```
macro queue-addremove(queuename,penalty)
{
        switch(${MACRO_EXTEN:0:1})
        {
        case I:  // Login
```

```
                {
                AddQueueMember(${queuename},Local/${MACRO_EXTEN:1}@agents,${penalt
                }
        case O:  // Logout
                {
                RemoveQueueMember(${queuename},Local/${MACRO_EXTEN:1}@agents);
                }
        case P:  // Pause
                {
                PauseQueueMember(${queuename},Local/${MACRO_EXTEN:1}@agents);
                }
        case U:  // Unpause
                {
                UnpauseQueueMember(${queuename},Local/${MACRO_EXTEN:1}@agents);
                }
        default: // Invalid
                {
                Playback(invalid);
                }
        }
}
```

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

13.2.4 Controlling The Way Queues Call the Agents

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

```
context agents
{
// General sales queue
8010 =>
{
Set(QUEUE_MAX_PENALTY=10);
Queue(sales-general|t);
```

```
Set(QUEUE_MAX_PENALTY=0);
Queue(sales-general|t);
Set(CALLERID(name)=EmptySalQ);
goto dispatch|s|1;
}
// Customer Service queue
8011 =>
{
Set(QUEUE_MAX_PENALTY=10);
Queue(customerservice|t);
Set(QUEUE_MAX_PENALTY=0);
Queue(customerservice|t);
Set(CALLERID(name)=EMptyCSVQ);
goto dispatch|s|1;
}
8013 =>
{
Dial(iax2/sweatshop/9456@from-ecstacy);

Set(CALLERID(name)=EmptySupQ);
Set(QUEUE_MAX_PENALTY=10);
Queue(support-dispatch,t);
Set(QUEUE_MAX_PENALTY=20);
Queue(support-dispatch,t);
Set(QUEUE_MAX_PENALTY=0); // means no max
Queue(support-dispatch,t);
goto dispatch|s|1;
}
6121 => &callagent(${RAQUEL});
6165 => &callagent(${SPEARS});
6170 => &callagent(${ROCK});
6070 => &callagent(${SALINE});
}
```

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

The 8010, 8011, and 8013 extensions are purely for transferring incoming callers to queues. For instance, a customer service agent might want to

transfer the caller to talk to sales. The agent only has to transfer to extension 8010, in this case.

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

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

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

The use of the GROUP_COUNT, and OUTBOUND_GROUP follow this line of thinking. Incoming calls can be queued to ring all agents in the current priority. If some of those agents are already talking, they would get bothersome call-waiting tones. To avoid this inconvenience, when an agent gets a call, the OUTBOUND_GROUP assigns that conversation to the group specified, for instance 6171@agents. The $GROUP_COUNT() variable on a

subsequent call should return "1" for that group. If GROUP_COUNT returns 1, then the busy() is returned without actually trying to dial the agent.

13.2.5 Pre Acknowledgement Message

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

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

13.2.6 Caveats

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

13.3 Queue Logs

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

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

ABANDON(position—origposition—waittime) The caller abandoned their position in the queue. The position is the caller's position in the queue when they hungup, the origposition is the original position the caller was when they first entered the queue, and the waittime is how long the call had been waiting in the queue at the time of disconnect.

AGENTDUMP The agent dumped the caller while listening to the queue announcement.

AGENTLOGIN(channel) The agent logged in. The channel is recorded.

AGENTCALLBACKLOGIN(exten@context) The callback agent logged in. The login extension and context is recorded.

AGENTLOGOFF(channel—logintime) The agent logged off. The channel is recorded, along with the total time the agent was logged in.

AGENTCALLBACKLOGOFF(exten@context—logintime—reason) The callback agent logged off. The last login extension and context is recorded, along with the total time the agent was logged in, and the reason for the logoff if it was not a normal logoff (e.g., Autologoff, Chanunavail)

COMPLETEAGENT(holdtime—calltime—origposition) The caller was connected to an agent, and the call was terminated normally by the *agent*. The caller's hold time and the length of the call are both recorded. The caller's original position in the queue is recorded in origposition.

COMPLETECALLER(holdtime—calltime—origposition) The caller was connected to an agent, and the call was terminated normally by the *caller*. The caller's hold time and the length of the call are both recorded. The caller's original position in the queue is recorded in origposition.

CONFIGRELOAD The configuration has been reloaded (e.g. with asterisk -rx reload)

CONNECT(holdtime—bridgedchanneluniqueid) The caller was connected to an agent. Hold time represents the amount of time the caller was on hold. The bridged channel unique ID contains the unique ID of the queue member channel that is taking the call. This is useful when trying to link recording filenames to a particular call in the queue.

ENTERQUEUE(url—callerid) A call has entered the queue. URL (if specified) and Caller*ID are placed in the log.

EXITEMPTY(position—origposition—waittime) The caller was exited from the queue forcefully because the queue had no reachable members and it's configured to do that to callers when there are no reachable members. The position is the caller's position in the queue when they hungup, the

origposition is the original position the caller was when they first entered the queue, and the waittime is how long the call had been waiting in the queue at the time of disconnect.

EXITWITHKEY(key—position) The caller elected to use a menu key to exit the queue. The key and the caller's position in the queue are recorded.

EXITWITHTIMEOUT(position) The caller was on hold too long and the timeout expired.

QUEUESTART The queueing system has been started for the first time this session.

RINGNOANSWER(ringtime) After trying for ringtime ms to connect to the available queue member, the attempt ended without the member picking up the call. Bad queue member!

SYSCOMPAT A call was answered by an agent, but the call was dropped because the channels were not compatible.

TRANSFER(extension—context—holdtime—calltime) Caller was transferred to a different extension. Context and extension are recorded. The caller's hold time and the length of the call are both recorded. PLEASE remember that transfers performed by SIP UA's by way of a reinvite may not always be caught by Asterisk and trigger off this event. The only way to be 100% sure that you will get this event when a transfer is performed by a queue member is to use the built-in transfer functionality of Asterisk.

Chapter 14

Application Reference

14.1 AddQueueMember

14.1.1 Synopsis

Dynamically adds queue members

14.1.2 Description

```
   AddQueueMember(queuename[|interface[|penalty[|options[|membername]]]]):
Dynamically adds interface to an existing queue.
If the interface is already in the queue and there exists an n+101 priority
then it will then jump to this priority.  Otherwise it will return an error
The option string may contain zero or more of the following characters:
       'j' -- jump to +101 priority when appropriate.
  This application sets the following channel variable upon completion:
     AQMSTATUS    The status of the attempt to add a queue member as a
                       text string, one of
           ADDED | MEMBERALREADY | NOSUCHQUEUE
Example: AddQueueMember(techsupport|SIP/3000)
```

14.2 ADSIProg

14.2.1 Synopsis

Load Asterisk ADSI Scripts into phone

14.2.2 Description

ADSIProg(script): This application programs an ADSI Phone with the given script. If nothing is specified, the default script (asterisk.adsi) is used.

14.3 AgentCallbackLogin

14.3.1 Synopsis

Call agent callback login

14.3.2 Description

AgentCallbackLogin([AgentNo][|[options][|[exten]@context]]):
Asks the agent to login to the system with callback.
The agent's callback extension is called (optionally with the specified context).
The option string may contain zero or more of the following characters:
's' -- silent login - do not announce the login ok segment agent logged in/o

14.4 AgentLogin

14.4.1 Synopsis

Call agent login

14.4.2 Description

AgentLogin([AgentNo][|options]):

```
Asks the agent to login to the system.  Always returns -1.  While
logged in, the agent can receive calls and will hear a 'beep'
when a new call comes in. The agent can dump the call by pressing
the star key.
The option string may contain zero or more of the following characters:
      's' -- silent login - do not announce the login ok segment after agent logge
```

14.5 AgentMonitorOutgoing

14.5.1 Synopsis

```
Record agent's outgoing call
```

14.5.2 Description

```
  AgentMonitorOutgoing([options]):
Tries to figure out the id of the agent who is placing outgoing call based on
comparison of the callerid of the current interface and the global variable
placed by the AgentCallbackLogin application. That's why it should be used only
with the AgentCallbackLogin app. Uses the monitoring functions in chan_agent
instead of Monitor application. That have to be configured in the agents.conf file

Return value:
Normally the app returns 0 unless the options are passed. Also if the callerid or
the agentid are not specified it'll look for n+101 priority.

Options:
'd' - make the app return -1 if there is an error condition and there is
      no extension n+101
'c' - change the CDR so that the source of the call is 'Agent/agent_id'
'n' - don't generate the warnings when there is no callerid or the
      agentid is not known.
             It's handy if you want to have one context for agent and non-agent ca
```

14.6 AGI

14.6.1 Synopsis

Executes an AGI compliant application

14.6.2 Description

[E|Dead]AGI(command|args): Executes an Asterisk Gateway Interface compliant program on a channel. AGI allows Asterisk to launch external programs written in any language to control a telephony channel, play audio, read DTMF digits, etc. by communicating with the AGI protocol on stdin and stdout.

This channel will stop dialplan execution on hangup inside of this application, except when using DeadAGI. Otherwise, dialplan execution will continue normally.

A locally executed AGI script will receive SIGHUP on hangup from the channel except when using DeadAGI. This can be disabled by setting the AGISIGHUP channel variable to "no" before executing the AGI application.

Using 'EAGI' provides enhanced AGI, with incoming audio available out of band on file descriptor 3

Use the CLI command 'agi show' to list available agi commands
This application sets the following channel variable upon completion:
AGISTATUS The status of the attempt to the run the AGI script
text string, one of SUCCESS | FAILED | HANGUP

14.7 AlarmReceiver

14.7.1 Synopsis

Provide support for receiving alarm reports from a burglar or fire alarm panel

14.7.2 Description

AlarmReceiver(): Only 1 signalling format is supported at this time: Ademco Contact ID. This application should be called whenever there is an alarm

panel calling in to dump its events. The application will handshake with the alarm panel, and receive events, validate them, handshake them, and store them until the panel hangs up. Once the panel hangs up, the application will run the system command specified by the eventcmd setting in alarmreceiver.conf and pipe the events to the standard input of the application. The configuration file also contains settings for DTMF timing, and for the loudness of the acknowledgement tones.

