

Asterisk Reference Information  
Version 1.8.0

Asterisk Development Team  
Asterisk.org

October 21, 2010

# Contents

<b>1</b>	<b>Introduction</b>	<b>11</b>
1.1	License Information . . . . .	11
1.1.1	Hold Music . . . . .	12
1.2	Security . . . . .	13
1.2.1	Introduction . . . . .	13
1.2.2	Network Security . . . . .	13
1.2.3	Dialplan Security . . . . .	13
1.2.4	Log Security . . . . .	14
1.3	Hardware . . . . .	14
1.3.1	Introduction . . . . .	14
1.3.2	DAHDI compatible hardware . . . . .	15
1.3.3	Non-DAHDI compatible hardware . . . . .	16
1.3.4	mISDN compatible hardware . . . . .	16
1.3.5	Miscellaneous other interfaces . . . . .	17
<b>2</b>	<b>Configuration</b>	<b>18</b>
2.1	General Configuration Information . . . . .	18
2.1.1	Configuration Parser . . . . .	18
2.1.2	Asterisk.conf . . . . .	22
2.1.3	CLI Prompt . . . . .	25
2.1.4	Extensions . . . . .	26
2.1.5	IP Quality of Service . . . . .	27
2.1.6	MP3 Support . . . . .	31
2.1.7	ICES . . . . .	31
2.2	Database Support . . . . .	31

2.2.1	Realtime Database Configuration . . . . .	31
2.2.2	FreeTDS . . . . .	35
2.3	Privacy . . . . .	35
2.3.1	First of all . . . . .	35
2.3.2	Next, Fight against autodialers!! . . . . .	35
2.3.3	Next, Fight against the empty CALLERID! . . . . .	35
2.3.4	Next, use a WELCOME MENU ! . . . . .	36
2.3.5	Next: Torture Them! . . . . .	37
2.3.6	Using Call Screening . . . . .	38
2.3.7	The 'N' and 'n' options . . . . .	39
2.3.8	Recorded Introductions . . . . .	40
<b>3</b>	<b>Channel Variables</b>	<b>43</b>
3.1	Introduction . . . . .	43
3.2	Parameter Quoting . . . . .	43
3.3	Variables . . . . .	44
3.4	Variable Inheritance . . . . .	44
3.4.1	Example . . . . .	45
3.5	Selecting Characters from Variables . . . . .	45
3.6	Expressions . . . . .	46
3.6.1	Spaces Inside Variables Values . . . . .	46
3.6.2	Operators . . . . .	47
3.6.3	Floating Point Numbers . . . . .	49
3.6.4	Functions . . . . .	50
3.6.5	Examples . . . . .	51
3.6.6	Numbers Vs. Strings . . . . .	53
3.6.7	Conditionals . . . . .	53
3.6.8	Parse Errors . . . . .	54
3.6.9	NULL Strings . . . . .	54
3.6.10	Warning . . . . .	54
3.6.11	Incompatibilities . . . . .	55
3.6.12	Debugging Hints . . . . .	56
3.7	Asterisk standard channel variables . . . . .	58
3.7.1	Application return values . . . . .	59
3.7.2	Various application variables . . . . .	60
3.7.3	The MeetMe Conference Bridge . . . . .	61
3.7.4	The VoiceMail() application . . . . .	61
3.7.5	The VMAuthenticate() application . . . . .	61

3.7.6	DUNDiLookup()	62
3.7.7	chan_dahdi	62
3.7.8	chan_sip	62
3.7.9	chan_agent	62
3.7.10	The Dial() application	63
3.7.11	The chanisavail() application	63
3.7.12	Dialplan Macros	63
3.7.13	The ChanSpy() application	64
3.7.14	OSP	64
3.7.15	Digit manipulation	65
<b>4</b>	<b>AEL: Asterisk Extension Language</b>	<b>66</b>
4.1	Introduction	66
4.2	Asterisk in a Nutshell	67
4.2.1	Contexts	67
4.2.2	Extensions and priorities	67
4.2.3	Macros	68
4.2.4	Applications	68
4.3	Getting Started	69
4.4	Debugging	69
4.5	About "aelparse"	69
4.6	General Notes about Syntax	70
4.7	Keywords	71
4.8	Procedural Interface and Internals	73
4.8.1	AEL version 2 BNF	73
4.9	AEL Example USAGE	76
4.9.1	Comments	76
4.9.2	Context	77
4.9.3	Extensions	77
4.9.4	Includes	78
4.9.5	#include	79
4.9.6	Dialplan Switches	79
4.9.7	Ignorepat	79
4.9.8	Variables	79
4.9.9	Local Variables	80
4.9.10	Loops	81
4.9.11	Conditionals	81
4.9.12	Break, Continue, and Return	83

4.9.13	goto, jump, and labels . . . . .	83
4.9.14	Macros . . . . .	85
4.10	Examples . . . . .	85
4.11	Semantic Checks . . . . .	86
4.12	Differences with the original version of AEL . . . . .	88
4.13	Hints and Bugs . . . . .	90
4.14	The Full Power of AEL . . . . .	91
<b>5</b>	<b>SLA: Shared Line Appearances</b>	<b>92</b>
5.1	Introduction . . . . .	92
5.2	Configuration . . . . .	92
5.2.1	Summary . . . . .	92
5.2.2	Dialplan . . . . .	92
5.2.3	Trunks . . . . .	93
5.2.4	Stations . . . . .	94
5.3	Configuration Examples . . . . .	95
5.3.1	Basic SLA . . . . .	95
5.3.2	SLA and Voicemail . . . . .	96
5.4	Call Handling . . . . .	98
5.4.1	Summary . . . . .	98
5.4.2	Station goes off hook (not ringing) . . . . .	98
5.4.3	Station goes off hook (ringing) . . . . .	98
5.4.4	Line button on a station is pressed . . . . .	99
<b>6</b>	<b>Channel Drivers</b>	<b>100</b>
6.1	IAX2 . . . . .	100
6.1.1	Introduction . . . . .	100
6.1.2	Why IAX2? . . . . .	100
6.1.3	Configuration . . . . .	102
6.1.4	IAX2 Jitterbuffer . . . . .	102
6.2	mISDN . . . . .	105
6.2.1	Introduction . . . . .	105
6.2.2	Features . . . . .	105
6.2.3	Fast Installation Guide . . . . .	106
6.2.4	Pre-Requisites . . . . .	106
6.2.5	Configuration . . . . .	106
6.2.6	mISDN CLI commands . . . . .	109
6.2.7	mISDN Variables . . . . .	109

6.2.8	Debugging and sending bug reports . . . . .	110
6.2.9	Examples . . . . .	110
6.2.10	Known Problems . . . . .	110
6.3	Local . . . . .	110
6.3.1	Introduction . . . . .	110
6.3.2	Examples . . . . .	111
6.3.3	Trivial Local channel example . . . . .	111
6.3.4	Delay dialing devices . . . . .	113
6.3.5	Dialing destinations with different information . . . . .	114
6.3.6	Using callfiles and Local channels . . . . .	115
6.3.7	Understanding When To Use /n . . . . .	117
6.3.8	Local channel modifiers . . . . .	120
6.4	Mobile . . . . .	121
6.4.1	Introduction . . . . .	121
6.4.2	Features . . . . .	121
6.4.3	Requirements . . . . .	122
6.4.4	Concepts . . . . .	122
6.4.5	Configuring chan_mobile . . . . .	122
6.4.6	Using chan_mobile . . . . .	123
6.4.7	Dialplan hints . . . . .	124
6.4.8	MobileStatus Application . . . . .	125
6.4.9	SMS Sending / Receiving . . . . .	125
6.4.10	DTMF Debouncing . . . . .	126
6.4.11	Debugging . . . . .	126
<b>7</b>	<b>Distributed Universal Number Discovery (DUNDi)</b>	<b>128</b>
7.1	Introduction . . . . .	128
7.2	DUNDIQUERY and DUNDIRESLT . . . . .	129
7.3	Peering Agreement . . . . .	129
<b>8</b>	<b>ENUM</b>	<b>143</b>
8.1	The ENUMLOOKUP dialplan function . . . . .	143
8.1.1	Arguments . . . . .	144
8.1.2	Examples . . . . .	145
8.1.3	Usage notes and subtle features . . . . .	147
8.1.4	Some more Examples . . . . .	149

<b>9</b>	<b>AMI: Asterisk Manager Interface</b>	<b>151</b>
9.1	The Asterisk Manager TCP/IP API . . . . .	151
9.2	Command Syntax . . . . .	152
9.3	Manager commands . . . . .	152
9.4	Examples . . . . .	152
9.5	Ensuring all modules are loaded . . . . .	153
9.6	Device status reports . . . . .	154
9.7	Some standard AMI headers . . . . .	154
9.8	Asynchronous Javascript Asterisk Manger (AJAM) . . . . .	158
9.8.1	Setup the Asterisk HTTP server . . . . .	158
9.8.2	Allow Manager Access via HTTP . . . . .	158
9.8.3	Integration with other web servers . . . . .	159
<b>10</b>	<b>CDR: Call Detail Records</b>	<b>160</b>
10.1	Applications . . . . .	160
10.2	CDR Fields . . . . .	160
10.3	Variables . . . . .	162
10.4	Storage Backends . . . . .	163
10.4.1	Microsoft SQL Server . . . . .	163
10.4.2	MySQL . . . . .	166
10.4.3	PostgreSQL . . . . .	166
10.4.4	SQLite 2 . . . . .	167
10.4.5	SQLite 3 . . . . .	167
10.4.6	RADIUS . . . . .	167
<b>11</b>	<b>CEL: Channel Event Logging</b>	<b>172</b>
11.1	Design Goals . . . . .	172
11.2	Events & Fields . . . . .	188
11.3	Applications & Functions . . . . .	189
11.3.1	CEL Function . . . . .	189
11.3.2	CELGenUserEvent Application . . . . .	190
11.4	Configuration Files . . . . .	190
11.5	Generating Billing Information . . . . .	191
11.6	Storage Backends . . . . .	191
11.6.1	Microsoft SQL Server . . . . .	191
11.6.2	MySQL . . . . .	195
11.6.3	PostgreSQL . . . . .	195
11.6.4	SQLite 3 . . . . .	196