14.8 AMD

14.8.1 Synopsis

Attempts to detect answering machines

14.8.2 Description

```
  AMD([initialSilence][|greeting][|afterGreetingSilence][|totalAnalysisTime]
      [|minimumWordLength][|betweenWordsSilence][|maximumNumberOfWords]
      [|silenceThreshold])
  This application attempts to detect answering machines at the beginning
  of outbound calls.  Simply call this application after the call
  has been answered (outbound only, of course).
  When loaded, AMD reads amd.conf and uses the parameters specified as
  default values. Those default values get overwritten when calling AMD
  with parameters.
- 'initialSilence' is the maximum silence duration before the greeting. If
   exceeded then MACHINE.
- 'greeting' is the maximum length of a greeting. If exceeded then MACHINE.
- 'afterGreetingSilence' is the silence after detecting a greeting.
   If exceeded then HUMAN.
- 'totalAnalysisTime' is the maximum time allowed for the algorithm to decide
   on a HUMAN or MACHINE.
- 'minimumWordLength'is the minimum duration of Voice to considered as a word.
- 'betweenWordsSilence' is the minimum duration of silence after a word to
   consider the audio that follows as a new word.
- 'maximumNumberOfWords'is the maximum number of words in the greeting.
```

```
   If exceeded then MACHINE.
- 'silenceThreshold' is the silence threshold.
This application sets the following channel variable upon completion:
    AMDSTATUS - This is the status of the answering machine detection.
                Possible values are:
                MACHINE | HUMAN | NOTSURE | HANGUP
    AMDCAUSE - Indicates the cause that led to the conclusion.
               Possible values are:
               TOOLONG-<%d total_time>
               INITIALSILENCE-<%d silenceDuration>-<%d initialSilence>
               HUMAN-<%d silenceDuration>-<%d afterGreetingSilence>
               MAXWORDS-<%d wordsCount>-<%d maximumNumberOfWords>
               LONGGREETING-<%d voiceDuration>-<%d greeting>
```

14.9 Answer

14.9.1 Synopsis

Answer a channel if ringing

14.9.2 Description

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

14.10 AppendCDRUserField

14.10.1 Synopsis

Append to the CDR user field

14.10.2 Description

```
[Synopsis]
AppendCDRUserField(value)

[Description]
AppendCDRUserField(value): Append value to the CDR user field
       The Call Data Record (CDR) user field is an extra field you
       can use for data not stored anywhere else in the record.
       CDR records can be used for billing or storing other arbitrary data
       (I.E. telephone survey responses)
       Also see SetCDRUserField().

This application is deprecated in favor of Set(CDR(userfield)=...)
```

14.11 Authenticate

14.11.1 Synopsis

```
Authenticate a user
```

14.11.2 Description

```
  Authenticate(password[|options[|maxdigits]]): This application asks the caller
to enter a given password in order to continue dialplan execution. If the passwor
begins with the '/' character, it is interpreted as a file which contains a list o
valid passwords, listed 1 password per line in the file.
  When using a database key, the value associated with the key can be anything.
Users have three attempts to authenticate before the channel is hung up. If the
passsword is invalid, the 'j' option is specified, and priority n+101 exists,
dialplan execution will continnue at this location.
  Options:
     a - Set the channels' account code to the password that is entered
     d - Interpret the given path as database key, not a literal file
     j - Support jumping to n+101 if authentication fails
     m - Interpret the given path as a file which contains a list of account
         codes and password hashes delimited with ':', listed one per line in
```

```
        the file. When one of the passwords is matched, the channel will have
        its account code set to the corresponding account code in the file.
    r - Remove the database key upon successful entry (valid with 'd' only)
    maxdigits  - maximum acceptable number of digits. Stops reading after
        maxdigits have been entered (without requiring the user to
        press the '#' key).
        Defaults to 0 - no limit - wait for the user press the '#' key.
```

14.12 BackGround

14.12.1 Synopsis

Play an audio file while waiting for digits of an extension to go to.

14.12.2 Description

```
  Background(filename1[&filename2...][|options[|langoverride][|context]]):
This application will play the given list of files while waiting for an
extension to be dialed by the calling channel. To continue waiting for digits
after this application has finished playing files, the WaitExten application
should be used. The 'langoverride' option explicitly specifies which language
to attempt to use for the requested sound files. If a 'context' is specified,
this is the dialplan context that this application will use when exiting to a
dialed extension.  If one of the requested sound files does not exist, call proces
terminated.
  Options:
    s - Causes the playback of the message to be skipped
          if the channel is not in the 'up' state (i.e. it
          hasn't been answered yet). If this happens, the
          application will return immediately.
    n - Don't answer the channel before playing the files.
    m - Only break if a digit hit matches a one digit
          extension in the destination context.
```

14.13 BackgroundDetect

14.13.1 Synopsis

Background a file with talk detect

14.13.2 Description

BackgroundDetect(filename[|sil[|min|[max]]]): Plays back a given filename, waiting for interruption from a given digit (the digit must start the beginning of a valid extension, or it will be ignored). During the playback of the file, audio is monitored in the receive direction, and if a period of non-silence which is greater than 'min' ms yet less than 'max' ms is followed by silence for at least 'sil' ms then the audio playback is aborted and processing jumps to the 'talk' extension if available. If unspecified, sil, min, and max default to 1000, 100, and infinity respectively.

14.14 Busy

14.14.1 Synopsis

Indicate the Busy condition

14.14.2 Description

Busy([timeout]): This application will indicate the busy condition to the calling channel. If the optional timeout is specified, the calling channel will be hung up after the specified number of seconds. Otherwise, this application will wait until the calling channel hangs up.

14.15 ChangeMonitor

14.15.1 Synopsis

Change monitoring filename of a channel

14.15.2 Description

```
ChangeMonitor(filename_base)
Changes monitoring filename of a channel. Has no effect if the channel is not moni
The argument is the new filename base to use for monitoring this channel.
```

14.16 ChanIsAvail

14.16.1 Synopsis

```
Check channel availability
```

14.16.2 Description

```
  ChanIsAvail(Technology/resource[&Technology2/resource2...][|options]):
This application will check to see if any of the specified channels are
available. The following variables will be set by this application:
  ${AVAILCHAN}     - the name of the available channel, if one exists
  ${AVAILORIGCHAN} - the canonical channel name that was used to create the channe
  ${AVAILSTATUS}   - the status code for the available channel
  Options:
    s - Consider the channel unavailable if the channel is in use at all
    j - Support jumping to priority n+101 if no channel is available
```

14.17 ChannelRedirect

14.17.1 Synopsis

```
Redirects given channel to a dialplan target.
```

14.17.2 Description

```
ChannelRedirect(channel|[[context|]extension|]priority):
  Sends the specified channel to the specified extension priority
```

14.18 ChanSpy

14.18.1 Synopsis

Listen to a channel, and optionally whisper into it

14.18.2 Description

```
  ChanSpy([chanprefix][|options]): This application is used to listen to the
audio from an Asterisk channel. This includes the audio coming in and
out of the channel being spied on. If the 'chanprefix' parameter is specified,
only channels beginning with this string will be spied upon.
  While spying, the following actions may be performed:
    - Dialing # cycles the volume level.
    - Dialing * will stop spying and look for another channel to spy on.
    - Dialing a series of digits followed by # builds a channel name to append
      to 'chanprefix'. For example, executing ChanSpy(Agent) and then dialing
      the digits '1234#' while spying will begin spying on the channel
      'Agent/1234'.
  Options:
    b             - Only spy on channels involved in a bridged call.
    g(grp)        - Match only channels where their ${SPYGROUP} variable is set to
                    contain 'grp' in an optional : delimited list.
    q             - Don't play a beep when beginning to spy on a channel, or speak
                    selected channel name.
    r[(basename)] - Record the session to the monitor spool directory. An
                    optional base for the filename may be specified. The
                    default is 'chanspy'.
    v([value])    - Adjust the initial volume in the range from -4 to 4. A
                    negative value refers to a quieter setting.
    w             - Enable 'whisper' mode, so the spying channel can talk to
                    the spied-on channel.
    W             - Enable 'private whisper' mode, so the spying channel can
                    talk to the spied-on channel but cannot listen to that
                    channel.
```

14.19 Congestion

14.19.1 Synopsis

Indicate the Congestion condition

14.19.2 Description

Congestion([timeout]): This application will indicate the congestion condition to the calling channel. If the optional timeout is specified, the calling channel will be hung up after the specified number of seconds. Otherwise, this application will wait until the calling channel hangs up.

14.20 ContinueWhile

14.20.1 Synopsis

Restart a While loop

14.20.2 Description

Usage: ContinueWhile()
Returns to the top of the while loop and re-evaluates the conditional.

14.21 ControlPlayback

14.21.1 Synopsis

Play a file with fast forward and rewind

14.21.2 Description

ControlPlayback(file[|skipms[|ff[|rew[|stop[|pause[|restart|options]]]]]]): This application will play back the given filename. By default, the '*' key can be used to rewind, and the '#' key can be used to fast-forward.

```
Parameters:
  skipms  - This is number of milliseconds to skip when rewinding or
            fast-forwarding.
  ff      - Fast-forward when this DTMF digit is received.
  rew     - Rewind when this DTMF digit is received.
  stop    - Stop playback when this DTMF digit is received.
  pause   - Pause playback when this DTMF digit is received.
  restart - Restart playback when this DTMF digit is received.
Options:
  j - Jump to priority n+101 if the requested file is not found.
This application sets the following channel variable upon completion:
  CPLAYBACKSTATUS -  This variable contains the status of the attempt as a text
                     string, one of: SUCCESS | USERSTOPPED | ERROR
```

14.22 DateTime

14.22.1 Synopsis

```
Says a specified time in a custom format
```

14.22.2 Description

```
DateTime([unixtime][|[timezone][|format]])
  unixtime: time, in seconds since Jan 1, 1970.  May be negative.
              defaults to now.
  timezone: timezone, see /usr/share/zoneinfo for a list.
              defaults to machine default.
  format:   a format the time is to be said in.  See voicemail.conf.
              defaults to "ABdY 'digits/at' IMp"
```

14.23 DBdel

14.23.1 Synopsis

```
Delete a key from the database
```

14.23.2 Description

DBdel(family/key): This application will delete a key from the Asterisk database.
This application has been DEPRECATED in favor of the DB_DELETE function.

14.24 DBdeltree

14.24.1 Synopsis

Delete a family or keytree from the database

14.24.2 Description

DBdeltree(family[/keytree]): This application will delete a family or keytree from the Asterisk database

14.25 DeadAGI

14.25.1 Synopsis

Executes AGI on a hungup channel

14.25.2 Description

[E|Dead]AGI(command|args): Executes an Asterisk Gateway Interface compliant program on a channel. AGI allows Asterisk to launch external programs written in any language to control a telephony channel, play audio, read DTMF digits, etc. by communicating with the AGI protocol on stdin and stdout.
This channel will stop dialplan execution on hangup inside of this application, except when using DeadAGI. Otherwise, dialplan execution will continue normally.
A locally executed AGI script will receive SIGHUP on hangup from the channel except when using DeadAGI. This can be disabled by setting the AGISIGHUP channel

```
variable to "no" before executing the AGI application.
  Using 'EAGI' provides enhanced AGI, with incoming audio available out of band
on file descriptor 3

  Use the CLI command 'agi show' to list available agi commands
  This application sets the following channel variable upon completion:
     AGISTATUS      The status of the attempt to the run the AGI script
                    text string, one of SUCCESS | FAILED | HANGUP
```

14.26 Dial

14.26.1 Synopsis

```
Place a call and connect to the current channel
```

14.26.2 Description

```
  Dial(Technology/resource[&Tech2/resource2...][|timeout][|options][|URL]):
This application will place calls to one or more specified channels. As soon
as one of the requested channels answers, the originating channel will be
answered, if it has not already been answered. These two channels will then
be active in a bridged call. All other channels that were requested will then
be hung up.
  Unless there is a timeout specified, the Dial application will wait
indefinitely until one of the called channels answers, the user hangs up, or
if all of the called channels are busy or unavailable. Dialplan executing will
continue if no requested channels can be called, or if the timeout expires.

  This application sets the following channel variables upon completion:
    DIALEDTIME   - This is the time from dialing a channel until when it
                   is disconnected.
    ANSWEREDTIME - This is the amount of time for actual call.
    DIALSTATUS   - This is the status of the call:
                   CHANUNAVAIL | CONGESTION | NOANSWER | BUSY | ANSWER | CANCEL
                   DONTCALL | TORTURE | INVALIDARGS
  For the Privacy and Screening Modes, the DIALSTATUS variable will be set to
```

```
DONTCALL if the called party chooses to send the calling party to the 'Go Away'
script. The DIALSTATUS variable will be set to TORTURE if the called party
wants to send the caller to the 'torture' script.
  This application will report normal termination if the originating channel
hangs up, or if the call is bridged and either of the parties in the bridge
ends the call.
  The optional URL will be sent to the called party if the channel supports it.
  If the OUTBOUND_GROUP variable is set, all peer channels created by this
application will be put into that group (as in Set(GROUP()=...).

  Options:
    A(x) - Play an announcement to the called party, using 'x' as the file.
    C    - Reset the CDR for this call.
    d    - Allow the calling user to dial a 1 digit extension while waiting for
           a call to be answered. Exit to that extension if it exists in the
           current context, or the context defined in the EXITCONTEXT variable,
           if it exists.
    D([called][:calling]) - Send the specified DTMF strings *after* the called
           party has answered, but before the call gets bridged. The 'called'
           DTMF string is sent to the called party, and the 'calling' DTMF
           string is sent to the calling party. Both parameters can be used
           alone.
    f    - Force the callerid of the *calling* channel to be set as the
           extension associated with the channel using a dialplan 'hint'.
           For example, some PSTNs do not allow CallerID to be set to anything
           other than the number assigned to the caller.
    g    - Proceed with dialplan execution at the current extension if the
           destination channel hangs up.
    G(context^exten^pri) - If the call is answered, transfer the calling party to
           the specified priority and the called party to the specified priority+1
           Optionally, an extension, or extension and context may be specified.
           Otherwise, the current extension is used. You cannot use any additional
           action post answer options in conjunction with this option.
    h    - Allow the called party to hang up by sending the '*' DTMF digit.
    H    - Allow the calling party to hang up by hitting the '*' DTMF digit.
    i    - Asterisk will ignore any forwarding requests it may receive on this
           dial attempt.
    j    - Jump to priority n+101 if all of the requested channels were busy.
```

```
L(x[:y][:z]) - Limit the call to 'x' ms. Play a warning when 'y' ms are
       left. Repeat the warning every 'z' ms. The following special
       variables can be used with this option:
       * LIMIT_PLAYAUDIO_CALLER   yes|no (default yes)
                                  Play sounds to the caller.
       * LIMIT_PLAYAUDIO_CALLEE   yes|no
                                  Play sounds to the callee.
       * LIMIT_TIMEOUT_FILE       File to play when time is up.
       * LIMIT_CONNECT_FILE       File to play when call begins.
       * LIMIT_WARNING_FILE       File to play as warning if 'y' is defined.
                                  The default is to say the time remaining.
m([class]) - Provide hold music to the calling party until a requested
       channel answers. A specific MusicOnHold class can be
       specified.
M(x[^arg]) - Execute the Macro for the *called* channel before connecting
       to the calling channel. Arguments can be specified to the Macro
       using '^' as a delimeter. The Macro can set the variable
       MACRO_RESULT to specify the following actions after the Macro is
       finished executing.
       * ABORT        Hangup both legs of the call.
       * CONGESTION   Behave as if line congestion was encountered.
       * BUSY         Behave as if a busy signal was encountered. This will al
                      have the application jump to priority n+101 if the
                      'j' option is set.
       * CONTINUE     Hangup the called party and allow the calling party
                      to continue dialplan execution at the next priority.
       * GOTO:<context>^<exten>^<priority> - Transfer the call to the
                      specified priority. Optionally, an extension, or
                      extension and priority can be specified.
       You cannot use any additional action post answer options in conjunctior
       with this option. Also, pbx services are not run on the peer (called) c
       so you will not be able to set timeouts via the TIMEOUT() function in t
n    - This option is a modifier for the screen/privacy mode. It specifies
       that no introductions are to be saved in the priv-callerintros
       directory.
N    - This option is a modifier for the screen/privacy mode. It specifies
       that if callerID is present, do not screen the call.
o    - Specify that the CallerID that was present on the *calling* channel
```

be set as the CallerID on the *called* channel. This was the behavior of Asterisk 1.0 and earlier.

O([x]) - "Operator Services" mode (Zaptel channel to Zaptel channel only, if specified on non-Zaptel interface, it will be ignored). When the destination answers (presumably an operator services station), the originator no longer has control of their line. They may hang up, but the switch will not release their line until the destination party hangs up (the operator). Specified without an arg, or with 1 as an arg, the originator hanging up will cause the phone to ring back immediately. With a 2 specified, when the "operator" flashes the trunk, it will ring their phone back.

p - This option enables screening mode. This is basically Privacy mode without memory.

P([x]) - Enable privacy mode. Use 'x' as the family/key in the database if it is provided. The current extension is used if a database family/key is not specified.

r - Indicate ringing to the calling party. Pass no audio to the calling party until the called channel has answered.

S(x) - Hang up the call after 'x' seconds *after* the called party has answered the call.

t - Allow the called party to transfer the calling party by sending the DTMF sequence defined in features.conf.

T - Allow the calling party to transfer the called party by sending the DTMF sequence defined in features.conf.

w - Allow the called party to enable recording of the call by sending the DTMF sequence defined for one-touch recording in features.conf.

W - Allow the calling party to enable recording of the call by sending the DTMF sequence defined for one-touch recording in features.conf.

k - Allow the called party to enable parking of the call by sending the DTMF sequence defined for call parking in features.conf.

K - Allow the calling party to enable parking of the call by sending the DTMF sequence defined for call parking in features.conf.

14.27 Dictate

14.27.1 Synopsis

Virtual Dictation Machine

14.27.2 Description

```
  Dictate([<base_dir>[|<filename>]])
Start dictation machine using optional base dir for files.
```

14.28 Directory

14.28.1 Synopsis

Provide directory of voicemail extensions

14.28.2 Description

```
  Directory(vm-context[|dial-context[|options]]): This application will present
the calling channel with a directory of extensions from which they can search
by name. The list of names and corresponding extensions is retrieved from the
voicemail configuration file, voicemail.conf.
  This application will immediately exit if one of the following DTMF digits are
received and the extension to jump to exists:
    0 - Jump to the 'o' extension, if it exists.
    * - Jump to the 'a' extension, if it exists.

  Parameters:
    vm-context   - This is the context within voicemail.conf to use for the
                   Directory.
    dial-context - This is the dialplan context to use when looking for an
                   extension that the user has selected, or when jumping to the
                   'o' or 'a' extension.

  Options:
    e - In addition to the name, also read the extension number to the
```

caller before presenting dialing options.
f - Allow the caller to enter the first name of a user in the directory
instead of using the last name.

14.29 DISA

14.29.1 Synopsis

DISA (Direct Inward System Access)

14.29.2 Description

DISA(<numeric passcode>[|<context>]) or DISA(<filename>)
The DISA, Direct Inward System Access, application allows someone from outside the telephone switch (PBX) to obtain an "internal" system dialtone and to place calls from it as if they were placing a call from within the switch.
DISA plays a dialtone. The user enters their numeric passcode, followed by the pound sign (#). If the passcode is correct, the user is then given system dialtone on which a call may be placed. Obviously, this type of access has SERIOUS security implications, and GREAT care must be taken NOT to compromise your security.

There is a possibility of accessing DISA without password. Simply exchange your password with "no-password".

Example: exten => s,1,DISA(no-password|local)

Be aware that using this compromises the security of your PBX.

The arguments to this application (in extensions.conf) allow either specification of a single global passcode (that everyone uses), or individual passcodes contained in a file. It also allows specification of the context on which the user will be dialing. If no context is specified, the DISA application defaults the context to "disa". Presumably a normal system will have a special context set up

for DISA use with some or a lot of restrictions.

The file that contains the passcodes (if used) allows specification of either just a passcode (defaulting to the "disa" context, or passcode|context on each line of the file. The file may contain blank lines, or comments starting with "#" or ";". In addition, the above arguments may have |new-callerid-string appended to them, to specify a new (different) callerid to be used for this call, for example: numeric-passcode|context|"My Phone" <(234) 123-4567> or full-pathname-of-passcode-file|"My Phone" <(234) 123-4567>. Last but not least, |mailbox[@context] may be appended, which will cause a stutter-dialtone (indication "dialrecall") to be used, if the specified mailbox contains any new messages, for example: numeric-passcode|context||1234 (w/a changing callerid). Note that in the case of specifying the numeric-passcode, the context must be specified if the callerid is specified also.

If login is successful, the application looks up the dialed number in the specified (or default) context, and executes it if found.
If the user enters an invalid extension and extension "i" (invalid) exists in the context, it will be used. Also, if you set the 5th argument to 'NOANSWER', the DISA application will not answer initially.

14.30 DumpChan

14.30.1 Synopsis

Dump Info About The Calling Channel

14.30.2 Description

DumpChan([<min_verbose_level>])
Displays information on channel and listing of all channel variables. If min_verbose_level is specified, output is only displayed when the verbose level is currently set to that number or greater.

14.31 EAGI

14.31.1 Synopsis

Executes an EAGI compliant application

14.31.2 Description

```
  [E|Dead]AGI(command|args): Executes an Asterisk Gateway Interface compliant
program on a channel. AGI allows Asterisk to launch external programs
written in any language to control a telephony channel, play audio,
read DTMF digits, etc. by communicating with the AGI protocol on stdin
and stdout.
  This channel will stop dialplan execution on hangup inside of this
application, except when using DeadAGI.  Otherwise, dialplan execution
will continue normally.
  A locally executed AGI script will receive SIGHUP on hangup from the channel
except when using DeadAGI. This can be disabled by setting the AGISIGHUP channel
variable to "no" before executing the AGI application.
  Using 'EAGI' provides enhanced AGI, with incoming audio available out of band
on file descriptor 3

  Use the CLI command 'agi show' to list available agi commands
  This application sets the following channel variable upon completion:
     AGISTATUS      The status of the attempt to the run the AGI script
                    text string, one of SUCCESS | FAILED | HANGUP
```

14.32 Echo

14.32.1 Synopsis

Echo audio, video, or DTMF back to the calling party

14.32.2 Description

Echo(): This application will echo any audio, video, or DTMF frames read from the calling channel back to itself. If the DTMF digit '#' is received, the application will exit.

14.33 EndWhile

14.33.1 Synopsis

End a while loop

14.33.2 Description

Usage: EndWhile()
Return to the previous called While

14.34 Exec

14.34.1 Synopsis

Executes dialplan application

14.34.2 Description

Usage: Exec(appname(arguments))
Allows an arbitrary application to be invoked even when not hardcoded into the dialplan. If the underlying application terminates the dialplan, or if the application cannot be found, Exec will terminate the dialplan.
To invoke external applications, see the application System.
If you would like to catch any error instead, see TryExec.

14.35 ExecIf

14.35.1 Synopsis

Executes dialplan application, conditionally

14.35.2 Description

```
Usage:  ExecIF (<expr>|<app>|<data>)
If <expr> is true, execute and return the result of <app>(<data>).
If <expr> is true, but <app> is not found, then the application
will return a non-zero value.
```

14.36 ExecIfTime

14.36.1 Synopsis

Conditional application execution based on the current time

14.36.2 Description

```
  ExecIfTime(<times>|<weekdays>|<mdays>|<months>?appname[|appargs]):
This application will execute the specified dialplan application, with optional
arguments, if the current time matches the given time specification.
```

14.37 ExitWhile

14.37.1 Synopsis

End a While loop

14.37.2 Description

```
Usage:  ExitWhile()
Exits a While loop, whether or not the conditional has been satisfied.
```

14.38 ExtenSpy

14.38.1 Synopsis

Listen to a channel, and optionally whisper into it

14.38.2 Description

```
  ExtenSpy(exten[@context][|options]): This application is used to listen to the
audio from an Asterisk channel. This includes the audio coming in and
out of the channel being spied on. Only channels created by outgoing calls for the
specified extension will be selected for spying. If the optional context is not
supplied, the current channel's context will be used.
  While spying, the following actions may be performed:
    - Dialing # cycles the volume level.
    - Dialing * will stop spying and look for another channel to spy on.
  Options:
    b             - Only spy on channels involved in a bridged call.
    g(grp)        - Match only channels where their ${SPYGROUP} variable is set to
                    contain 'grp' in an optional : delimited list.
    q             - Don't play a beep when beginning to spy on a channel, or speak
                    selected channel name.
    r[(basename)] - Record the session to the monitor spool directory. An
                    optional base for the filename may be specified. The
                    default is 'chanspy'.
    v([value])    - Adjust the initial volume in the range from -4 to 4. A
                    negative value refers to a quieter setting.
    w             - Enable 'whisper' mode, so the spying channel can talk to
                    the spied-on channel.
    W             - Enable 'private whisper' mode, so the spying channel can
                    talk to the spied-on channel but cannot listen to that
                    channel.
```

14.39 ExternalIVR

14.39.1 Synopsis

Interfaces with an external IVR application

14.39.2 Description

ExternalIVR(command[|arg[|arg...]]): Forks an process to run the supplied commar
and starts a generator on the channel. The generator's play list is
controlled by the external application, which can add and clear entries
via simple commands issued over its stdout. The external application
will receive all DTMF events received on the channel, and notification
if the channel is hung up. The application will not be forcibly terminated
when the channel is hung up.
See doc/externalivr.txt for a protocol specification.