11.6.5	RADIUS . . . . .	196
<b>12</b>	<b>Voicemail</b>	<b>201</b>
12.1	ODBC Storage . . . . .	201
12.2	IMAP Storage . . . . .	202
12.2.1	Installation Notes . . . . .	202
12.2.2	Modify voicemail.conf . . . . .	204
12.2.3	IMAP Folders . . . . .	205
12.2.4	Separate vs. Shared Email Accounts . . . . .	205
12.2.5	IMAP Server Implementations . . . . .	205
12.2.6	Quota Support . . . . .	206
12.2.7	Application Notes . . . . .	206
<b>13</b>	<b>SMS</b>	<b>208</b>
13.1	Introduction . . . . .	208
13.2	Background . . . . .	208
13.3	Typical use with Asterisk . . . . .	209
13.4	Terminology . . . . .	209
13.5	Sub address . . . . .	210
13.6	extensions.conf . . . . .	210
13.7	Using smsq . . . . .	211
13.8	File formats . . . . .	217
13.9	Delivery reports . . . . .	219
<b>14</b>	<b>Queues</b>	<b>220</b>
14.1	Introduction . . . . .	220
14.2	Configuring Call Queues . . . . .	220
14.2.1	queues.conf . . . . .	220
14.2.2	Routing incoming Calls to Queues . . . . .	221
14.2.3	Assigning agents to Queues . . . . .	224
14.2.4	Controlling The Way Queues Call the Agents . . . . .	227
14.2.5	Pre Acknowledgement Message . . . . .	229
14.2.6	Caveats . . . . .	229
14.3	Queue Logs . . . . .	230
<b>15</b>	<b>Phone Provisioning</b>	<b>233</b>
15.1	Introduction . . . . .	233
15.2	Configuration of phoneprov.conf . . . . .	233

15.2.1	The [general] section . . . . .	233
15.2.2	Creating phone profiles . . . . .	234
15.3	Configuration of users.conf . . . . .	235
15.3.1	The [general] section . . . . .	235
15.3.2	Individual Users . . . . .	236
15.4	Templates . . . . .	236
15.5	Putting it all together . . . . .	238
<b>16</b>	<b>Calendaring</b>	<b>240</b>
16.1	Introduction . . . . .	240
16.2	Configuring Asterisk Calendaring . . . . .	240
16.2.1	Module-independent settings . . . . .	241
16.2.2	Module-dependent settings . . . . .	242
16.3	Dialplan functions . . . . .	242
16.3.1	Read functions . . . . .	242
16.3.2	Write functions . . . . .	243
16.4	Dialplan Examples . . . . .	243
16.4.1	Office hours . . . . .	243
16.4.2	Meeting reminders . . . . .	244
16.4.3	Writing an event . . . . .	244
<b>17</b>	<b>Security Framework</b>	<b>245</b>
17.1	Introduction . . . . .	245
17.2	Framework Overview . . . . .	246
17.2.1	Security Event Generation . . . . .	246
17.2.2	Security Event Logger . . . . .	246
17.3	Events to Log . . . . .	247
17.4	Security Log File Format . . . . .	248
<b>18</b>	<b>Secure Calls</b>	<b>252</b>
18.1	Introduction . . . . .	252
18.2	Channel-specific configuration . . . . .	252
18.3	Security-based dialplan branching . . . . .	253
18.4	Forcing bridged channels to be secure . . . . .	253
<b>19</b>	<b>Manipulating Party ID Information</b>	<b>254</b>
19.1	Introduction . . . . .	254
19.2	Tools available . . . . .	255

19.2.1	CALLERID dialplan function . . . . .	255
19.2.2	CONNECTEDLINE dialplan function . . . . .	255
19.2.3	REDIRECTING dialplan function . . . . .	256
19.2.4	Special REDIRECTING considerations for ISDN . . .	257
19.2.5	Dial() and Queue() dialplan application 'P' option . . .	257
19.2.6	Interception macros . . . . .	257
19.3	Manipulation examples . . . . .	258
19.3.1	Simple recording playback . . . . .	259
19.3.2	Straightforward dial through . . . . .	259
19.3.3	Use of interception macro . . . . .	259
19.3.4	Simple redirection . . . . .	260
19.4	Ideas for usage . . . . .	260
19.5	Troubleshooting tips . . . . .	261
19.6	For further reading... . . . . .	261
<b>20</b>	<b>Call Completion Supplementary Services</b>	<b>262</b>
20.1	Introduction . . . . .	262
20.2	What is CCSS? . . . . .	262
20.3	Glossary of Terms . . . . .	263
20.4	The CC Process . . . . .	264
20.4.1	The Initial Call . . . . .	264
20.4.2	Requesting CC . . . . .	264
20.4.3	Cancelling CC . . . . .	265
20.4.4	Monitoring the Called Party . . . . .	265
20.4.5	Alerting the Caller . . . . .	266
20.4.6	If the Caller is unavailable . . . . .	266
20.4.7	The CC recall . . . . .	266
20.5	Miscellaneous Information and Tips . . . . .	267
20.6	Simple Example of generic call completion . . . . .	270
<b>21</b>	<b>Packet Loss Concealment</b>	<b>272</b>
21.1	What is PLC? . . . . .	272
21.2	How does Asterisk detect packet loss? . . . . .	272
21.3	A bit of background on translation . . . . .	273
21.4	Additional restrictions and caveats . . . . .	274
21.5	Summary . . . . .	275
21.6	Protip . . . . .	275

<b>22</b>	<b>Sounds Packages</b>	<b>277</b>
22.1	Introduction . . . . .	277
22.2	Getting The Sounds Tools . . . . .	277
22.3	About The Sounds Tools . . . . .	278
22.3.1	audiofilter . . . . .	278
22.3.2	makeg722 . . . . .	278
22.3.3	scripts . . . . .	279
<b>23</b>	<b>Development</b>	<b>280</b>
23.1	Backtrace . . . . .	280

# 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. These components are also distributed under the GPL version 2 as well.

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

If you wish to use our code in other GPL programs, don't worry – there is no requirement that you provide the same exception in your GPL'd prod-

ucts (although if you've written a module for Asterisk we would strongly encourage you to make the same exception that we do).

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

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

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

<http://www.digium.com/en/company/profile/trademarkpolicy.php>

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

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

Digium, Inc. 445 Jan Davis Drive Huntsville, AL 35806 USA

### **1.1.1 Hold Music**

Digium has licensed the music included with the Asterisk distribution From opsound.org 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(DAHDIG2/${EXTEN:1})
include => local

[local]
exten => _9NXXNXXX,1,Dial(DAHDIG2/${EXTEN:1})
include => default

[default]
exten => 6123,Dial(DAHDIG1)
```

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, etc.), but you can also talk to the real PSTN through various cards.

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

### 1.3.2 DAHDI 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<sup>TM</sup> 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-DAHDI compatible hardware

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

### 1.3.4 mISDN compatible hardware

mISDN homepage: <http://www.misdn.org/>

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, dahdi.conf and agents.conf), the syntax is a bit different. In these files the syntax is as follows:

```
[section]
label1 = value1
label2 = value2
object => name
```

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

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

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

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

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

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

## Comments

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

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

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

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

## Including other files

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

```
#include myusers.conf
```

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

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

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

The exec directive is used like this:

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

### Adding to an existing section

```
[section]
label = value

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

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

### Defining a template-only section

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

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

### Using templates (or other configuration sections)

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

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

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

```

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

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

```

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

```

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

```

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

## Additional Examples

(in top-level sip.conf)

```

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

#include accounts/*/sip.conf

```

(in accounts/customer1/sip.conf)

```

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

```

```

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

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

```

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

## 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

; Language Options
documentation_language = en | es | ru

; These options have no command line equivalent

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

; Build transcode paths via SLINEAR
transcode_via_sln = yes | no

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

```

```

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

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

; Stop accepting calls when free memory falls below this amount specified in MB
minmemfree = 256

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

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

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

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

; Locking mode for voicemail
; - lockfile: default, for normal use
; - flock: for where the lockfile locking method doesn't work
;       eh. on SMB/CIFS mounts
lockmode = lockfile | flock

; Entity ID. This is in the form of a MAC address. It should be universally
; unique. It must be unique between servers communicating with a protocol
; that uses this value. The only thing that uses this currently is DUNDi,
; but other things will use it in the future.
; entityid=00:11:22:33:44:55

[files]
; Changing the following lines may compromise your security
; Asterisk.ctl is the pipe that is used to connect the remote CLI
; (asterisk -r) to Asterisk. Changing these settings change the
; permissions and ownership of this file.
; The file is created when Asterisk starts, in the "astrundir" above.

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

```

## 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
- `%u` - Username
- `%g` - Groupname
- `%%` - Percent sign
- `%#` - '#' if Asterisk is run in console mode, '>' if running as remote console
- `%Cn[;n]` - Change terminal foreground (and optional background) color to specified A full list of colors may be found in `include/asterisk/term.h`

On systems which implement `getloadavg(3)`, you may also use:

- `%l1` - Load average over past minute
- `%l2` - Load average over past 5 minutes
- `%l3` - Load average over past 15 minutes

## 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 "dialplan reload" to load the new dialplan without disrupting service in your PBX.

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

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

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

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

### Example dialplan

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

### Special extensions

There are some extensions with important meanings:

- s
  - What to do when an extension context is entered (unless overridden by the low level channel interface) This is used in macros,

and some special cases. "s" is not a generic catch-all wildcard extension.

- i
  - What to do if an invalid extension is entered
- h
  - The hangup extension, executed at hangup
- t
  - What to do if nothing is entered in the requisite amount of time.
- T
  - This is the extension that is executed when the 'absolute' timeout is reached. See "core show function TIMEOUT" for more information on setting timeouts.
- e
  - This extension will substitute as a catchall for any of the 'i', 't', or 'T' extensions, if any of them do not exist and catching the error in a single routine is desired. The function EXCEPTION may be used to query the type of exception or the location where it occurred.

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

## 2.1.5 IP Quality of Service

### Introduction

Asterisk supports different QoS settings at the application level for various protocols on both signaling and media. The Type of Service (TOS) byte can be set on outgoing IP packets for various protocols. The TOS byte is used

by the network to provide some level of Quality of Service (QoS) even if the network is congested with other traffic.

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

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

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

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

Table 2.1: Channel Driver QoS Settings

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

Table 2.2: Other ToS Settings

## IP TOS values

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

The tos\* parameters also take numeric values.

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

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

### 802.1p CoS values

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

### Recommended values

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

	tos	cos
Signaling	cs3	3
Audio	ef	5
Video	af41	4
Text	af41	3
Other	ef	

Table 2.3: Recommended QoS Settings

### IAX2

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

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

## SIP

In sip.conf, there are four parameters that control the TOS settings: "tos\_sip", "tos\_audio", "tos\_video" and "tos\_text". tos\_sip controls what TOS SIP call signaling packets are set to. tos\_audio, tos\_video and tos\_text control what TOS values are used for RTP audio, video, and text packets, respectively.

There are four parameters to control 802.1p CoS: "cos\_sip", "cos\_audio", "cos\_video" and "cos\_text". The behavior of these parameters is the same as for the SIP TOS settings described above.

## Other RTP channels

chan\_mgcp, chan\_h323, chan\_skinny and chan\_unistim also support TOS and CoS via setting tos and cos parameters in their corresponding configuration files. Naming style and behavior are the same as for chan\_sip.

## Reference

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

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

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

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

For more information on Quality of Service for VoIP networks see the "Enterprise QoS Solution Reference Network Design Guide" version 3.3 from Cisco at: [http://www.cisco.com/application/pdf/en/us/guest/netsol/ns432/c649/ccmigration\\_09186a008049b062.pdf](http://www.cisco.com/application/pdf/en/us/guest/netsol/ns432/c649/ccmigration_09186a008049b062.pdf)

## 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](http://svn.digium.com) or in the asterisk-addons release at <http://downloads.asterisk.org/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.

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](http://www.unixodbc.org) for more information.
- MySQL: Native support for MySQL, integrated into Asterisk
- PostgreSQL: Native support for Postgres, integrated into Asterisk

### **Two modes: Static and Realtime**

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

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

You just have to always make sure the `var_metric` values are properly set and ordered as you expect in your database server if you're using the static mode with ARA (either sequentially or with the same `var_metric` value for everybody).

If you have an option that depends on another one in a given configuration file (i.e, 'musiconhold' depending on 'agent' from `agents.conf`) but their `var_metric` are not sequential you'll probably get default values being assigned for those options instead of the desired ones. You can still use the same `var_metric` for all entries in your DB, just make sure the entries are recorded in an order that does not break the option dependency.

That doesn't happen when you use a static file in the file system. Although this might be interpreted as a bug or limitation, it is not.

### **Realtime SIP friends**

The SIP realtime objects are users and peers that are loaded in memory when needed, then deleted. This means that Asterisk currently can't handle voicemail notification and NAT keepalives for these peers. Other than that, most of the functionality works the same way for realtime friends as for the ones in static configuration.

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

If you specify a separate family called "sipregs" SIP registration data will be stored in that table and not in the "sippeers" table.

### **Realtime H.323 friends**

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

### **New function in the dial plan: The Realtime Switch**

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

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

### **Capabilities**

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

```
asterisk -rx "reload"
```

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

### **Configuration in extconfig.conf**

You configure the ARA in extconfig.conf (yes, it's a strange name, but is was defined in the early days of the realtime architecture and kind of stuck).

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

```
<family> => <realtime driver>,<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)

Voicemail storage with the support of ODBC described in file `docs/odbcstorage.tex` (12.1).

### **Limitations**

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

### **FreeTDS supported with connection pooling**

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

You may notice a performance issue under high load using UnixODBC. The UnixODBC driver supports threading but you must specifically enable threading within the UnixODBC configuration file like below for each engine:

```
Threading = 2
```

This will enable the driver to service many requests at a time, rather than serially.

## 2.2.2 FreeTDS

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

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

## 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

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 application can help here. It will ask the caller to enter a 10-digit phone number. They get 3 tries(configurable), and this is configurable, with control being passed to next priority where you can check the channelvariable PRIVACYMGRSTATUS. If the callerid was valid this variable will have the value SUCCESS, otherwise it will have the value FAILED.

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

PrivacyManager can be told about a context where you can have patterns that describe valid phone numbers. If none of the patterns match the input, it will be considered a non-valid phonenumber and the user can try again until the retry counter is reached. This helps in resolving the issues stated in the previous paragraph.

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

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

### **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
    ; do this if they don't enter a number to Privacy Manager
exten => s,5,GotoIf($[ "${PRIVACYMGRSTATUS}" = "FAILED" ]?s,105)
exten => s,6,GotoIf($[ "${CALLERID(num)}" = "7773334444" & "${CALLERID(name)}" : "Privacy Manager" ]?callerid-liar,
exten => s,7,Dial(SIP/yourphone)
exten => s,105,Background(tt-allbusy)
exten => s,106,Background(tt-somethingwrong)
exten => s,107,Background(tt-monkeysintro)
exten => s,108,Background(tt-monkeys)
exten => s,109,Background(tt-weasels)
exten => s,110,Hangup

```

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

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

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

### 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(DAHDI/5r3&DAHDI/6r3,35,tmpA(beep))
      or
exten => 3,3,Dial(DAHDI/5r3&DAHDI/6r3,35,tmp(something)A(beep))
      or
exten => 3,3,Dial(DAHDI/5r3&DAHDI/6r3,35,tmpA(beep))
```

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

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

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

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

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

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

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

### 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,6,Set(PATH=/var/lib/asterisk/sounds/priv-callerintros)
exten => s,7,System(/usr/bin/play ${PATH}/${CALLERID(num)}.gsm&,0)
```

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

(Note: the ,0 option at the end of the System command above, is a local mod I made to the System command. It forces a 0 result code to be returned, whether the play command successfully completed or not. Therefore, I don't

have to ensure that the file exists or not. While I've turned this mod into the developers, it hasn't been incorporated yet. You might want to write an AGI or shell script to handle it a little more intelligently)

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

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

[home-introduction-record]
exten => s,1,Background(intro-record-choices) ;; Script:
    ;; If you want some advice about recording your
    ;; introduction, dial 1.
    ;; otherwise, dial 2, and introduce yourself after
    ;; the beep.
exten => 1,1,Playback(intro-record)
    ;; Your introduction should be short and sweet and crisp.
    ;; Your introduction will be limited to 4 seconds.
    ;; This is NOT meant to be a voice mail message, so
    ;; please, don't say anything about why you are calling.
    ;; After we are done making the recording, your introduction
    ;; will be saved for playback.
    ;; If you are the only person that would call from this number,
    ;; please state your name. Otherwise, state your business
    ;; or residence name instead. For instance, if you are
    ;; friend of the family, say, Olie McPherson, and both
    ;; you and your kids might call here a lot, you might
```

```

;; say: "This is the distinguished Olie McPherson Residence!"
;; If you are the only person calling, you might say this:
;; "This is the illustrious Kermit McFrog! Pick up the Phone, someone!!"
;; If you are calling from a business, you might pronounce a more sedate introduction,like,
;; "Fritz from McDonalds calling.", or perhaps the more original introduction:
;; "John, from the Park County Morgue. You stab 'em, we slab 'em!".
;; Just one caution: the kids will hear what you record every time
;; you call. So watch your language!
;; I will begin recording after the tone.
;; When you are done, hit the # key. Gather your thoughts and get
;; ready. Remember, the # key will end the recording, and play back
;; your intro. Good Luck, and Thank you!"
exten => 1,2,Goto(2,1)
exten => 2,1,Background(intro-start)
    ;; OK, here we go! After the beep, please give your introduction.
exten => 2,2,Background(beep)
exten => 2,3,Record(priv-callerintros/${CALLERID(num)}:gsm,4)
exten => 2,4,Background(priv-callerintros/${CALLERID(num)})
exten => 2,5,Goto(home-introduction,s,1)
exten => t,1,Goto(s,1)
exten => i,1,Background(invalid)
exten => i,2,Goto(s,1)
exten => o,1,Goto(s,1)

```

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

# 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($[ "${CALLERID(name)}" : "Privacy Manager" ]?callerid-liar,s,1:s,7)
```

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

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

and will evaluate to 0.

The above without double quotes would have evaluated to:

```
DELOREAN MOTORS : Privacy Manager
```

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

### 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 floating point comparison if both arguments are numbers; otherwise, returns the results of string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relation is true, or 0 if the relation is false.
- `expr1 {+, -} expr2`  
Return the results of addition or subtraction of floating point-valued arguments.

- `expr1 {*, /, %} expr2`  
Return the results of multiplication, floating point division, or remainder of arguments.
- `- expr1`  
Return the result of subtracting `expr1` from 0. This, the unary minus operator, is right associative, and has the same precedence as the `!` operator.
- `! expr1`  
Return the result of a logical complement of `expr1`. In other words, if `expr1` is null, 0, an empty string, or the string "0", return a 1. Otherwise, return a 0. It has the same precedence as the unary minus operator, and is also right associative.
- `expr1 : expr2`  
The `':'` operator matches `expr1` against `expr2`, which must be a regular expression. The regular expression is anchored to the beginning of the string with an implicit `^`.  
  
If the match succeeds and the pattern contains at least one regular expression subexpression `'...'`, the string corresponding to `'\1'` is returned; otherwise the matching operator returns the number of characters matched. If the match fails and the pattern contains a regular expression subexpression the null string is returned; otherwise 0.  
  
Normally, the double quotes wrapping a string are left as part of the string. This is disastrous to the `:` operator. Therefore, before the regex match is made, beginning and ending double quote characters are stripped from both the pattern and the string.
- `expr1 =~ expr2`  
Exactly the same as the `':'` operator, except that the match is not anchored to the beginning of the string. Pardon any similarity to seemingly similar operators in other programming languages! The `':'` and `"=~"` operators share the same precedence.
- `expr1 ? expr2 :: expr3`

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

- `expr1 ~ ~ expr2`

Concatenation operator. The two `exprs` are evaluated and turned into strings, stripped of surrounding double quotes, and are turned into a single string with no intervening spaces. This operator is new to trunk after 1.6.0; it is not needed in existing `extensions.conf` code. Because of the way asterisk evaluates `and[]` constructs (recursively, bottom-up), no `or` is ever present when the contents of a `or[]` is evaluated. Thus, tokens are usually already merged at evaluation time. But, in AEL, various `exprs` are evaluated raw, and `and[]` are gathered and treated as tokens. And in AEL, no two tokens can sit side by side without an intervening operator. So, in AEL, concatenation must be explicitly specified in expressions. This new operator will play well into future plans, where expressions (`constructs`, `andvariableferences`(`constructs`) are merged into a single grammar.

Parentheses are used for grouping in the usual manner.

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

### 3.6.3 Floating Point Numbers

In 1.6 and above, we shifted the `$[...]` expressions to be calculated via floating point numbers instead of integers. We use 'long double' numbers when possible, which provide around 16 digits of precision with 12 byte numbers.

To specify a floating point constant, the number has to have this format: D.D, where D is a string of base 10 digits. So, you can say 0.10, but you can't say .10 or 20.– we hope this is not an excessive restriction!

Floating point numbers are turned into strings via the `'%g'/'%Lg'` format of the `printf` function set. This allows numbers to still 'look' like integers to those counting on integer behavior. If you were counting on 1/4 evaluating to

0, you need to now say `TRUNC(1/4)`. For a list of all the truncation/rounding capabilities, see the next section.

### 3.6.4 Functions

In 1.6 and above, we upgraded the `$[]` expressions to handle floating point numbers. Because of this, folks counting on integer behavior would be disrupted. To make the same results possible, some rounding and integer truncation functions have been added to the core of the Expr2 parser. Indeed, dialplan functions can be called from `$[.]` expressions without the `${...}` operators. The only trouble might be in the fact that the arguments to these functions must be specified with a comma. If you try to call the MATH function, for example, and try to say `3 + MATH(7*8)`, the expression parser will evaluate `7*8` for you into 56, and the MATH function will most likely complain that its input doesn't make any sense.

We also provide access to most of the floating point functions in the C library. (but not all of them).

While we don't expect someone to want to do Fourier analysis in the dialplan, we don't want to preclude it, either.

Here is a list of the 'builtin' functions in Expr2. All other dialplan functions are available by simply calling them (read-only). In other words, you don't need to surround function calls in `$[...]` expressions with `${...}`. Don't jump to conclusions, though! – you still need to wrap variable names in curly braces!

1. `COS(x)` x is in radians. Results vary from -1 to 1.
2. `SIN(x)` x is in radians. Results vary from -1 to 1.
3. `TAN(x)` x is in radians.
4. `ACOS(x)` x should be a value between -1 and 1.
5. `ASIN(x)` x should be a value between -1 and 1.
6. `ATAN(x)` returns the arc tangent in radians; between  $-\pi/2$  and  $\pi/2$ .
7. `ATAN2(x,y)` returns a result resembling  $y/x$ , except that the signs of both args are used to determine the quadrant of the result. Its result is in radians, between  $-\pi$  and  $\pi$ .

8. POW(x,y) returns the value of x raised to the power of y.
9. SQRT(x) returns the square root of x.
10. FLOOR(x) rounds x down to the nearest integer.
11. CEIL(x) rounds x up to the nearest integer.
12. ROUND(x) rounds x to the nearest integer, but round halfway cases away from zero.
13. RINT(x) rounds x to the nearest integer, rounding halfway cases to the nearest even integer.
14. TRUNC(x) rounds x to the nearest integer not larger in absolute value.
15. REMAINDER(x,y) computes the remainder of dividing x by y. The return value is  $x - n*y$ , where n is the value  $x/y$ , rounded to the nearest integer. If this quotient is  $1/2$ , it is rounded to the nearest even number.
16. EXP(x) returns e to the x power.
17. EXP2(x) returns 2 to the x power.
18. LOG(x) returns the natural logarithm of x.
19. LOG2(x) returns the base 2 log of x.
20. LOG10(x) returns the base 10 log of x.

### 3.6.5 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 addition).

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

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

(3+8)/2
  returns 5.5 now.

TRUNC((3+8)/2)
  returns 5.

FLOOR(2.5)
  returns 2

FLOOR(-2.5)
  returns -3

CEIL(2.5)
  returns 3.

CEIL(-2.5)
  returns -2.

ROUND(2.5)
  returns 3.

ROUND(3.5)
  returns 4.

ROUND(-2.5)
  returns -3

RINT(2.5)
  returns 2.

RINT(3.5)
  returns 4.

RINT(-2.5)
  returns -2.

RINT(-3.5)
  returns -4.