14.40 Festival

14.40.1 Synopsis

Say text to the user

14.40.2 Description

Festival(text[|intkeys]): Connect to Festival, send the argument, get back the
the value, or 'any' to allow any number back (useful in dialplan)

14.41 Flash

14.41.1 Synopsis

Flashes a Zap Trunk

14.41.2 Description

```
  Flash(): Sends a flash on a zap trunk.  This is only a hack for
people who want to perform transfers and such via AGI and is generally
quite useless oths application will only work on Zap trunks.
```

14.42 FollowMe

14.42.1 Synopsis

```
Find-Me/Follow-Me application
```

14.42.2 Description

```
  FollowMe(followmeid|options):
This application performs Find-Me/Follow-Me functionality for the caller
as defined in the profile matching the <followmeid> parameter in
followme.conf. If the specified <followmeid> profile doesn't exist in
followme.conf, execution will be returned to the dialplan and call
execution will continue at the next priority.

  Options:
    s    - Playback the incoming status message prior to starting the follow-me st
    a    - Record the caller's name so it can be announced to the callee on each s
    n    - Playback the unreachable status message if we've run out of steps to re
           or the callee has elected not to be reachable.
Returns -1 on hangup
```

14.43 ForkCDR

14.43.1 Synopsis

```
Forks the Call Data Record
```

14.43.2 Description

ForkCDR([options]): Causes the Call Data Record to fork an additional cdr record starting from the time of the fork call
If the option 'v' is passed all cdr variables will be passed along also.

14.44 GetCPEID

14.44.1 Synopsis

Get ADSI CPE ID

14.44.2 Description

GetCPEID: Obtains and displays ADSI CPE ID and other information in order to properly setup zapata.conf for on-hook operations.

14.45 Gosub

14.45.1 Synopsis

Jump to label, saving return address

14.45.2 Description

Gosub([[context|]exten|]priority)
Jumps to the label specified, saving the return address.

14.46 GosubIf

14.46.1 Synopsis

Conditionally jump to label, saving return address

14.46.2 Description

GosubIf(condition?labeliftrue[:labeliffalse])
  If the condition is true, then jump to labeliftrue. If false, jumps to labeliffalse, if specified. In either case, a jump saves the return point in the dialplan, to be returned to with a Return.

14.47 Goto

14.47.1 Synopsis

Jump to a particular priority, extension, or context

14.47.2 Description

  Goto([[context|]extension|]priority): This application will set the current context, extension, and priority in the channel structure. After it completes, the pbx engine will continue dialplan execution at the specified location.
If no specific extension, or extension and context, are specified, then this application will just set the specified priority of the current extension.
  At least a priority is required as an argument, or the goto will return a -1, and the channel and call will be terminated.
  If the location that is put into the channel information is bogus, and asterisk find that location in the dialplan,
then the execution engine will try to find and execute the code in the 'i' (invali extension in the current context. If that does not exist, it will try to execute t 'h' extension. If either or neither the 'h' or 'i' extensions have been defined, t channel is hung up, and the execution of instructions on the channel is terminated What this means is that, for example, you specify a context that does not exist, t it will not be possible to find the 'h' or 'i' extensions, and the call will termi

14.48 GotoIf

14.48.1 Synopsis

Conditional goto

14.48.2 Description

GotoIf(condition?[labeliftrue]:[labeliffalse]): This application will set the cu
context, extension, and priority in the channel structure based on the evaluation
the given condition. After this application completes, the
pbx engine will continue dialplan execution at the specified location in the dialp
The channel will continue at
'labeliftrue' if the condition is true, or 'labeliffalse' if the condition is
false. The labels are specified with the same syntax as used within the Goto
application. If the label chosen by the condition is omitted, no jump is
performed, and the execution passes to the next instruction.
If the target location is bogus, and does not exist, the execution engine will try
to find and execute the code in the 'i' (invalid)
extension in the current context. If that does not exist, it will try to execute t
'h' extension. If either or neither the 'h' or 'i' extensions have been defined, t
channel is hung up, and the execution of instructions on the channel is terminated
Remember that this command can set the current context, and if the context specifi
does not exist, then it will not be able to find any 'h' or 'i' extensions there,
the channel and call will both be terminated!

14.49 GotoIfTime

14.49.1 Synopsis

Conditional Goto based on the current time

14.49.2 Description

GotoIfTime(<times>|<weekdays>|<mdays>|<months>?[[context|]exten|]priority):
This application will set the context, extension, and priority in the channel stru
if the current time matches the given time specification. Otherwise, nothing is do
Further information on the time specification can be found in examples
illustrating how to do time-based context includes in the dialplan.
If the target jump location is bogus, the same actions would be taken as for Goto.

14.50 Hangup

14.50.1 Synopsis

Hang up the calling channel

14.50.2 Description

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

14.51 HasNewVoicemail

14.51.1 Synopsis

Conditionally branches to priority + 101 with the right options set

14.51.2 Description

HasNewVoicemail(vmbox[/folder][@context][|varname[|options]])
Assumes folder 'INBOX' if folder is not specified. Optionally sets <varname> to th
in that folder.
  The option string may contain zero of the following character:
'j' -- jump to priority n+101, if there is new voicemail in folder 'folder' or INB
  This application sets the following channel variable upon completion:
HASVMSTATUS The result of the new voicemail check returned as a text string as fol
<# of messages in the folder, 0 for NONE>

This application has been deprecated in favor of the VMCOUNT() function

14.52 HasVoicemail

14.52.1 Synopsis

Conditionally branches to priority + 101 with the right options set

14.52.2 Description

```
HasVoicemail(vmbox[/folder][@context][|varname[|options]])
  Optionally sets <varname> to the number of messages in that folder.  Assumes fol
  The option string may contain zero or the following character:
'j' -- jump to priority n+101, if there is voicemail in the folder indicated.
  This application sets the following channel variable upon completion:
HASVMSTATUS The result of the voicemail check returned as a text string as follows
<# of messages in the folder, 0 for NONE>

This application has been deprecated in favor of the VMCOUNT() function
```

14.53 IAX2Provision

14.53.1 Synopsis

```
Provision a calling IAXy with a given template
```

14.53.2 Description

```
  IAX2Provision([template]): Provisions the calling IAXy (assuming
the calling entity is in fact an IAXy) with the given template or
default if one is not specified.  Returns -1 on error or 0 on success.
```

14.54 ICES

14.54.1 Synopsis

```
Encode and stream using 'ices'
```

14.54.2 Description

```
  ICES(config.xml) Streams to an icecast server using ices
(available separately).  A configuration file must be supplied
for ices (see examples/asterisk-ices.conf).
```

14.55 ImportVar

14.55.1 Synopsis

Import a variable from a channel into a new variable

14.55.2 Description

ImportVar(newvar=channelname|variable): This application imports a variable from the specified channel (as opposed to the current one) and stores it as a variable in the current channel (the channel that is calling this application). Variables created by this application have the same inheritance properties as those created with the Set application. See the documentation for Set for more information.

14.56 IVRDemo

14.56.1 Synopsis

IVR Demo Application

14.56.2 Description

This is a skeleton application that shows you the basic structure to create your own asterisk applications and demonstrates the IVR demo.

14.57 JabberSend

14.57.1 Synopsis

JabberSend(jabber,screenname,message)

14.57.2 Description

JabberSend(Jabber,ScreenName,Message)

```
  Jabber - Client or transport Asterisk uses to connect to Jabber
  ScreenName - User Name to message.
  Message - Message to be sent to the buddy
```

14.58 JabberStatus

14.58.1 Synopsis

```
JabberStatus(Jabber,ScreenName,Variable)
```

14.58.2 Description

```
JabberStatus(Jabber,ScreenName,Variable)
  Jabber - Client or transport Asterisk uses to connect to Jabber
  ScreenName - User Name to retrieve status from.
  Variable - Variable to store presence in will be 1-6.
             In order, Online, Chatty, Away, XAway, DND, Offline
             If not in roster variable will = 7
```

14.59 Log

14.59.1 Synopsis

```
Send arbitrary text to a selected log level
```

14.59.2 Description

```
Log(<level>|<message>)
  level must be one of ERROR, WARNING, NOTICE, DEBUG, VERBOSE, DTMF
```

14.60 LookupBlacklist

14.60.1 Synopsis

Look up Caller*ID name/number from blacklist database

14.60.2 Description

LookupBlacklist(options): Looks up the Caller*ID number on the active channel in the Asterisk database (family 'blacklist').
The option string may contain the following character:
'j' -- jump to n+101 priority if the number/name is found in the blacklist
This application sets the following channel variable upon completion:
LOOKUPBLSTATUS The status of the Blacklist lookup as a text string, one of FOUND | NOTFOUND
Example: exten => 1234,1,LookupBlacklist()

This application is deprecated and may be removed from a future release. Please use the dialplan function BLACKLIST() instead.

14.61 LookupCIDName

14.61.1 Synopsis

Look up CallerID Name from local database

14.61.2 Description

LookupCIDName: Looks up the Caller*ID number on the active channel in the Asterisk database (family 'cidname') and sets the Caller*ID name. Does nothing if no Caller*ID was received on the channel. This is useful if you do not subscribe to Caller*ID name delivery, or if you want to change the names on some incoming calls.

LookupCIDName is deprecated. Please use ${DB(cidname/${CALLERID(num)})} instead.

14.62 Macro

14.62.1 Synopsis

Macro Implementation

14.62.2 Description

```
  Macro(macroname|arg1|arg2...): Executes a macro using the context
'macro-<macroname>', jumping to the 's' extension of that context and
executing each step, then returning when the steps end.
The calling extension, context, and priority are stored in ${MACRO_EXTEN},
${MACRO_CONTEXT} and ${MACRO_PRIORITY} respectively.  Arguments become
${ARG1}, ${ARG2}, etc in the macro context.
If you Goto out of the Macro context, the Macro will terminate and control
will be returned at the location of the Goto.
If ${MACRO_OFFSET} is set at termination, Macro will attempt to continue
at priority MACRO_OFFSET + N + 1 if such a step exists, and N + 1 otherwise.
WARNING: Because of the way Macro is implemented (it executes the priorities
         contained within it via sub-engine), and a fixed per-thread
         memory stack allowance, macros are limited to 7 levels
         of nesting (macro calling macro calling macro, etc.); It
         may be possible that stack-intensive applications in deeply nested macros
         could cause asterisk to crash earlier than this limit.
```

14.63 MacroExclusive

14.63.1 Synopsis

Exclusive Macro Implementation

14.63.2 Description

```
  MacroExclusive(macroname|arg1|arg2...):
```

```
Executes macro defined in the context 'macro-macroname'
Only one call at a time may run the macro.
(we'll wait if another call is busy executing in the Macro)
Arguments and return values as in application Macro()
```

14.64 MacroExit

14.64.1 Synopsis

```
Exit From Macro
```

14.64.2 Description

```
  MacroExit():
Causes the currently running macro to exit as if it had
ended normally by running out of priorities to execute.
If used outside a macro, will likely cause unexpected
behavior.
```

14.65 MacroIf

14.65.1 Synopsis

```
Conditional Macro Implementation
```

14.65.2 Description

```
  MacroIf(<expr>?macroname_a[|arg1][:macroname_b[|arg1]])
Executes macro defined in <macroname_a> if <expr> is true
(otherwise <macroname_b> if provided)
Arguments and return values as in application macro()
```

14.66 MailboxExists

14.66.1 Synopsis

Check to see if Voicemail mailbox exists

14.66.2 Description

```
  MailboxExists(mailbox[@context][|options]): Check to see if the specified
mailbox exists. If no voicemail context is specified, the 'default' context
will be used.
  This application will set the following channel variable upon completion:
    VMBOXEXISTSSTATUS - This will contain the status of the execution of the
                        MailboxExists application. Possible values include:
                        SUCCESS | FAILED

  Options:
    j - Jump to priority n+101 if the mailbox is found.
```

14.67 MeetMe

14.67.1 Synopsis

MeetMe conference bridge

14.67.2 Description

```
  MeetMe([confno][,[options][,pin]]): Enters the user into a specified MeetMe
conference.  If the conference number is omitted, the user will be prompted
to enter one.  User can exit the conference by hangup, or if the 'p' option
is specified, by pressing '#'.
Please note: The Zaptel kernel modules and at least one hardware driver (or ztdumm
             must be present for conferencing to operate properly. In addition, th
             channel driver must be loaded for the 'i' and 'r' options to operate

The option string may contain zero or more of the following characters:
      'a' -- set admin mode
```

```
'A' -- set marked mode
'b' -- run AGI script specified in ${MEETME_AGI_BACKGROUND}
       Default: conf-background.agi  (Note: This does not work with
       non-Zap channels in the same conference)
'c' -- announce user(s) count on joining a conference
'd' -- dynamically add conference
'D' -- dynamically add conference, prompting for a PIN
'e' -- select an empty conference
'E' -- select an empty pinless conference
'F' -- Pass DTMF through the conference.  DTMF used to activate any
       conference features will not be passed through.
'i' -- announce user join/leave with review
'I' -- announce user join/leave without review
'l' -- set listen only mode (Listen only, no talking)
'm' -- set initially muted
'M' -- enable music on hold when the conference has a single caller
'o' -- set talker optimization - treats talkers who aren't speaking as
       being muted, meaning (a) No encode is done on transmission and
       (b) Received audio that is not registered as talking is omitted
       causing no buildup in background noise
'p' -- allow user to exit the conference by pressing '#'
'P' -- always prompt for the pin even if it is specified
'q' -- quiet mode (don't play enter/leave sounds)
'r' -- Record conference (records as ${MEETME_RECORDINGFILE}
       using format ${MEETME_RECORDINGFORMAT}). Default filename is
       meetme-conf-rec-${CONFNO}-${UNIQUEID} and the default format is
       wav.
's' -- Present menu (user or admin) when '*' is received ('send' to menu)
't' -- set talk only mode. (Talk only, no listening)
'T' -- set talker detection (sent to manager interface and meetme list)
'w[(<secs>)]'
    -- wait until the marked user enters the conference
'x' -- close the conference when last marked user exits
'X' -- allow user to exit the conference by entering a valid single
       digit extension ${MEETME_EXIT_CONTEXT} or the current context
       if that variable is not defined.
'1' -- do not play message when first person enters
```

14.68 MeetMeAdmin

14.68.1 Synopsis

MeetMe conference Administration

14.68.2 Description

```
  MeetMeAdmin(confno,command[,user]): Run admin command for conference
      'e' -- Eject last user that joined
      'k' -- Kick one user out of conference
      'K' -- Kick all users out of conference
      'l' -- Unlock conference
      'L' -- Lock conference
      'm' -- Unmute one user
      'M' -- Mute one user
      'n' -- Unmute all users in the conference
      'N' -- Mute all non-admin users in the conference
      'r' -- Reset one user's volume settings
      'R' -- Reset all users volume settings
      's' -- Lower entire conference speaking volume
      'S' -- Raise entire conference speaking volume
      't' -- Lower one user's talk volume
      'T' -- Raise one user's talk volume
      'u' -- Lower one user's listen volume
      'U' -- Raise one user's listen volume
      'v' -- Lower entire conference listening volume
      'V' -- Raise entire conference listening volume
```

14.69 MeetMeCount

14.69.1 Synopsis

MeetMe participant count

14.69.2 Description

MeetMeCount(confno[|var]): Plays back the number of users in the specified MeetMe conference. If var is specified, playback will be skipped and the value will be returned in the variable. Upon app completion, MeetMeCount will hangup the channel, unless priority n+1 exists, in which case priority progress will continue.
A ZAPTEL INTERFACE MUST BE INSTALLED FOR CONFERENCING FUNCTIONALITY.

14.70 Milliwatt

14.70.1 Synopsis

Generate a Constant 1000Hz tone at 0dbm (mu-law)

14.70.2 Description

Milliwatt(): Generate a Constant 1000Hz tone at 0dbm (mu-law)

14.71 MixMonitor

14.71.1 Synopsis

Record a call and mix the audio during the recording

14.71.2 Description

MixMonitor(<file>.<ext>[|<options>[|<command>]])

Records the audio on the current channel to the specified file.
If the filename is an absolute path, uses that path, otherwise
creates the file in the configured monitoring directory from
asterisk.conf.

Valid options:

```
 a      - Append to the file instead of overwriting it.
 b      - Only save audio to the file while the channel is bridged.
          Note: Does not include conferences or sounds played to each bridged
                party.
 v(<x>) - Adjust the heard volume by a factor of <x> (range -4 to 4)
 V(<x>) - Adjust the spoken volume by a factor of <x> (range -4 to 4)
 W(<x>) - Adjust the both heard and spoken volumes by a factor of <x>
         (range -4 to 4)

<command> will be executed when the recording is over
Any strings matching ^{X} will be unescaped to ${X} and
all variables will be evaluated at that time.
The variable MIXMONITOR_FILENAME will contain the filename used to record.
```

14.72 Monitor

14.72.1 Synopsis

```
Monitor a channel
```

14.72.2 Description

```
Monitor([file_format[:urlbase]|[fname_base]|[options]]):
Used to start monitoring a channel. The channel's input and output
voice packets are logged to files until the channel hangs up or
monitoring is stopped by the StopMonitor application.
  file_format optional, if not set, defaults to "wav"
  fname_base if set, changes the filename used to the one specified.
  options:
    m   - when the recording ends mix the two leg files into one and
          delete the two leg files.  If the variable MONITOR_EXEC is set, the
          application referenced in it will be executed instead of
          soxmix and the raw leg files will NOT be deleted automatically.
          soxmix or MONITOR_EXEC is handed 3 arguments, the two leg files
          and a target mixed file name which is the same as the leg file names
          only without the in/out designator.
```

If MONITOR_EXEC_ARGS is set, the contents will be passed on as
additional arguements to MONITOR_EXEC
Both MONITOR_EXEC and the Mix flag can be set from the
administrator interface

b - Don't begin recording unless a call is bridged to another channel

Returns -1 if monitor files can't be opened or if the channel is already monitored, otherwise 0.

14.73 Morsecode

14.73.1 Synopsis

Plays morse code

14.73.2 Description

Usage: Morsecode(<string>)
Plays the Morse code equivalent of the passed string. If the variable MORSEDITLEN is set, it will use that value for the length (in ms) of the dit (defaults to 80). Additionally, if MORSETONE is set, it will use that tone (in Hz). The tone default is 800.

14.74 MP3Player

14.74.1 Synopsis

Play an MP3 file or stream

14.74.2 Description

MP3Player(location) Executes mpg123 to play the given location, which typically would be a filename or a URL. User can exit by pressing any key on the dialpad, or by hanging up.

14.75 MusicOnHold

14.75.1 Synopsis

Play Music On Hold indefinitely

14.75.2 Description

MusicOnHold(class): Plays hold music specified by class. If omitted, the default
music source for the channel will be used. Set the default
class with the SetMusicOnHold() application.
Returns -1 on hangup.
Never returns otherwise.

14.76 NBScat

14.76.1 Synopsis

Play an NBS local stream

14.76.2 Description

NBScat: Executes nbscat to listen to the local NBS stream.
User can exit by pressing any key
.

14.77 NoCDR

14.77.1 Synopsis

Tell Asterisk to not maintain a CDR for the current call

14.77.2 Description

NoCDR(): This application will tell Asterisk not to maintain a CDR for the
current call.

14.78 NoOp

14.78.1 Synopsis

Do Nothing

14.78.2 Description

NoOp(): This applicatiion does nothing. However, it is useful for debugging purposes. Any text that is provided as arguments to this application can be viewed at the Asterisk CLI. This method can be used to see the evaluations of variables or functions without having any effect.

14.79 Page

14.79.1 Synopsis

Pages phones

14.79.2 Description

Page(Technology/Resource&Technology2/Resource2[|options])
Places outbound calls to the given technology / resource and dumps them into a conference bridge as muted participants. The original caller is dumped into the conference as a speaker and the room is destroyed when the original caller leaves. Valid options are:

d - full duplex audio
q - quiet, do not play beep to caller
r - record the page into a file (see 'r' for app_meetme)

14.80 Park

14.80.1 Synopsis

Park yourself

14.80.2 Description

```
Park():Used to park yourself (typically in combination with a supervised
transfer to know the parking space). This application is always
registered internally and does not need to be explicitly added
into the dialplan, although you should include the 'parkedcalls'
context (or the context specified in features.conf).

If you set the PARKINGEXTEN variable to an extension in your
parking context, park() will park the call on that extension, unless
it already exists. In that case, execution will continue at next
priority.
```

14.81 ParkAndAnnounce

14.81.1 Synopsis

```
Park and Announce
```

14.81.2 Description

```
  ParkAndAnnounce(announce:template|timeout|dial|[return_context]):
Park a call into the parkinglot and announce the call to another channel.

announce template: Colon-separated list of files to announce.  The word PARKED
                   will be replaced by a say_digits of the extension in which
                   the call is parked.
timeout:           Time in seconds before the call returns into the return
                   context.
dial:              The app_dial style resource to call to make the
                   announcement.  Console/dsp calls the console.
return_context:    The goto-style label to jump the call back into after
                   timeout.  Default <priority+1>.

The variable ${PARKEDAT} will contain the parking extension into which the
call was placed.  Use with the Local channel to allow the dialplan to make
use of this information.
```

14.82 ParkedCall

14.82.1 Synopsis

Answer a parked call

14.82.2 Description

ParkedCall(exten):Used to connect to a parked call. This application is always
registered internally and does not need to be explicitly added
into the dialplan, although you should include the 'parkedcalls'
context.

14.83 PauseMonitor

14.83.1 Synopsis

Pause monitoring of a channel

14.83.2 Description

PauseMonitor
Pauses monitoring of a channel until it is re-enabled by a call to UnpauseMonitor.