```

```
TRUNC(2.5)
  returns 2.

TRUNC(3.5)
  returns 3.

TRUNC(-3.5)
  returns -3.
```

Of course, all of the above examples use constants, but would work the same if any of the numeric or string constants were replaced with a variable reference `#{CALLERID(num)}`, for instance.

### 3.6.6 Numbers Vs. Strings

Tokens consisting only of numbers are converted to 'long double' if possible, which are from 80 bits to 128 bits depending on the OS, compiler, and hardware. This means that overflows can occur when the numbers get above 18 digits (depending on the number of bits involved). Warnings will appear in the logs in this case.

### 3.6.7 Conditionals

There is one conditional application - the conditional goto :

```
exten => 1,2,GotoIf(condition?label1:label2)
```

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

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

```
exten => 1,2,GotoIf(${#{CALLERID(all)} = 123456}?2,1:3,1)
```

Example of use :

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

### 3.6.8 Parse Errors

Syntax errors are now output with 3 lines.

If the extensions.conf file contains a line like:

```
exten => s,6,GotoIf($[ "${CALLERID(num)}" = "3071234567" & & "${CALLERID(name)}" : "Privacy Manager" ]?callerid-1
```

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

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

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

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

### 3.6.9 NULL Strings

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

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

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

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

### 3.6.10 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.

### 3.6.11 Incompatibilities

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

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

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

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

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

1. Tokens separated by space(s). Previously, tokens were separated by spaces. Thus, ' 1 + 1 ' would evaluate to the value '2', but '1+1' would evaluate to the string '1+1'. If this behavior was depended on, then the expression evaluation will break. '1+1' will now evaluate to '2', and something is not going to work right. To keep such strings from being evaluated, simply wrap them in double quotes: ' "1+1" '
2. The colon operator. In versions previous to double quoting, the colon operator takes the right hand string, and using it as a regex pattern, looks for it in the left hand string. It is given an implicit  $\hat{}$  operator at the beginning, meaning the pattern will match only at the beginning of the left hand string. If the pattern or the matching string had double quotes around them, these could get in the way of the pattern match. Now, the wrapping double quotes are stripped from both the pattern and the left hand string before applying the pattern. This was done because it recognized that the new way of scanning the expression doesn't use spaces to separate tokens, and the average regex expression is full of

operators that the scanner will recognize as expression operators. Thus, unless the pattern is wrapped in double quotes, there will be trouble. For instance, `{VAR1} : (Who|What*)+` may 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.

### 3.6.12 Debugging Hints

There are two utilities you can build to help debug the `$( )` in your `extensions.conf` file.

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

```
make testexpr2
```

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

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

```
testexpr2 '2*2+2/2'
```

is an example.

And, in the `utils` directory, you can say:

```
make check_expr
```

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

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

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

```
check_expr /etc/asterisk/extensions.conf CALLERID(num)=3075551212 DIALSTATUS=TORTURE EXTEN=121
```

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

```
check_expr /etc/asterisk/extensions.conf CALLERID(num)=3075551212 DIALSTATUS=TORTURE EXTEN:2=121
```

on stdout, you will see something like:

```
OK -- ${ "${DIALSTATUS}" = "TORTURE" | "${DIALSTATUS}" = "DONTCALL" } at line 416
```

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

```
line 416, evaluation of ${ "TORTURE" = "TORTURE" | "TORTURE" = "DONTCALL" } result: 1
```

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

## 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.

<code>\${CDR(accountcode)}</code>	* Account code (if specified)
<code>\${BLINDTRANSFER}</code>	The name of the channel on the other side of a blind transfer
<code>\${BRIDGEPEER}</code>	Bridged peer
<code>\${BRIDGEPVTCALLID}</code>	Bridged peer PVT call ID (SIP Call ID if a SIP call)
<code>\${CALLERID(ani)}</code>	* Caller ANI (PRI channels)
<code>\${CALLERID(ani2)}</code>	* ANI2 (Info digits) also called Originating line information
<code>\${CALLERID(all)}</code>	* Caller ID
<code>\${CALLERID(dnid)}</code>	* Dialed Number Identifier
<code>\${CALLERID(name)}</code>	* Caller ID Name only
<code>\${CALLERID(num)}</code>	* Caller ID Number only
<code>\${CALLERID(rdnis)}</code>	* Redirected Dial Number ID Service
<code>\${CALLINGANI2}</code>	* Caller ANI2 (PRI channels)
<code>\${CALLINGPRES}</code>	* Caller ID presentation for incoming calls (PRI channels)
<code>\${CALLINGTNS}</code>	* Transit Network Selector (PRI channels)
<code>\${CALLINGTON}</code>	* Caller Type of Number (PRI channels)
<code>\${CHANNEL}</code>	* Current channel name
<code>\${CONTEXT}</code>	* Current context
<code>\${DATETIME}</code>	* Current date time in the format: DDMMYYYY-HH:MM:SS (Deprecated; use <code>\${STRFTIME(\${EPOCH},,%d%m%Y-%H:%M:%S)}</code> )
<code>\${DB_RESULT}</code>	Result value of DB_EXISTS() dial plan function
<code>\${EPOCH}</code>	* Current unix style epoch
<code>\${EXTEN}</code>	* Current extension
<code>\${ENV(VAR)}</code>	Environmental variable VAR
<code>\${GOTO_ON_BLINDXFR}</code>	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)
<code>\${HANGUPCAUSE}</code>	* Asterisk cause of hangup (inbound/outbound)
<code>\${HINT}</code>	* Channel hints for this extension
<code>\${HINTNAME}</code>	* Suggested Caller*ID name for this extension

<code>\${INVALID_EXTEN}</code>	The invalid called extension (used in the "i" extension)
<code>\${LANGUAGE}</code>	* Current language (Deprecated; use <code>\${LANGUAGE()}</code> )
<code>\${LEN(VAR)}</code>	* String length of VAR (integer)
<code>\${PRIORITY}</code>	* Current priority in the dialplan
<code>\${PRIREDIRECTREASON}</code>	Reason for redirect on PRI, if a call was directed
<code>\${TIMESTAMP}</code>	* Current date time in the format: YYYYMMDD-HHMMSS (Deprecated; use <code>\${STRFTIME(\${EPOCH},,%Y%m%d-%H%M%S)}</code> )
<code>\${TRANSFER_CONTEXT}</code>	Context for transferred calls
<code>\${FORWARD_CONTEXT}</code>	Context for forwarded calls
<code>\${DYNAMIC_PEERNAME}</code>	The name of the channel on the other side when a dynamic feature is used.
<code>\${DYNAMIC_FEATURENAME}</code>	The name of the last triggered dynamic feature.
<code>\${UNIQUEID}</code>	* Current call unique identifier
<code>\${SYSTEMNAME}</code>	* value of the systemname option of asterisk.conf
<code>\${ENTITYID}</code>	* Global Entity ID set automatically, or from asterisk.conf

### 3.7.1 Application return values

Many applications return the result in a variable that you read to get the result of the application. These status fields are unique for each application. For the various status values, see each application's help text.

<code>\${AGISTATUS}</code>	* <code>agi()</code>
<code>\${AQMSTATUS}</code>	* <code>addqueuemember()</code>
<code>\${AVAILSTATUS}</code>	* <code>chanisavail()</code>
<code>\${CHECKGROUPSTATUS}</code>	* <code>checkgroup()</code>
<code>\${CHECKMD5STATUS}</code>	* <code>checkmd5()</code>
<code>\${CPLAYBACKSTATUS}</code>	* <code>controlplayback()</code>
<code>\${DIALSTATUS}</code>	* <code>dial()</code>
<code>\${DBGETSTATUS}</code>	* <code>dbget()</code>
<code>\${ENUMSTATUS}</code>	* <code>enumlookup()</code>
<code>\${HASVMSTATUS}</code>	* <code>hasnewvoicemail()</code>
<code>\${LOOKUPBLSTATUS}</code>	* <code>lookupblacklist()</code>
<code>\${OSPAUTHSTATUS}</code>	* <code>ospauth()</code>
<code>\${OSPLOOKUPSTATUS}</code>	* <code>osplookup()</code>
<code>\${OSPNEXTSTATUS}</code>	* <code>ospnext()</code>
<code>\${OSPFINISHSTATUS}</code>	* <code>ospfinish()</code>
<code>\${PARKEDAT}</code>	* <code>parkandannounce()</code>

<code>\${PLAYBACKSTATUS}</code>	* playback()
<code>\${PQMSTATUS}</code>	* pausequeuemember()
<code>\${PRIVACYMGRSTATUS}</code>	* privacymanager()
<code>\${QUEUESTATUS}</code>	* queue()
<code>\${RQMSTATUS}</code>	* removequeuemember()
<code>\${SENDIMAGESTATUS}</code>	* sendimage()
<code>\${SENDTEXTSTATUS}</code>	* sendtext()
<code>\${SENDURLSTATUS}</code>	* sendurl()
<code>\${SYSTEMSTATUS}</code>	* system()
<code>\${TRANSFERSTATUS}</code>	* transfer()
<code>\${TXTCIDNAMESTATUS}</code>	* txtcidname()
<code>\${UPQMSTATUS}</code>	* unpausequeuemember()
<code>\${VMSTATUS}</code>	* voicemail()
<code>\${VMBOXEXISTSSTATUS}</code>	* vmboxexists()
<code>\${WAITSTATUS}</code>	* waitforsilence()