14.84 PauseQueueMember

14.84.1 Synopsis

Pauses a queue member

14.84.2 Description

```
   PauseQueueMember([queuename]|interface[|options]):
Pauses (blocks calls for) a queue member.
The given interface will be paused in the given queue.  This prevents
any calls from being sent from the queue to the interface until it is
unpaused with UnpauseQueueMember or the manager interface.  If no
queuename is given, the interface is paused in every queue it is a
member of.  If the interface is not in the named queue, or if no queue
is given and the interface is not in any queue, it will jump to
priority n+101, if it exists and the appropriate options are set.
The application will fail if the interface is not found and no extension
to jump to exists.
The option string may contain zero or more of the following characters:
       'j' -- jump to +101 priority when appropriate.
  This application sets the following channel variable upon completion:
     PQMSTATUS      The status of the attempt to pause a queue member as a
                     text string, one of
           PAUSED | NOTFOUND
Example: PauseQueueMember(|SIP/3000)
```

14.85 Pickup

14.85.1 Synopsis

```
Directed Call Pickup
```

14.85.2 Description

```
  Pickup(extension[@context][&extension2@context...]): This application can pickup
that is calling the specified extension. If no context is specified, the current
context will be used. If you use the special string "PICKUPMARK" for the context p
10@PICKUPMARK, this application tries to find a channel which has defined a channe
as "extension".
```

14.86 Playback

14.86.1 Synopsis

Play a file

14.86.2 Description

```
  Playback(filename[&filename2...][|option]):  Plays back given filenames (do not
extension). Options may also be included following a pipe symbol. The 'skip'
option causes the playback of the message to be skipped if the channel
is not in the 'up' state (i.e. it hasn't been  answered  yet). If 'skip' is
specified, the application will return immediately should the channel not be
off hook.  Otherwise, unless 'noanswer' is specified, the channel will
be answered before the sound is played. Not all channels support playing
messages while still on hook. If 'j' is specified, the application
will jump to priority n+101 if present when a file specified to be played
does not exist.
This application sets the following channel variable upon completion:
 PLAYBACKSTATUS    The status of the playback attempt as a text string, one of
                SUCCESS | FAILED
```

14.87 PlayTones

14.87.1 Synopsis

Play a tone list

14.87.2 Description

```
PlayTones(arg): Plays a tone list. Execution will continue with the next step imme
while the tones continue to play.
Arg is either the tone name defined in the indications.conf configuration file, or
specified list of frequencies and durations.
See the sample indications.conf for a description of the specification of a toneli

Use the StopPlayTones application to stop the tones playing.
```

14.88 PrivacyManager

14.88.1 Synopsis

```
Require phone number to be entered, if no CallerID sent
```

14.88.2 Description

```
  PrivacyManager([maxretries[|minlength[|options]]]): If no Caller*ID
is sent, PrivacyManager answers the channel and asks the caller to
enter their phone number. The caller is given 3 attempts to do so.
The application does nothing if Caller*ID was received on the channel.
  Configuration file privacy.conf contains two variables:
   maxretries  default 3  -maximum number of attempts the caller is allowed
               to input a callerid.
   minlength   default 10 -minimum allowable digits in the input callerid number.
If you don't want to use the config file and have an i/o operation with
every call, you can also specify maxretries and minlength as application
parameters. Doing so supercedes any values set in privacy.conf.
The option string may contain the following character:
  'j' -- jump to n+101 priority after <maxretries> failed attempts to collect
         the minlength number of digits.
The application sets the following channel variable upon completion:
PRIVACYMGRSTATUS  The status of the privacy manager's attempt to collect
                  a phone number from the user. A text string that is either:
          SUCCESS | FAILED
```

14.89 Progress

14.89.1 Synopsis

```
Indicate progress
```

14.89.2 Description

```
  Progress(): This application will request that in-band progress information
be provided to the calling channel.
```

14.90 Queue

14.90.1 Synopsis

```
Queue a call for a call queue
```

14.90.2 Description

```
  Queue(queuename[|options[|URL][|announceoverride][|timeout][|AGI]):
Queues an incoming call in a particular call queue as defined in queues.conf.
This application will return to the dialplan if the queue does not exist, or
any of the join options cause the caller to not enter the queue.
The option string may contain zero or more of the following characters:
      'd' -- data-quality (modem) call (minimum delay).
      'h' -- allow callee to hang up by hitting *.
      'H' -- allow caller to hang up by hitting *.
      'n' -- no retries on the timeout; will exit this application and
             go to the next step.
      'i' -- ignore call forward requests from queue members and do nothing
             when they are requested.
      'r' -- ring instead of playing MOH
      't' -- allow the called user transfer the calling user
      'T' -- to allow the calling user to transfer the call.
      'w' -- allow the called user to write the conversation to disk via Monitor
      'W' -- allow the calling user to write the conversation to disk via Monitor
  In addition to transferring the call, a call may be parked and then picked
up by another user.
  The optional URL will be sent to the called party if the channel supports
it.
  The optional AGI parameter will setup an AGI script to be executed on the
calling party's channel once they are connected to a queue member.
  The timeout will cause the queue to fail out after a specified number of
```

```
seconds, checked between each queues.conf 'timeout' and 'retry' cycle.
  This application sets the following channel variable upon completion:
      QUEUESTATUS    The status of the call as a text string, one of
             TIMEOUT | FULL | JOINEMPTY | LEAVEEMPTY | JOINUNAVAIL | LEAVEUNAVAIL
```

14.91 QueueLog

14.91.1 Synopsis

```
Writes to the queue_log
```

14.91.2 Description

```
   QueueLog(queuename|uniqueid|agent|event[|additionalinfo]):
Allows you to write your own events into the queue log
Example: QueueLog(101|${UNIQUEID}|${AGENT}|WENTONBREAK|600)
```

14.92 Random

14.92.1 Synopsis

```
Conditionally branches, based upon a probability
```

14.92.2 Description

```
Random([probability]:[[context|]extension|]priority)
  probability := INTEGER in the range 1 to 100
DEPRECATED: Use GotoIf($[${RAND(1,100)} > <number>]?<label>)
```

14.93 Read

14.93.1 Synopsis

```
Read a variable
```

14.93.2 Description

```
  Read(variable[|filename][|maxdigits][|option][|attempts][|timeout])

Reads a #-terminated string of digits a certain number of times from the
user in to the given variable.
  filename   -- file to play before reading digits or tone with option i
  maxdigits  -- maximum acceptable number of digits. Stops reading after
                maxdigits have been entered (without requiring the user to
                press the '#' key).
                Defaults to 0 - no limit - wait for the user press the '#' key.
                Any value below 0 means the same. Max accepted value is 255.
  option     -- options are 's' , 'i', 'n'
                's' to return immediately if the line is not up,
                'i' to play  filename as an indication tone from your indications.
                'n' to read digits even if the line is not up.
  attempts   -- if greater than 1, that many attempts will be made in the
                event no data is entered.
  timeout    -- An integer number of seconds to wait for a digit response. If grea
                than 0, that value will override the default timeout.

Read should disconnect if the function fails or errors out.
```

14.94 ReadFile

14.94.1 Synopsis

```
ReadFile(varname=file,length)
```

14.94.2 Description

```
ReadFile(varname=file,length)
  Varname - Result stored here.
  File - The name of the file to read.
  Length - Maximum number of characters to capture.
```

14.95 RealTime

14.95.1 Synopsis

Realtime Data Lookup

14.95.2 Description

Use the RealTime config handler system to read data into channel variables.
RealTime(<family>|<colmatch>|<value>[|<prefix>])

All unique column names will be set as channel variables with optional prefix to the name. For example, a prefix of 'var_' would make the column 'name' become the variable ${var_name}. REALTIMECOUNT will be set with the number of values read.

14.96 RealTimeUpdate

14.96.1 Synopsis

Realtime Data Rewrite

14.96.2 Description

Use the RealTime config handler system to update a value
RealTimeUpdate(<family>|<colmatch>|<value>|<newcol>|<newval>)

The column <newcol> in 'family' matching column <colmatch>=<value> will be updated to <newval>. REALTIMECOUNT will be set with the number of rows updated or -1 if an error occurs.

14.97 Record

14.97.1 Synopsis

Record to a file

14.97.2 Description

```
  Record(filename.format|silence[|maxduration][|options])

Records from the channel into a given filename. If the file exists it will
be overwritten.
- 'format' is the format of the file type to be recorded (wav, gsm, etc).
- 'silence' is the number of seconds of silence to allow before returning.
- 'maxduration' is the maximum recording duration in seconds. If missing
or 0 there is no maximum.
- 'options' may contain any of the following letters:
     'a' : append to existing recording rather than replacing
     'n' : do not answer, but record anyway if line not yet answered
     'q' : quiet (do not play a beep tone)
     's' : skip recording if the line is not yet answered
     't' : use alternate '*' terminator key (DTMF) instead of default '#'
     'x' : ignore all terminator keys (DTMF) and keep recording until hangup

If filename contains '%d', these characters will be replaced with a number
incremented by one each time the file is recorded. A channel variable
named RECORDED_FILE will also be set, which contains the final filemname.

Use 'core show file formats' to see the available formats on your system

User can press '#' to terminate the recording and continue to the next priority.

If the user should hangup during a recording, all data will be lost and the
application will teminate.
```

14.98 RemoveQueueMember

14.98.1 Synopsis

```
Dynamically removes queue members
```

14.98.2 Description

```
   RemoveQueueMember(queuename[|interface[|options]]):
Dynamically removes interface to an existing queue
If the interface is NOT in the queue and there exists an n+101 priority
then it will then jump to this priority.  Otherwise it will return an error
The option string may contain zero or more of the following characters:
       'j' -- jump to +101 priority when appropriate.
  This application sets the following channel variable upon completion:
     RQMSTATUS      The status of the attempt to remove a queue member as a
                     text string, one of
           REMOVED | NOTINQUEUE | NOSUCHQUEUE
Example: RemoveQueueMember(techsupport|SIP/3000)
```

14.99 ResetCDR

14.99.1 Synopsis

```
Resets the Call Data Record
```

14.99.2 Description

```
  ResetCDR([options]):  This application causes the Call Data Record to be
reset.
  Options:
    w -- Store the current CDR record before resetting it.
    a -- Store any stacked records.
    v -- Save CDR variables.
```

14.100 RetryDial

14.100.1 Synopsis

```
Place a call, retrying on failure allowing optional exit extension.
```

14.100.2 Description

RetryDial(announce|sleep|retries|dialargs): This application will attempt to place a call using the normal Dial application. If no channel can be reached, the 'announce' file will be played. Then, it will wait 'sleep' number of seconds before retying the call. After 'retires' number of attempts, the calling channel will continue at the next priority in the dialplan. If the 'retries' setting is set to 0, this application will retry endlessly.

While waiting to retry a call, a 1 digit extension may be dialed. If that extension exists in either the context defined in ${EXITCONTEXT} or the current one, The call will jump to that extension immediately.

The 'dialargs' are specified in the same format that arguments are provided to the Dial application.

14.101 Return

14.101.1 Synopsis

Return from gosub routine

14.101.2 Description

Return()

Jumps to the last label on the stack, removing it.

14.102 Ringing

14.102.1 Synopsis

Indicate ringing tone

14.102.2 Description

Ringing(): This application will request that the channel indicate a ringing tone to the user.

14.103 Rpt

14.103.1 Synopsis

Radio Repeater/Remote Base Control System

14.103.2 Description

```
  Rpt(nodename[|options]):  Radio Remote Link or Remote Base Link Endpoint Process

    Not specifying an option puts it in normal endpoint mode (where source
    IP and nodename are verified).

    Options are as follows:

        X - Normal endpoint mode WITHOUT security check. Only specify
            this if you have checked security already (like with an IAX2
            user/password or something).

        Rannounce-string[|timeout[|timeout-destination]] - Amateur Radio
            Reverse Autopatch. Caller is put on hold, and announcement (as
            specified by the 'announce-string') is played on radio system.
            Users of radio system can access autopatch, dial specified
            code, and pick up call. Announce-string is list of names of
            recordings, or "PARKED" to substitute code for un-parking,
            or "NODE" to substitute node number.

        P - Phone Control mode. This allows a regular phone user to have
            full control and audio access to the radio system. For the
            user to have DTMF control, the 'phone_functions' parameter
            must be specified for the node in 'rpt.conf'. An additional
            function (cop,6) must be listed so that PTT control is available.

        D - Dumb Phone Control mode. This allows a regular phone user to
            have full control and audio access to the radio system. In this
            mode, the PTT is activated for the entire length of the call.
            For the user to have DTMF control (not generally recomended in
            this mode), the 'dphone_functions' parameter must be specified
```

for the node in 'rpt.conf'. Otherwise no DTMF control will be available to the phone user.

14.104 SayAlpha

14.104.1 Synopsis

Say Alpha

14.104.2 Description

SayAlpha(string): This application will play the sounds that correspond to the letters of the given string.

14.105 SayDigits

14.105.1 Synopsis

Say Digits

14.105.2 Description

SayDigits(digits): This application will play the sounds that correspond to the digits of the given number. This will use the language that is currently set for the channel. See the LANGUAGE function for more information on setting the language for the channel.

14.106 SayNumber

14.106.1 Synopsis

Say Number

14.106.2 Description

SayNumber(digits[,gender]): This application will play the sounds that correspond to the given number. Optionally, a gender may be specified. This will use the language that is currently set for the channel. See the LANGUAGE function for more information on setting the language for the channel.