### 3.7.2 Various application variables

<code>\${CURL}</code>	* Resulting page content for curl()
<code>\${ENUM}</code>	* Result of application EnumLookup
<code>\${EXITCONTEXT}</code>	Context to exit to in IVR menu (app background()) or in the RetryDial() application
<code>\${MONITOR}</code>	* Set to "TRUE" if the channel is/has been monitored (app
<code>\${MONITOR_EXEC}</code>	Application to execute after monitoring a call
<code>\${MONITOR_EXEC_ARGS}</code>	Arguments to application
<code>\${MONITOR_FILENAME}</code>	File for monitoring (recording) calls in queue
<code>\${QUEUE_PRIO}</code>	Queue priority
<code>\${QUEUE_MAX_PENALTY}</code>	Maximum member penalty allowed to answer caller
<code>\${QUEUE_MIN_PENALTY}</code>	Minimum member penalty allowed to answer caller
<code>\${QUEUESTATUS}</code>	Status of the call, one of: (TIMEOUT   FULL   JOINEMPTY   LEAVEEMPTY   JOINUNAVAIL   ...)
<code>\${QUEUEPOSITION}</code>	* When a caller is removed from a queue, his current position in this variable. If the value is 0, then this means that the caller was not serviced by a queue member. If non-zero, then this was the position in the queue the caller was in when he left.
<code>\${RECORDED_FILE}</code>	* Recorded file in record()
<code>\${TALK_DETECTED}</code>	* Result from talkdetect()
<code>\${TOUCH_MONITOR}</code>	The filename base to use with Touch Monitor (auto record)

\${TOUCH\_MONITOR\_PREF} \* The prefix for automonitor recording filenames.  
 \${TOUCH\_MONITOR\_FORMAT} The audio format to use with Touch Monitor (auto record)  
 \${TOUCH\_MONITOR\_OUTPUT} \* Recorded file from Touch Monitor (auto record)  
 \${TOUCH\_MONITOR\_MESSAGE\_START} Recorded file to play for both channels at start of  
 \${TOUCH\_MONITOR\_MESSAGE\_STOP} Recorded file to play for both channels at end of m  
 \${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 (DAHDI only)  
 \${MEETMESECS} \* Number of seconds a user participated in a MeetMe con  
 \${CONF\_LIMIT\_TIMEOUT\_FILE} File to play when time is up. Used with the L() opti  
 \${CONF\_LIMIT\_WARNING\_FILE} File to play as warning if 'y' is defined.  
 The default is to say the time remaining. Used with  
 \${MEETMEBOOKID} \* This variable exposes the bookid column for a realtim

### 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\_dahdi

`#{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\_specifi  
4=subscriber, 6=abbreviated, 7=reserved  
`#{CALLINGSUBADDR}` \* Caller's PRI Subaddress  
`#{FAXEXTEN}` \* The extension called before being redirected to "fax"  
`#{PRIREDIRECTREASON}` \* Reason for redirect, if a call was directed  
`#{SMDI_VM_TYPE}` \* When an call is received with an SMDI message, the 'type'  
of message 'b' or 'u'

### 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)  
`#{SIPFROMDOMAIN}` Set SIP domain on outbound calls  
`#{SIPUSERAGENT}` \* SIP user agent (deprecated)  
`#{SIPURI}` \* SIP uri  
`#{SIP_MAX_FORWARDS}` Set the value of the Max-Forwards header for outbound call  
`#{SIP_CODEC}` Set the SIP codec for an inbound call  
`#{SIP_CODEC_INBOUND}` Set the SIP codec for an inbound call  
`#{SIP_CODEC_OUTBOUND}` Set the SIP codec for an outbound call  
`#{SIP_URI_OPTIONS}` \* additional options to add to the URI for an outgoing call  
`#{RTPAUDIOQOS}` RTCP QoS report for the audio of this call  
`#{RTPVIDEOQOS}` RTCP QoS report for the video of this call

### 3.7.9 chan\_agent

`#{AGENTMAXLOGINTRIES}` Set the maximum number of failed logins  
`#{AGENTUPDATECDR}` Whether to update the CDR record with Agent channel data

<code>\${AGENTGOODBYE}</code>	Sound file to use for "Good Bye" when agent logs out
<code>\${AGENTACKCALL}</code>	Whether the agent should acknowledge the incoming call
<code>\${AGENTAUTOLOGOFF}</code>	Auto logging off for an agent
<code>\${AGENTWRAPUPTIME}</code>	Setting the time for wrapup between incoming calls
<code>\${AGENTNUMBER}</code>	* Agent number (username) set at login
<code>\${AGENTSTATUS}</code>	* Status of login ( fail   on   off )
<code>\${AGENTEXTEN}</code>	* Extension for logged in agent

### 3.7.10 The Dial() application

<code>\${DIALEDPEERNAME}</code>	* Dialed peer name
<code>\${DIALEDPEERNUMBER}</code>	* Dialed peer number
<code>\${DIALEDTIME}</code>	* Time for the call (seconds). Is only set if call was answered
<code>\${ANSWEREDTIME}</code>	* Time from answer to hangup (seconds)
<code>\${DIALSTATUS}</code>	* Status of the call, one of: (CHANUNAVAIL   CONGESTION   BUSY   NOANSWER   ANSWER   CANCEL   DONTCALL   TORTURE)
<code>\${DYNAMIC_FEATURES}</code>	* The list of features (from the [applicationmap] section of features.conf) to activate during the call, with feature names separated by '#' characters
<code>\${LIMIT_PLAYAUDIO_CALLER}</code>	Soundfile for call limits
<code>\${LIMIT_PLAYAUDIO_CALLEE}</code>	Soundfile for call limits
<code>\${LIMIT_WARNING_FILE}</code>	Soundfile for call limits
<code>\${LIMIT_TIMEOUT_FILE}</code>	Soundfile for call limits
<code>\${LIMIT_CONNECT_FILE}</code>	Soundfile for call limits
<code>\${OUTBOUND_GROUP}</code>	Default groups for peer channels (as in SetGroup) * See "show application dial" for more information

### 3.7.11 The chanisavail() application

<code>\${AVAILCHAN}</code>	* the name of the available channel if one was found
<code>\${AVAILORIGCHAN}</code>	* the canonical channel name that was used to create the channel
<code>\${AVAILSTATUS}</code>	* Status of requested channel

### 3.7.12 Dialplan Macros

<code>\${MACRO_EXTEN}</code>	* The calling extensions
<code>\${MACRO_CONTEXT}</code>	* 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(g

### 3.7.14 OSP

`#{OSPINHANDLE}` The inbound call OSP transaction handle.  
`#{OSPINTOKEN}` The inbound OSP token.  
`#{OSPINTIMELIMIT}` The inbound call duration limit in seconds.  
`#{OSPINPEERIP}` The last hop IP address.  
`#{OSPINNETWORKID}` The inbound source network ID.  
`#{OSPINNPRN}` The inbound routing number.  
`#{OSPINNPCIC}` The inbound carrier identification code.  
`#{OSPINNPDI}` The inbound number portability database dip indicator.  
`#{OSPINSPID}` The inbound service provider identity.  
`#{OSPINOCN}` The inbound operator company number.  
`#{OSPINSPN}` The inbound service provider name.  
`#{OSPINALTSPN}` The inbound alternate service provider name.  
`#{OSPINMCC}` The inbound mobile country code.  
`#{OSPINMNC}` The inbound mobile network code.  
`#{OSPINDIVUSER}` The inbound Diversion header user part.  
`#{OSPINDIVHOST}` The inbound Diversion header host part.  
`#{OSPINTOHOST}` The inbound To header host part.  
`#{OSPINCUSTOMINFOn}` The inbound custom information.  
 Where n is the index beginning with 1 upto 8.  
`#{OSPOUTHANDLE}` The outbound call OSP transaction handle.  
`#{OSPOUTTOKEN}` The outbound OSP token.  
`#{OSPOUTTIMELIMIT}` The outbound call duration limit in seconds.  
`#{OSPOUTTECH}` The outbound channel technology.  
`#{OSPOUTCALLIDTYPES}` The outbound Call-ID types.  
`#{OSPOUTCALLID}` The outbound Call-ID. Only for H.323.  
`#{OSPDESTINATION}` The destination IP address.  
`#{OSPDESTREMAILS}` The number of remained destinations.  
`#{OSPOUTCALLING}` The outbound calling number.

<code>\${OSPOUTCALLED}</code>	The outbound called number.
<code>\${OSPOUTNETWORKID}</code>	The outbound destination network ID.
<code>\${OSPOUTNPRN}</code>	The outbound routing number.
<code>\${OSPOUTNPCIC}</code>	The outbound carrier identification code.
<code>\${OSPOUTNPDI}</code>	The outbound number portability database dip indicator.
<code>\${OSPOUTSPID}</code>	The outbound service provider identity.
<code>\${OSPOUTOCN}</code>	The outbound operator company number.
<code>\${OSPOUTSPN}</code>	The outbound service provider name.
<code>\${OSPOUTALTSPN}</code>	The outbound alternate service provider name.
<code>\${OSPOUTMCC}</code>	The outbound mobile country code.
<code>\${OSPOUTMNC}</code>	The outbound mobile network code.
<code>\${OSPDIALSTR}</code>	The outbound Dial command string.
<code>\${OSPINAUDIOQOS}</code>	The inbound call leg audio QoS string.
<code>\${OSPOUTAUDIOQOS}</code>	The outbound call leg audio QoS string.

### 3.7.15 Digit manipulation

`${REDIRECTING_CALLEE_SEND_MACRO}`

Macro to call before sending a redirecting update to the callee

`${REDIRECTING_CALLEE_SEND_MACRO_ARGS}`

Arguments to pass to `${REDIRECTING_CALLEE_SEND_MACRO}`

`${REDIRECTING_CALLER_SEND_MACRO}`

Macro to call before sending a redirecting update to the caller

`${REDIRECTING_CALLER_SEND_MACRO_ARGS}`

Arguments to pass to `${REDIRECTING_CALLER_SEND_MACRO}`

`${CONNECTED_LINE_CALLEE_SEND_MACRO}`

Macro to call before sending a connected line update to the callee

`${CONNECTED_LINE_CALLEE_SEND_MACRO_ARGS}`

Arguments to pass to `${CONNECTED_LINE_CALLEE_SEND_MACRO}`

`${CONNECTED_LINE_CALLER_SEND_MACRO}`

Macro to call before sending a connected line update to the caller

`${CONNECTED_LINE_CALLER_SEND_MACRO_ARGS}`

Arguments to pass to `${CONNECTED_LINE_CALLER_SEND_MACRO}`

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

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

To reload `extensions.ael`, the following command can be issued at the CLI:  
\*CLI> ael reload

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

## 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-Z0-9"_.\<\>\*\!#\[\]] [-a-zA-Z0-9
<word3-list> a concatenation of up to 3 <word>s.
<collected-word> all characters encountered until the character that follows the <collected-word> in the grammar.
-----

<file> ::= <objects>

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

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

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

<word3-list> ::= <word>

```

```

| <word> <word>
| <word> <word> <word>

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

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

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

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

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

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

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

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

```



## 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.