14.107 SayPhonetic

14.107.1 Synopsis

Say Phonetic

14.107.2 Description

SayPhonetic(string): This application will play the sounds from the phonetic alphabet that correspond to the letters in the given string.

14.108 SayUnixTime

14.108.1 Synopsis

Says a specified time in a custom format

14.108.2 Description

```
SayUnixTime([unixtime][|[timezone][|format]])
  unixtime: time, in seconds since Jan 1, 1970.  May be negative.
            defaults to now.
  timezone: timezone, see /usr/share/zoneinfo for a list.
            defaults to machine default.
  format:   a format the time is to be said in.  See voicemail.conf.
            defaults to "ABdY 'digits/at' IMp"
```

14.109 SendDTMF

14.109.1 Synopsis

Sends arbitrary DTMF digits

14.109.2 Description

```
SendDTMF(digits[|timeout_ms]): Sends DTMF digits on a channel.
Accepted digits: 0-9, *#abcd, w (.5s pause)
The application will either pass the assigned digits or terminate if it
encounters an error.
```

14.110 SendImage

14.110.1 Synopsis

Send an image file

14.110.2 Description

```
  SendImage(filename): Sends an image on a channel.
If the channel supports image transport but the image send
fails, the channel will be hung up. Otherwise, the dialplan
continues execution.
The option string may contain the following character:
'j' -- jump to priority n+101 if the channel doesn't support image transport
This application sets the following channel variable upon completion:
SENDIMAGESTATUS The status is the result of the attempt as a text string, one of
OK | NOSUPPORT
```

14.111 SendText

14.111.1 Synopsis

Send a Text Message

14.111.2 Description

```
  SendText(text[|options]): Sends text to current channel (callee).
Result of transmission will be stored in the SENDTEXTSTATUS
channel variable:
      SUCCESS      Transmission succeeded
      FAILURE      Transmission failed
      UNSUPPORTED  Text transmission not supported by channel

At this moment, text is supposed to be 7 bit ASCII in most channels.
The option string many contain the following character:
'j' -- jump to n+101 priority if the channel doesn't support
       text transport
```

14.112 SendURL

14.112.1 Synopsis

```
Send a URL
```

14.112.2 Description

```
  SendURL(URL[|option]): Requests client go to URL (IAX2) or sends the
URL to the client (other channels).
Result is returned in the SENDURLSTATUS channel variable:
    SUCCESS       URL successfully sent to client
    FAILURE       Failed to send URL
    NOLOAD        Client failed to load URL (wait enabled)
    UNSUPPORTED   Channel does not support URL transport

If the option 'wait' is specified, execution will wait for an
acknowledgement that the URL has been loaded before continuing

If jumping is specified as an option (the 'j' flag), the client does not
support Asterisk "html" transport, and there exists a step with priority
n + 101, then execution will continue at that step.
```

SendURL continues normally if the URL was sent correctly or if the channel does not support HTML transport. Otherwise, the channel is hung up.

14.113 Set

14.113.1 Synopsis

Set channel variable(s) or function value(s)

14.113.2 Description

```
  Set(name1=value1|name2=value2|..[|options])
```
This function can be used to set the value of channel variables or dialplan functions. It will accept up to 24 name/value pairs. When setting variables, if the variable name is prefixed with _, the variable will be inherited into channels created from the current channel. If the variable name is prefixed with __, the variable will be inherited into channels created from the current channel and all children channels.

```
  Options:
    g - Set variable globally instead of on the channel
        (applies only to variables, not functions)
```

The use of Set to set multiple variables at once and the g flag have both been deprecated. Please use multiple Set calls and the GLOBAL() dialplan function instead.

14.114 SetAMAFlags

14.114.1 Synopsis

Set the AMA Flags

14.114.2 Description

SetAMAFlags([flag]): This application will set the channel's AMA Flags for

billing purposes.

14.115 SetCallerID

14.115.1 Synopsis

Set CallerID

14.115.2 Description

SetCallerID(clid[|a]): Set Caller*ID on a call to a new value. Sets ANI as well if a flag is used.

This application has been deprecated in favor of Set(CALLERID(all)=...)

14.116 SetCallerPres

14.116.1 Synopsis

Set CallerID Presentation

14.116.2 Description

SetCallerPres(presentation): Set Caller*ID presentation on a call. Valid presentations are:

```
    allowed_not_screened    : Presentation Allowed, Not Screened
    allowed_passed_screen   : Presentation Allowed, Passed Screen
    allowed_failed_screen   : Presentation Allowed, Failed Screen
    allowed                 : Presentation Allowed, Network Number
    prohib_not_screened     : Presentation Prohibited, Not Screened
    prohib_passed_screen    : Presentation Prohibited, Passed Screen
    prohib_failed_screen    : Presentation Prohibited, Failed Screen
    prohib                  : Presentation Prohibited, Network Number
    unavailable             : Number Unavailable
```

14.117 SetCDRUserField

14.117.1 Synopsis

Set the CDR user field

14.117.2 Description

```
[Synopsis]
SetCDRUserField(value)

[Description]
SetCDRUserField(value): Set the CDR 'user field' to value
       The Call Data Record (CDR) user field is an extra field you
       can use for data not stored anywhere else in the record.
       CDR records can be used for billing or storing other arbitrary data
       (I.E. telephone survey responses)
       Also see AppendCDRUserField().

This application has been deprecated in favor of Set(CDR(userfield)=...)
```

14.118 SetGlobalVar

14.118.1 Synopsis

Set a global variable to a given value

14.118.2 Description

```
  SetGlobalVar(variable=value): This application sets a given global variable to
the specified value.

This application has been deprecated in favor of Set(GLOBAL(var)=value)
```

14.119 SetMusicOnHold

14.119.1 Synopsis

```
Set default Music On Hold class
```

14.119.2 Description

```
SetMusicOnHold(class): Sets the default class for music on hold for a given channe
music on hold is activated, this class will be used to select which
music is played.
```

14.120 SetTransferCapability

14.120.1 Synopsis

```
Set ISDN Transfer Capability
```

14.120.2 Description

```
  SetTransferCapability(transfercapability): Set the ISDN Transfer
Capability of a call to a new value.
Valid Transfer Capabilities are:

  SPEECH              : 0x00 - Speech (default, voice calls)
  DIGITAL             : 0x08 - Unrestricted digital information (data calls)
  RESTRICTED_DIGITAL  : 0x09 - Restricted digital information
  3K1AUDIO            : 0x10 - 3.1kHz Audio (fax calls)
  DIGITAL_W_TONES     : 0x11 - Unrestricted digital information with tones/announce
  VIDEO               : 0x18 - Video

This application has been deprecated in favor of Set(CHANNEL(transfercapability)=
```

14.121 SIPAddHeader

14.121.1 Synopsis

Add a SIP header to the outbound call

14.121.2 Description

```
  SIPAddHeader(Header: Content)
Adds a header to a SIP call placed with DIAL.
Remember to user the X-header if you are adding non-standard SIP
headers, like "X-Asterisk-Accountcode:". Use this with care.
Adding the wrong headers may jeopardize the SIP dialog.
Always returns 0
```

14.122 SIPDtmfMode

14.122.1 Synopsis

Change the dtmfmode for a SIP call

14.122.2 Description

SIPDtmfMode(inband|info|rfc2833): Changes the dtmfmode for a SIP call

14.123 SLAStation

14.123.1 Synopsis

Shared Line Appearance Station

14.123.2 Description

```
  SLAStation(station):
This application should be executed by an SLA station.  The argument depends
```

on how the call was initiated. If the phone was just taken off hook, then the argument "station" should be just the station name. If the call was initiated by pressing a line key, then the station name should be preceded by an underscore and the trunk name associated with that line button. For example: "station1_line1".

14.124 SLATrunk

14.124.1 Synopsis

Shared Line Appearance Trunk

14.124.2 Description

  SLATrunk(trunk):
This application should be executed by an SLA trunk on an inbound call. The channel calling this application should correspond to the SLA trunk with the name "trunk" that is being passed as an argument.

14.125 SMS

14.125.1 Synopsis

Communicates with SMS service centres and SMS capable analogue phones

14.125.2 Description

  SMS(name|[a][s]): SMS handles exchange of SMS data with a call to/from SMS capa
phone or SMS PSTN service center. Can send and/or receive SMS messages.
Works to ETSI ES 201 912 compatible with BT SMS PSTN service in UK
Typical usage is to use to handle called from the SMS service centre CLI, or to set up a call using 'outgoing' or manager interface to connect service centre to SMS()
name is the name of the queue used in /var/spool/asterisk/sms
Arguments:
 a: answer, i.e. send initial FSK packet.

```
 s: act as service centre talking to a phone.
Messages are processed as per text file message queues.
smsq (a separate software) is a command to generate message
queues and send messages.
```

14.126 SoftHangup

14.126.1 Synopsis

```
Soft Hangup Application
```

14.126.2 Description

```
  SoftHangup(Technology/resource|options)
Hangs up the requested channel.  If there are no channels to hangup,
the application will report it.
- 'options' may contain the following letter:
     'a' : hang up all channels on a specified device instead of a single resource
```

14.127 SpeechActivateGrammar

14.127.1 Synopsis

```
Activate a Grammar
```

14.127.2 Description

```
SpeechActivateGrammar(Grammar Name)
This activates the specified grammar to be recognized by the engine. A grammar tel
and how to portray it back to you in the dialplan. The grammar name is the only ar
```

14.128 SpeechBackground

14.128.1 Synopsis

Play a sound file and wait for speech to be recognized

14.128.2 Description

SpeechBackground(Sound File|Timeout)
This application plays a sound file and waits for the person to speak. Once they s
Once they stop talking the processing sound is played to indicate the speech recog
Once results are available the application returns and results (score and text) ar
The first text and score are ${SPEECH_TEXT(0)} AND ${SPEECH_SCORE(0)} while the se
The first argument is the sound file and the second is the timeout. Note the timeo

14.129 SpeechCreate

14.129.1 Synopsis

Create a Speech Structure

14.129.2 Description

SpeechCreate(engine name)
This application creates information to be used by all the other applications. It
It takes the engine name to use as the argument, if not specified the default engi

14.130 SpeechDeactivateGrammar

14.130.1 Synopsis

Deactivate a Grammar

14.130.2 Description

SpeechDeactivateGrammar(Grammar Name)
This deactivates the specified grammar so that it is no longer recognized. The onl

14.131 SpeechDestroy

14.131.1 Synopsis

End speech recognition

14.131.2 Description

SpeechDestroy()
This destroys the information used by all the other speech recognition applicatior
If you call this application but end up wanting to recognize more speech, you must
again before calling any other application. It takes no arguments.

14.132 SpeechLoadGrammar

14.132.1 Synopsis

Load a Grammar

14.132.2 Description

SpeechLoadGrammar(Grammar Name|Path)
Load a grammar only on the channel, not globally.
It takes the grammar name as first argument and path as second.

14.133 SpeechProcessingSound

14.133.1 Synopsis

Change background processing sound

14.133.2 Description

SpeechProcessingSound(Sound File)
This changes the processing sound that SpeechBackground plays back when the speech
It takes the sound file as the only argument.

14.134 SpeechStart

14.134.1 Synopsis

Start recognizing voice in the audio stream

14.134.2 Description

SpeechStart()
Tell the speech recognition engine that it should start trying to get results from

14.135 SpeechUnloadGrammar

14.135.1 Synopsis

Unload a Grammar

14.135.2 Description

SpeechUnloadGrammar(Grammar Name)
Unload a grammar. It takes the grammar name as the only argument.

14.136 StackPop

14.136.1 Synopsis

Remove one address from gosub stack

14.136.2 Description

StackPop()
  Removes last label on the stack, discarding it.

14.137 StartMusicOnHold

14.137.1 Synopsis

Play Music On Hold

14.137.2 Description

StartMusicOnHold(class): Starts playing music on hold, uses default music class fo
Starts playing music specified by class. If omitted, the default
music source for the channel will be used. Always returns 0.