```
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 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=DAHDI/g2;
}
```

Variables can be set within extensions as well.

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

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

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

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

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

You can declare variables in Macros, as so:

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

## 4.9.9 Local Variables

In 1.2, and 1.4, ALL VARIABLES are CHANNEL variables, including the function arguments and associated ARG1, ARG2, etc variables. Sorry.

In trunk (1.6 and higher), we have made all arguments local variables to a macro call. They will not affect channel variables of the same name. This includes the ARG1, ARG2, etc variables.

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

```

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

```

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

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

```

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

```

## 4.9.10 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 " $\{y\} \geq 0$ " 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.

## 4.9.11 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.

## 4.9.12 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.

## 4.9.13 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.

## 4.9.14 Macros

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

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

A macro is then called by preceding the macro name with an ampersand. Empty arguments can be passed simply with nothing between 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  $\$[ \dots ]$ , 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 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.

## 4.12 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.

## 4.13 Hints and Bugs

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

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

## 4.14 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 `dahdi.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 dahdi.conf or sip.conf. Once the trunk is configured, then map it to an SLA trunk in sla.conf.

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

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

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

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

For this example, let's say we have a SIP trunk called "mytrunk" that is going to be used as line4. Furthermore, when calls come in on this trunk,

they are going to say that they are calling the number "12564286000". Also, let's say that the numbers that are valid for calling out this trunk are NANP numbers, of the form `_1NXXNXXXXXX`.

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

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

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

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

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

## 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=DAHDI/1
autocontext=line1

[line2]
type=trunk
device=DAHDI/2
autocontext=line2

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

[station1](station)
device=SIP/station1

[station2](station)
device=SIP/station2

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

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

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

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

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

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

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

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

### 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 DAHDI/1 and DAHDI/2. The stations are SIP/station1, SIP/station2, and SIP/station3.

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

In sip.conf, all three stations are configured with context=sla\_stations.

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

sla.conf:

```
[line1]
```

```
type=trunk
device=Local/disa@line1_outbound
```

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

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

```
[station1](station)
device=SIP/station1
```

```
[station2](station)
device=SIP/station2
```

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

extensions.conf:

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

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

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

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

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

```
[sla_stations]

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

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

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

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

## 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 appearance 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 losspec <n>"`. This will simulate n percent packet loss coming `_in_` to `chan_iax2`. You should find that with PLC and the new JB, 10 percent packet loss should lead to just a tiny amount of distortion, while without PLC, it would lead to silent gaps in your audio.

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

The stats shown are:

- Jit: The jitter we have measured (milliseconds)
- Del: The maximum delay imposed by the jitterbuffer (milliseconds)
- Lost: The number of packets we've detected as lost.
- %: The percentage of packets we've detected as lost recently.

- Drop: The number of packets we've purposely dropped (to lower latency).
- OOO: The number of packets we've received out-of-order
- Kpkts: The number of packets we've received / 1000.

## Reporting problems

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

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 it)
- app\_SendText
- HOLD/RETRIEVE/TRANSFER on ISDN phones : )
- Allow/restrict user number presentation
- Volume control
- Crypting with mISDNdsp (Blowfish)
- Data (HDLC) callthrough
- Data calling (with app\_ptyfork +pppd)
- Echo cancellation
- Call deflection
- Some others

## 6.2.3 Fast Installation Guide

It is easy to install mISDN and mISDNuser. This can be done by:

- You can download latest stable releases from <http://www.misdn.org/downloads/>
- Just fetch the newest head of the GIT (mISDN project moved from CVS) In details this process described here: <http://www.misdn.org/index.php/GIT>

then compile and install both with:

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

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

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

Now you can compile chan\_misdn, just by making Asterisk:

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

That's all!

Follow the instructions in the mISDN Package for how to load the Kernel Modules. Also install process described in [http://www.misdn.org/index.php/Installing\\_mISDN](http://www.misdn.org/index.php/Installing_mISDN)

## 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.

You should use Kernels  $\geq 2.6.9$

## 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 trace file option, which takes a path+filename where debug output is written to.

### **misdn.conf: [default] subsection**

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

### **misdn.conf: user/port subsections**

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

Especially for TE-Mode Ports there is a msn option. This option tells the chan\_misdn driver to listen for incoming calls with the given msn, you can insert a '\*' as single msn, which leads to getting every incoming call. If you want to share on PMP TE S0 with Asterisk and a phone or ISDN card you should insert here the msn which you assign to Asterisk. Finally a context variable resides in the user subsections, which tells chan\_misdn where to send incoming calls to in the Asterisk dial plan (extension.conf).

### **Dial and Options String**

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

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

```
The Optionsstring looks Like:  
:<optchar><optarg>:<optchar><optarg>...
```

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

```
The available options are:
```

```

a - Have Asterisk detect DTMF tones on called channel
c - Make crypted outgoing call, optarg is keyindex
d - Send display text to called phone, text is the optarg
e - Perform echo cancelation on this channel,
    takes taps as optarg (32,64,128,256)
e! - Disable echo cancelation on this channel
f - Enable fax detection
h - Make digital outgoing call
h1 - Make HDLC mode digital outgoing call
i - Ignore detected DTMF tones, don't signal them to Asterisk,
    they will be transported inband.
jb - Set jitter buffer length, optarg is length
jt - Set jitter buffer upper threshold, optarg is threshold
jn - Disable jitter buffer
n - Disable mISDN DSP on channel.
    Disables: echo cancel, DTMF detection, and volume control.
p - Caller ID presentation,
    optarg is either 'allowed' or 'restricted'
s - Send Non-inband DTMF as inband
vr - Rx gain control, optarg is gain
vt - Tx gain control, optarg is gain

```

chan\_misdn registers a new dial plan application "misdn\_set\_opt" when loaded. This application takes the Optionsstring as argument. The Syntax is:

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

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

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

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

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

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

## 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 (must not be isdn though).

Then you use it like this:

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

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

## 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 into Asterisk and mISDN parts. mISDN Debug messages begin with an 'I', Asterisk messages begin with an '\*', the rest is clear I think.

Please take a trace of the problem and open a report in the Asterisk issue tracker at <https://issues.asterisk.org> in the "channel drivers" project, "chan\_misdn" category. Read the bug guidelines to make sure you provide all the information needed.

## 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 => _OX.,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

Q: I cannot hear any tone after a successful `CONNECT` to the other end.

A: You forgot to load `mISDNdsp`, which is now needed by `chan_misdn` for switching and DTMF tone detection.

## 6.3 Local

### 6.3.1 Introduction

In Asterisk, Local channels are a method used to treat an extension in the dialplan as if it were an external device. In essence, Asterisk will send the

call back into the dialplan as the destination of the call, versus sending the call to a device.

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

## 6.3.2 Examples

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

## 6.3.3 Trivial Local channel example

In our dialplan (extensions.conf), we can Dial() another part of the dialplan through the use Local channels. To do this, we can use the following dialplan:

```
[devices]
exten => 201,1,Verbose(2,Dial another part of the dialplan via the Local chan)
exten => 201,n,Verbose(2,Outside channel: ${CHANNEL})
exten => 201,n,Dial(Local/201@extensions)
exten => 201,n,Hangup()

[extensions]
exten => 201,1,Verbose(2,Made it to the Local channel)
exten => 201,n,Verbose(2,Inside channel: ${CHANNEL})
exten => 201,n,Dial(SIP/some-named-extension,30)
exten => 201,n,Hangup()
```

The output of the dialplan would look something like the following. The output has been broken up with some commentary to explain what we're looking at.

```
-- Executing [201@devices:1] Verbose("SIP/my_desk_phone-00000014", "2,Dial another part of the dialplan via the
Local chan") in new stack
== Dial another part of the dialplan via the Local chan
```

We dial extension 201 from SIP/my\_desk\_phone which has entered the [devices] context. The first line simply outputs some information via the Verbose() application.

```
-- Executing [201@devices:2] Verbose("SIP/my_desk_phone-00000014",
"2,Outside channel: SIP/my_desk_phone-00000014") in new stack
== Outside channel: SIP/my_desk_phone-00000014
```

The next line is another `Verbose()` application statement that tells us our current channel name. We can see that the channel executing the current dialplan is a desk phone (aptly named `'my_desk_phone'`).

```
-- Executing [201@devices:3] Dial("SIP/my_desk_phone-00000014", "Local/201@extensions") in new stack
-- Called 201@extensions
```

Now the third step in our dialplan executes the `Dial()` application which calls extension 201 in the `[extensions]` context of our dialplan. There is no requirement that we use the same extension number – we could have just as easily used a named extension, or some other number. Remember that we're dialing another channel, but instead of dialing a device, we're "dialing" another part of the dialplan.

```
-- Executing [201@extensions:1] Verbose("Local/201@extensions-7cf4;2", "2,Made it to the Local
channel") in new stack
== Made it to the Local channel
```

Now we've verified we've dialed another part of the dialplan. We can see the channel executing the dialplan has changed to `Local/201@extensions-7cf4;2`. The part `'-7cf4;2'` is just the unique identifier, and will be different for you.

```
-- Executing [201@extensions:2] Verbose("Local/201@extensions-7cf4;2", "2,Inside channel:
Local/201@extensions-7cf4;2") in new stack
== Inside channel: Local/201@extensions-7cf4;2
```

Here we use the `Verbose()` application to see what our current channel name is. As you can see the current channel is a Local channel which we created from our SIP channel.

```
-- Executing [201@extensions:3] Dial("Local/201@extensions-7cf4;2", "SIP/some-named-extension,30") in new stack
```

And from here, we're using another `Dial()` application to call a SIP device configured in `sip.conf` as `[some-named-extension]`.

Now that we understand a simple example of calling the Local channel, let's expand upon this example by using Local channels to call two devices at the same time, but delay calling one of the devices.

### 6.3.4 Delay dialing devices

Lets say when someone calls extension 201, we want to ring both the desk phone and their cellphone at the same time, but we want to wait about 6 seconds to start dialing the cellphone. This is useful in a situation when someone might be sitting at their desk, but don't want both devices ringing at the same time, but also doesn't want to wait for the full ring cycle to execute on their desk phone before rolling over to their cellphone.

The dialplan for this would look something like the following:

```
[devices]
exten => 201,1,Verbose(2,Call desk phone and cellphone but with delay)
exten => 201,n,Dial(Local/deskphone-201@extensions&Local/cellphone-201@extensions,30)
exten => 201,n,Voicemail(201@default,${IF(${DIALSTATUS} = BUSY)?b:u})
exten => 201,n,Hangup()

[extensions]
; Dial the desk phone
exten => deskphone-201,1,Verbose(2,Dialing desk phone of extension 201)
exten => deskphone-201,n,Dial(SIP/0004f2040001) ; SIP device with MAC address
; of 0004f2040001

; Dial the cellphone
exten => cellphone-201,1,Verbose(2,Dialing cellphone of extension 201)
exten => cellphone-201,n,Verbose(2,-- Waiting 6 seconds before dialing)
exten => cellphone-201,n,Wait(6)
exten => cellphone-201,n,Dial(DAHDI/g0/14165551212)
```

When someone dials extension 201 in the [devices] context, it will execute the Dial() application, and call two Local channels at the same time:

- Local/deskphone-201@extensions
- Local/cellphone-201@extensions

It will then ring both of those extensions for 30 seconds before rolling over to the Voicemail() application and playing the appropriate voicemail recording depending on whether the `${DIALSTATUS}` variable returned BUSY or not.

When reaching the deskphone-201 extension, we execute the Dial() application which calls the SIP device configured as '0004f204001' (the MAC address of the device). When reaching the cellphone-201 extension, we dial the cellphone via the DAHDI channel using group zero (g0) and dialing phone number 1-416-555-1212.

### 6.3.5 Dialing destinations with different information

With Asterisk, we can place a call to multiple destinations by separating the technology/destination pair with an ampersand (&). For example, the following Dial() line would ring two separate destinations for 30 seconds:

```
exten => 201,1,Dial(SIP/0004f2040001&DAHDI/g0/14165551212,30)
```

That line would dial both the SIP/0004f2040001 device (likely a SIP device on the network) and dial the phone number 1-416-555-1212 via a DAHDI interface. In our example though, we would be sending the same callerID information to both end points, but perhaps we want to send a different callerID to one of the destinations?

We can send different callerIDs to each of the destinations if we want by using the Local channel. The following example shows how this is possible because we would Dial() two different Local channels from our top level Dial(), and that would then execute some dialplan before sending the call off to the final destinations.

```
[devices]
exten => 201,1,NoOp()
exten => 201,n,Dial(Local/201@internal&Local/201@external,30)
exten => 201,n,Voicemail(201@default,${IF(${DIALSTATUS} = BUSY)?b:u})
exten => 201,n,Hangup()

[internal]
exten => 201,1,Verbose(2,Placing internal call for extension 201)
exten => 201,n,Set(CALLERID(name)=From Sales)
exten => 201,n,Dial(SIP/0004f2040001,30)

[external]
exten => 201,1,Verbose(2,Placing external call for extension 201)
exten => 201,n,Set(CALLERID(name)=Acme Cleaning)
exten => 201,n,Dial(DAHDI/g0/14165551212)
```

With the dialplan above, we've sent two different callerIDs to the destinations:

- "From Sales" was sent to the local device SIP/0004f2040001
- "Acme Cleaning" was sent to the remote number 1-416-555-1212 via DAHDI

Because each of the channels is independent from the other, you could perform any other call manipulation you need. Perhaps the 1-416-555-1212

number is a cell phone and you know you can only ring that device for 18 seconds before the voicemail would pick up. You could then limit the length of time the external number is dialed, but still allow the internal device to be dialed for a longer period of time.

### 6.3.6 Using callfiles and Local channels

Another example is to use callfiles and Local channels so that you can execute some dialplan prior to performing a Dial(). We'll construct a callfile which will then utilize a Local channel to lookup a bit of information in the AstDB and then place a call via the channel configured in the AstDB.

First, lets construct our callfile that will use the Local channel to do some lookups prior to placing our call. More information on constructing callfiles is located in the doc/callfiles.txt file of your Asterisk source.

Our callfile will simply look like the following:

```
Channel: Local/201@devices
Application: Playback
Data: silence/1&tt-weasels
```

Add the callfile information to a file such as 'callfile.new' or some other appropriately named file.

Our dialplan will perform a lookup in the AstDB to determine which device to call, and will then call the device, and upon answer, Playback() the silence/1 (1 second of silence) and the tt-weasels sound files.

Before looking at our dialplan, lets put some data into AstDB that we can then lookup from the dialplan. From the Asterisk CLI, run the following command:

```
*CLI> database put phones 201/device SIP/0004f2040001
```

We've now put the device destination (SIP/0004f2040001) into the 201/device key within the phones family. This will allow us to lookup the device location for extension 201 from the database.

We can then verify our entry in the database using the 'database show' CLI command:

```
*CLI> database show
/phones/201/device : SIP/0004f2040001
```

Now lets create the dialplan that will allow us to call SIP/0004f2040001 when we request extension 201 from the [extensions] context via our Local channel.

```
[devices]
exten => 201,1,NoOp()
exten => 201,n,Set(DEVICE=${DB(phones/${EXTEN}/device)})
exten => 201,n,GotoIf($[${ISNULL}(${DEVICE})]?hangup) ; if nothing returned,
                                           ; then hangup
exten => 201,n,Dial(${DEVICE},30)
exten => 201,n(hangup(),Hangup())
```

Then, we can perform a call to our device using the callfile by moving it into the /var/spool/asterisk/outgoing/ directory.

```
# mv callfile.new /var/spool/asterisks/outgoing
```

Then after a moment, you should see output on your console similar to the following, and your device ringing. Information about what is going on during the output has also been added throughout.

```
-- Attempting call on Local/201@devices for application Playback(silence/1&tt-weasels) (Retry 1)
```

You'll see the line above as soon as Asterisk gets the request from the callfile.

```
-- Executing [201@devices:1] NoOp("Local/201@devices-ecf0;2", "") in new stack
-- Executing [201@devices:2] Set("Local/201@devices-ecf0;2", "DEVICE=SIP/0004f2040001") in new stack
```

This is where we performed our lookup in the AstDB. The value of SIP/0004f2040001 was then returned and saved to the DEVICE channel variable.

```
-- Executing [201@devices:3] GotoIf("Local/201@devices-ecf0;2", "0?hangup") in new stack
```

We perform a check to make sure \${DEVICE} isn't NULL. If it is, we'll just hangup here.

```
-- Executing [201@devices:4] Dial("Local/201@devices-ecf0;2", "SIP/0004f2040001,30") in new stack
-- Called 000f2040001
-- SIP/0004f2040001-00000022 is ringing
```

Now we call our device SIP/0004f2040001 from the Local channel.

```
-- SIP/0004f2040001-00000022 answered Local/201@devices-ecf0;2
```

We answer the call.

```
> Channel Local/201@devices-ecf0;1 was answered.  
> Launching Playback(silence/1&tt-weasels) on Local/201@devices-ecf0;1
```

We then start playing back the files.

```
-- <Local/201@devices-ecf0;1> Playing 'silence/1.slin' (language 'en')  
== Spawn extension (devices, 201, 4) exited non-zero on 'Local/201@devices-ecf0;2'
```

At this point we now see the Local channel has been optimized out of the call path. This is important as we'll see in examples later. By default, the Local channel will try to optimize itself out of the call path as soon as it can. Now that the call has been established and audio is flowing, it gets out of the way.

```
-- <SIP/0004f2040001-00000022> Playing 'tt-weasels.ulaw' (language 'en')  
[Mar 1 13:35:23] NOTICE[16814]: pbx_spool.c:349 attempt_thread: Call completed to Local/201@devices
```

We can now see the tt-weasels file is played directly to the destination (instead of through the Local channel which was optimized out of the call path) and then a NOTICE stating the call was completed.

### 6.3.7 Understanding When To Use /n

Lets take a look at an example that demonstrates when the use of the /n directive is necessary. If we spawn a Local channel which does a Dial() to a SIP channel, but we use the L() option (which is used to limit the amount of time a call can be active, along with warning tones when the time is nearly up), it will be associated with the Local channel, which is then optimized out of the call path, and thus won't perform as expected.

This following dialplan will not perform as expected.

```
[services]  
exten => 2,1,Dial(SIP/PHONE\_B,,L(60000:45000:15000))
```

```
[internal]  
exten => 4,1,Dial(Local/2@services)
```

By default, the Local channel will try to optimize itself out of the call path. This means that once the Local channel has established the call between the destination and Asterisk, the Local channel will get out of the way and let Asterisk and the end point talk directly, instead of flowing through the Local channel.

This can have some adverse effects when you're expecting information to be available during the call that gets associated with the Local channel. When the Local channel is optimized out of the call path, any Dial() flags, or channel variables associated with the Local channel are also destroyed and are no longer available to Asterisk.

We can force the Local channel to remain in the call path by utilizing the /n directive. By adding /n to the end of the channel definition, we can keep the Local channel in the call path, along with any channel variables, or other channel specific information.

In order to make this behave as we expect (limiting the call), we would change:

```
[internal]
exten => 4,1,Dial(Local/2@services)
```

...into the following:

```
[internal]
exten => 4,1,Dial(Local/2@services/n)
```

By adding /n to the end, our Local channel will now stay in the call path and not go away.

Why does adding the /n option all of a sudden make the 'L' option work? First we need to show an overview of the call flow that doesn't work properly, and discuss the information associated with the channels:

1. SIP device PHONE\_A calls Asterisk via a SIP INVITE
2. Asterisk accepts the INVITE and then starts processing dialplan logic in the [internal] context
3. Our dialplan calls Dial(Local/2@services) – notice no /n
4. The Local channel then executes dialplan at extension 2 within the [services] context

5. Extension 2 within [services] then performs Dial() to PHONE\_B with the line: Dial(SIP/PHONE\_B,,L(60000:45000:15000))
6. SIP/PHONE\_B then answers the call
7. Even though the L option was given when dialing the SIP device, the L information is stored in the channel that is doing the Dial() which is the Local channel, and not the endpoint SIP channel.
8. The Local channel in the middle, containing the information for tracking the time allowance of the call, is then optimized out of the call path, losing all information about when to terminate the call.
9. SIP/PHONE\_A and SIP/PHONE\_B then continue talking indefinitely.

Now, if we were to add /n to our dialplan at step three (3) then we would force the Local channel to stay in the call path, and the L() option associated with the Dial() from the Local channel would remain, and our warning sounds and timing would work as expected.

There are two workarounds for the above described scenario:

1. Use what we just described, Dial(Local/2@services/n) to cause the Local channel to remain in the call path so that the L() option used inside the Local channel is not discarded when optimization is performed.
2. Place the L() option at the outermost part of the path so that when the middle is optimized out of the call path, the information required to make L() work is associated with the outside channel. The L information will then be stored on the calling channel, which is PHONE\_A. For example:

```
[services]
exten => 2,1,Dial(SIP/PHONE_B)

[internal]
exten => 4,1,Dial(Local/2@services,,L(60000:45000:15000));
```

### 6.3.8 Local channel modifiers

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

- 'n' – Adding "/n" at the end of the string will make the Local channel not do a native transfer (the "n" stands for "no release") upon the remote end answering the line. This is an esoteric, but important feature if you expect the Local channel to handle calls exactly like a normal channel. If you do not have the "no release" feature set, then as soon as the destination (inside of the Local channel) answers the line and one audio frame passes, the variables and dial plan will revert back to that of the original call, and the Local channel will become a zombie and be removed from the active channels list. This is desirable in some circumstances, but can result in unexpected dialplan behavior if you are doing fancy things with variables in your call handling.
- 'j' – Adding "/j" at the end of the string allows you to use the generic jitterbuffer on incoming calls going to Asterisk applications. For example, this would allow you to use a jitterbuffer for an incoming SIP call to Voicemail by putting a Local channel in the middle. The 'j' option must be used in conjunction with the 'n' option to make sure that the Local channel does not get optimized out of the call.