14.138 StopMixMonitor

14.138.1 Synopsis

Stop recording a call through MixMonitor

14.138.2 Description

  StopMixMonitor()

Stops the audio recording that was started with a call to MixMonitor()
on the current channel.

14.139 StopMonitor

14.139.1 Synopsis

Stop monitoring a channel

14.139.2 Description

StopMonitor
Stops monitoring a channel. Has no effect if the channel is not monitored

14.140 StopMusicOnHold

14.140.1 Synopsis

Stop Playing Music On Hold

14.140.2 Description

StopMusicOnHold: Stops playing music on hold.

14.141 StopPlayTones

14.141.1 Synopsis

Stop playing a tone list

14.141.2 Description

Stop playing a tone list

14.142 System

14.142.1 Synopsis

Execute a system command

14.142.2 Description

System(command): Executes a command by using system(). If the command fails, the console should report a fallthrough.
Result of execution is returned in the SYSTEMSTATUS channel variable:
FAILURE Could not execute the specified command
SUCCESS Specified command successfully executed

Old behaviour:
If the command itself executes but is in error, and if there exists a priority n + 101, where 'n' is the priority of the current instance, then the channel will be setup to continue at that priority level. Note that this jump functionality has been deprecated and will only occur if the global priority jumping option is enabled in extensions.conf.

14.143 TestClient

14.143.1 Synopsis

Execute Interface Test Client

14.143.2 Description

TestClient(testid): Executes test client with given testid.
Results stored in /var/log/asterisk/testreports/<testid>-client.txt

14.144 TestServer

14.144.1 Synopsis

Execute Interface Test Server

14.144.2 Description

```
TestServer(): Perform test server function and write call report.
Results stored in /var/log/asterisk/testreports/<testid>-server.txt
```

14.145 Transfer

14.145.1 Synopsis

```
Transfer caller to remote extension
```

14.145.2 Description

```
  Transfer([Tech/]dest[|options]):  Requests the remote caller be transferred
to a given destination. If TECH (SIP, IAX2, LOCAL etc) is used, only
an incoming call with the same channel technology will be transfered.
Note that for SIP, if you transfer before call is setup, a 302 redirect
SIP message will be returned to the caller.

The result of the application will be reported in the TRANSFERSTATUS
channel variable:
       SUCCESS      Transfer succeeded
       FAILURE      Transfer failed
       UNSUPPORTED  Transfer unsupported by channel driver
The option string many contain the following character:
'j' -- jump to n+101 priority if the channel transfer attempt
       fails
```

14.146 TryExec

14.146.1 Synopsis

```
Executes dialplan application, always returning
```

14.146.2 Description

```
Usage: TryExec(appname(arguments))
```

```
  Allows an arbitrary application to be invoked even when not
hardcoded into the dialplan. To invoke external applications
see the application System.  Always returns to the dialplan.
The channel variable TRYSTATUS will be set to:
    SUCCESS   if the application returned zero
    FAILED    if the application returned non-zero
    NOAPP     if the application was not found or was not specified
```

14.147 TrySystem

14.147.1 Synopsis

```
Try executing a system command
```

14.147.2 Description

```
  TrySystem(command): Executes a command  by  using  system().
on any situation.
Result of execution is returned in the SYSTEMSTATUS channel variable:
   FAILURE Could not execute the specified command
   SUCCESS Specified command successfully executed
   APPERROR Specified command successfully executed, but returned error code

Old behaviour:
If  the command itself executes but is in error, and if
there exists a priority n + 101, where 'n' is the priority of the current
instance, then  the  channel  will  be  setup  to continue at that
priority level.  Otherwise, System will terminate.
```

14.148 UnpauseMonitor

14.148.1 Synopsis

```
Unpause monitoring of a channel
```

14.148.2 Description

```
UnpauseMonitor
Unpauses monitoring of a channel on which monitoring had
previously been paused with PauseMonitor.
```

14.149 UnpauseQueueMember

14.149.1 Synopsis

```
Unpauses a queue member
```

14.149.2 Description

```
   UnpauseQueueMember([queuename]|interface[|options]):
Unpauses (resumes calls to) a queue member.
This is the counterpart to PauseQueueMember and operates exactly the
same way, except it unpauses instead of pausing the given interface.
The option string may contain zero or more of the following characters:
       'j' -- jump to +101 priority when appropriate.
  This application sets the following channel variable upon completion:
     UPQMSTATUS       The status of the attempt to unpause a queue
                      member as a text string, one of
             UNPAUSED | NOTFOUND
Example: UnpauseQueueMember(|SIP/3000)
```

14.150 UserEvent

14.150.1 Synopsis

```
Send an arbitrary event to the manager interface
```

14.150.2 Description

```
  UserEvent(eventname[|body]): Sends an arbitrary event to the manager
```

interface, with an optional body representing additional arguments. The body may be specified as a | delimeted list of headers. Each additional argument will be placed on a new line in the event. The format of the event will be:

```
    Event: UserEvent
    UserEvent: <specified event name>
    [body]
```

If no body is specified, only Event and UserEvent headers will be present.

14.151 Verbose

14.151.1 Synopsis

Send arbitrary text to verbose output

14.151.2 Description

```
Verbose([<level>|]<message>)
  level must be an integer value.  If not specified, defaults to 0.
```

14.152 VMAuthenticate

14.152.1 Synopsis

Authenticate with Voicemail passwords

14.152.2 Description

VMAuthenticate([mailbox][@context][|options]): This application behaves the same way as the Authenticate application, but the passwords are taken from voicemail.conf.

If the mailbox is specified, only that mailbox's password will be considered valid. If the mailbox is not specified, the channel variable AUTH_MAILBOX will be set with the authenticated mailbox.

```
  Options:
    s - Skip playing the initial prompts.
```

14.153 VoiceMail

14.153.1 Synopsis

```
Leave a Voicemail message
```

14.153.2 Description

```
  VoiceMail(mailbox[@context][&mailbox[@context]][...][|options]): This
application allows the calling party to leave a message for the specified
list of mailboxes. When multiple mailboxes are specified, the greeting will
be taken from the first mailbox specified. Dialplan execution will stop if the
specified mailbox does not exist.
  The Voicemail application will exit if any of the following DTMF digits are
received:
    0 - Jump to the 'o' extension in the current dialplan context.
    * - Jump to the 'a' extension in the current dialplan context.
  This application will set the following channel variable upon completion:
    VMSTATUS - This indicates the status of the execution of the VoiceMail
               application. The possible values are:
               SUCCESS | USEREXIT | FAILED

  Options:
    b    - Play the 'busy' greeting to the calling party.
    g(#) - Use the specified amount of gain when recording the voicemail
           message. The units are whole-number decibels (dB).
    s    - Skip the playback of instructions for leaving a message to the
           calling party.
    u    - Play the 'unavailable greeting.
    j    - Jump to priority n+101 if the mailbox is not found or some other
           error occurs.
```

14.154 VoiceMailMain

14.154.1 Synopsis

Check Voicemail messages

14.154.2 Description

VoiceMailMain([mailbox][@context][|options]): This application allows the calling party to check voicemail messages. A specific mailbox, and optional corresponding context, may be specified. If a mailbox is not provided, the calling party will be prompted to enter one. If a context is not specified, the 'default' context will be used.

Options:
- p - Consider the mailbox parameter as a prefix to the mailbox that is entered by the caller.
- g(#) - Use the specified amount of gain when recording a voicemail message. The units are whole-number decibels (dB).
- s - Skip checking the passcode for the mailbox.
- a(#) - Skip folder prompt and go directly to folder specified. Defaults to INBOX

14.155 Wait

14.155.1 Synopsis

Waits for some time

14.155.2 Description

Wait(seconds): This application waits for a specified number of seconds. Then, dialplan execution will continue at the next priority.

Note that the seconds can be passed with fractions of a second. For example, '1.5' will ask the application to wait for 1.5 seconds.

14.156 WaitExten

14.156.1 Synopsis

Waits for an extension to be entered

14.156.2 Description

```
  WaitExten([seconds][|options]): This application waits for the user to enter
a new extension for a specified number of seconds.
  Note that the seconds can be passed with fractions of a second. For example,
'1.5' will ask the application to wait for 1.5 seconds.
  Options:
    m[(x)] - Provide music on hold to the caller while waiting for an extension.
               Optionally, specify the class for music on hold within parenthesis.
```

14.157 WaitForRing

14.157.1 Synopsis

Wait for Ring Application

14.157.2 Description

```
  WaitForRing(timeout)
Returns 0 after waiting at least timeout seconds. and
only after the next ring has completed.  Returns 0 on
success or -1 on hangup
```

14.158 WaitForSilence

14.158.1 Synopsis

Waits for a specified amount of silence

14.158.2 Description

```
  WaitForSilence(silencerequired[|iterations][|timeout])
Wait for Silence: Waits for up to 'silencerequired'
milliseconds of silence, 'iterations' times or once if omitted.
An optional timeout specified the number of seconds to return
after, even if we do not receive the specified amount of silence.
Use 'timeout' with caution, as it may defeat the purpose of this
application, which is to wait indefinitely until silence is detected
on the line.  This is particularly useful for reverse-911-type
call broadcast applications where you need to wait for an answering
machine to complete its spiel before playing a message.
The timeout parameter is specified only to avoid an infinite loop in
cases where silence is never achieved.  Typically you will want to
include two or more calls to WaitForSilence when dealing with an answering
machine; first waiting for the spiel to finish, then waiting for the beep, etc.

Examples:
  - WaitForSilence(500|2) will wait for 1/2 second of silence, twice
  - WaitForSilence(1000) will wait for 1 second of silence, once
  - WaitForSilence(300|3|10) will wait for 300ms silence, 3 times,
     and returns after 10 sec, even if silence is not detected

Sets the channel variable WAITSTATUS with to one of these values:
SILENCE - if exited with silence detected
TIMEOUT - if exited without silence detected after timeout
```

14.159 WaitMusicOnHold

14.159.1 Synopsis

```
Wait, playing Music On Hold
```

14.159.2 Description

```
WaitMusicOnHold(delay): Plays hold music specified number of seconds.  Returns 0 w
done, or -1 on hangup.  If no hold music is available, the delay will
```

still occur with no sound.

14.160 While

14.160.1 Synopsis

Start a while loop

14.160.2 Description

Usage: While(<expr>)
Start a While Loop. Execution will return to this point when
EndWhile is called until expr is no longer true.

14.161 Zapateller

14.161.1 Synopsis

Block telemarketers with SIT

14.161.2 Description

Zapateller(options): Generates special information tone to block
telemarketers from calling you. Options is a pipe-delimited list of
options. The following options are available:
'answer' causes the line to be answered before playing the tone,
'nocallerid' causes Zapateller to only play the tone if there
is no callerid information available. Options should be separated by |
characters

14.162 ZapBarge

14.162.1 Synopsis

Barge in (monitor) Zap channel

14.162.2 Description

ZapBarge([channel]): Barges in on a specified zap channel or prompts if one is not specified. Returns -1 when caller user hangs up and is independent of the state of the channel being monitored.

14.163 ZapRAS

14.163.1 Synopsis

Executes Zaptel ISDN RAS application

14.163.2 Description

ZapRAS(args): Executes a RAS server using pppd on the given channel. The channel must be a clear channel (i.e. PRI source) and a Zaptel channel to be able to use this function (No modem emulation is included). Your pppd must be patched to be zaptel aware. Arguments should be separated by | characters.

14.164 ZapScan

14.164.1 Synopsis

Scan Zap channels to monitor calls

14.164.2 Description

ZapScan([group]) allows a call center manager to monitor Zap channels in

a convenient way. Use '#' to select the next channel and use '*' to exit
Limit scanning to a channel GROUP by setting the option group argument.

14.165 ZapSendKeypadFacility

14.165.1 Synopsis

Send digits out of band over a PRI

14.165.2 Description

ZapSendKeypadFacility(): This application will send the given string of digits i
IE over the current channel.