This option is available starting in the Asterisk 1.6.0 branch.

- 'm' – Using the "/m" option will cause the Local channel to forward music on hold (MoH) start and stop requests. Normally the Local channel acts on them and it is started or stopped on the Local channel itself. This options allows those requests to be forwarded through the Local channel.

This option is available starting in the Asterisk 1.4 branch.

- 'b' – The "/b" option causes the Local channel to return the actual channel that is behind it when queried. This is useful for transfer scenarios as the actual channel will be transferred, not the Local channel.

This option is available starting in the Asterisk 1.6.0 branch.

## 6.4 Mobile

### 6.4.1 Introduction

Asterisk Channel Driver to allow Bluetooth Cell/Mobile Phones to be used as FXO devices, and Headsets as FXS devices.

### 6.4.2 Features

- Multiple Bluetooth Adapters supported.
- Multiple phones can be connected.
- Multiple headsets can be connected.
- Asterisk automatically connects to each configured mobile phone / headset when it comes in range.
- CLI command to discover bluetooth devices.
- Inbound calls on the mobile network to the mobile phones are handled by Asterisk, just like inbound calls on a Zap channel.
- CLI passed through on inbound calls.
- Dial outbound on a mobile phone using Dial(Mobile/device/nnnnnnn)
- Dial a headset using Dial(Mobile/device) in the dialplan.
- Application MobileStatus can be used in the dialplan to see if a mobile phone / headset is connected.
- Supports devicestate for dialplan hinting.
- Supports Inbound and Outbound SMS.
- Supports 'channel' groups for implementing 'GSM Gateways'

### 6.4.3 Requirements

In order to use `chan_mobile`, you must have a working bluetooth subsystem on your Asterisk box. This means one or more working bluetooth adapters, and the BlueZ packages.

Any bluetooth adapter supported by the Linux kernel will do, including usb bluetooth dongles.

The BlueZ package you need is `bluez-utils`. If you are using a GUI then you might want to install `bluez-pin` also. You also need `libbluetooth`, and `libbluetooth-dev` if you are compiling Asterisk from source.

You need to get bluetooth working with your phone before attempting to use `chan_mobile`. This means 'pairing' your phone or headset with your Asterisk box. I dont describe how to do this here as the process differs from distro to distro. You only need to pair once per adapter.

See [www.bluez.org](http://www.bluez.org) for details about setting up Bluetooth under Linux.

### 6.4.4 Concepts

`chan_mobile` deals with both bluetooth adapters and bluetooth devices. This means you need to tell `chan_mobile` about the bluetooth adapters installed in your server as well as the devices (phones / headsets) you wish to use.

`chan_mobile` currently only allows one device (phone or headset) to be connected to an adapter at a time. This means you need one adapter for each device you wish to use simultaneously. Much effort has gone into trying to make multiple devices per adapter work, but in short it doesnt.

Periodically `chan_mobile` looks at each configured adapter, and if it is not in use (i.e. no device connected) will initiate a search for devices configured to use this adapater that may be in range. If it finds one it will connect the device and it will be available for Asterisk to use. When the device goes out of range, `chan_mobile` will disconnect the device and the adapter will become available for other devices.

### 6.4.5 Configuring `chan_mobile`

The configuration file for `chan_mobile` is `/etc/asterisk/mobile.conf`. It is a normal Asterisk config file consisting of sections and key=value pairs.

See `configs/mobile.conf.sample` for an example and an explanation of the configuration.

## 6.4.6 Using chan\_mobile

chan\_mobile.so must be loaded either by loading it using the Asterisk CLI, or by adding it to /etc/asterisk/modules.conf

Search for your bluetooth devices using the CLI command 'mobile search'. Be patient with this command as it will take 8 - 10 seconds to do the discovery. This requires a free adapter.

Headsets will generally have to be put into 'pairing' mode before they will show up here.

This will return something like the following :-

```
*CLI> mobile search
Address          Name                Usable Type    Port
00:12:56:90:6E:00 LG TU500           Yes  Phone    4
00:80:C8:35:52:78 Toaster            No   Headset   0
00:0B:9E:11:74:A5 Hello II Plus      Yes  Headset   1
00:0F:86:0E:AE:42 Daves Blackberry  Yes  Phone     7
```

This is a list of all bluetooth devices seen and whether or not they are usable with chan\_mobile. The Address field contains the 'bd address' of the device. This is like an ethernet mac address. The Name field is whatever is configured into the device as its name. The Usable field tells you whether or not the device supports the Bluetooth Handsfree Profile or Headset profile. The Type field tells you whether the device is usable as a Phone line (FXO) or a headset (FXS) The Port field is the number to put in the configuration file.

Choose which device(s) you want to use and edit /etc/asterisk/mobile.conf. There is a sample included with the Asterisk-addons source under configs/mobile.conf.sample.

Be sure to configure the right bd address and port number from the search. If you want inbound calls on a device to go to a specific context, add a context= line, otherwise the default will be used. The 'id' of the device [bitinbrackets] can be anything you like, just make it unique.

If you are configuring a Headset be sure to include the type=headset line, if left out it defaults to phone.

The CLI command 'mobile show devices' can be used at any time to show the status of configured devices, and whether or not the device is capable of sending / receiving SMS via bluetooth.

```
*CLI> mobile show devices
```

ID	Address	Group	Adapter	Connected	State	SMS
headset	00:0B:9E:11:AE:C6	0	blue	No	Init	No
LGTU550	00:E0:91:7F:46:44	1	dlink	No	Init	No

As each phone is connected you will see a message on the Asterisk console :-

```
Loaded chan_mobile.so => (Bluetooth Mobile Device Channel Driver)
-- Bluetooth Device blackberry has connected.
-- Bluetooth Device dave has connected.
```

To make outbound calls, add something to your Dialplan like the following :- (modify to suit)

```
; Calls via LGTU5500
```

```
exten => _9X.,1,Dial(Mobile/LGTU550/${EXTEN:1},45)
exten => _9X.,n,Hangup
```

To use channel groups, add an entry to each phone's definition in `mobile.conf` like `group=n` where `n` is a number.

Then if you do something like `Dial(Mobile/g1/123456)` Asterisk will dial 123456 on the first connected free phone in group 1.

Phones which do not have a specific `'group=n'` will be in group 0.

To dial out on a headset, you need to use some other mechanism, because the headset is not likely to have all the needed buttons on it. `res_clioriginat` is good for this :-

```
*CLI> originate Mobile/headset extension NNNNN@context
```

This will call your headset, once you answer, Asterisk will call NNNNN at context context

## 6.4.7 Dialplan hints

`chan_mobile` supports 'device status' so you can do something like

```
exten => 1234,hint,SIP/30&Mobile/dave&Mobile/blackberry
```

## 6.4.8 MobileStatus Application

chan\_mobile also registers an application named MobileStatus. You can use this in your Dialplan to determine the 'state' of a device.

For example, suppose you wanted to call dave's extension, but only if he was in the office. You could test to see if his mobile phone was attached to Asterisk, if it is dial his extension, otherwise dial his mobile phone.

```
exten => 40,1,MobileStatus(dave,DAVECELL)
exten => 40,2,GotoIf($["${DAVECELL}" = "1"]?3:5)
exten => 40,3,Dial(ZAP/g1/0427466412,45,tT)
exten => 40,4,Hangup
exten => 40,5,Dial(SIP/40,45,tT)
exten => 40,6,Hangup
```

MobileStatus sets the value of the given variable to :-

```
1 = Disconnected. i.e. Device not in range of Asterisk, or turned off etc etc
2 = Connected and Not on a call. i.e. Free
3 = Connected and on a call. i.e. Busy
```

## 6.4.9 SMS Sending / Receiving

If Asterisk has detected your mobile phone is capable of SMS via bluetooth, you will be able to send and receive SMS.

Incoming SMS's cause Asterisk to create an inbound call to the context you defined in mobile.conf or the default context if you did not define one. The call will start at extension 'sms'. Two channel variables will be available, SMSSRC = the number of the originator of the SMS and SMSTXT which is the text of the SMS. This is not a voice call, so grab the values of the variables and hang the call up.

So, to handle incoming SMS's, do something like the following in your dialplan

```
[incoming-mobile]
exten => sms,1,Verbose(Incoming SMS from ${SMSSRC} ${SMSTXT})
exten => sms,n,Hangup()
```

The above will just print the message on the console.

If you use res\_jabber, you could do something like this :-

```
[incoming-mobile]
exten => sms,1,JabberSend(transport,user@jabber.somewhere.com,SMS from ${SMSRC} ${SMSTXT})
exten => sms,2,Hangup()
```

To send an SMS, use the application MobileSendSMS like the following :-

```
exten => 99,1,MobileSendSMS(dave,0427123456,Hello World)
```

This will send 'Hello World' via device 'dave' to '0427123456'

### 6.4.10 DTMF Debouncing

DTMF detection varies from phone to phone. There is a configuration variable that allows you to tune this to your needs. e.g. in mobile.conf

```
[LGTU550]
address=00:12:56:90:6E:00
port=4
context=incoming-mobile
dtmfskip=50
```

change dtmfskip to suit your phone. The default is 200. The larger the number, the more chance of missed DTMF. The smaller the number the more chance of multiple digits being detected.

### 6.4.11 Debugging

Different phone manufacturers have different interpretations of the Bluetooth Handsfree Profile Spec. This means that not all phones work the same way, particularly in the connection setup / initialisation sequence. I've tried to make chan\_mobile as general as possible, but it may need modification to support some phone i've never tested.

Some phones, most notably Sony Ericsson 'T' series, dont quite conform to the Bluetooth HFP spec. chan\_mobile will detect these and adapt accordingly. The T-610 and T-630 have been tested and work fine.

If your phone doesnt behave as expected, turn on Asterisk debugging with 'core set debug 1'.

This will log a bunch of debug messages indicating what the phone is doing, importantly the rfcmm conversation between Asterisk and the phone.

This can be used to sort out what your phone is doing and make chan\_mobile support it.

Be aware also, that just about all mobile phones behave differently. For example my LG TU500 wont dial unless the phone is a the 'idle' screen. i.e. if the phone is showing a 'menu' on the display, when you dial via Asterisk, the call will not work. chan\_mobile handles this, but there may be other phones that do other things too...

Important: Watch what your mobile phone is doing the first few times. Asterisk wont make random calls but if chan\_mobile fails to hangup for some reason and you get a huge bill from your telco, dont blame me ;)

# 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 DUNDIQUERY and DUNDIRESLT

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

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

## 7.3 Peering Agreement

DIGIUM GENERAL PEERING AGREEMENT (TM)

Version 1.0.0, September 2004

Copyright (C) 2004 Digium, Inc.

445 Jan Davis Drive, Huntsville, AL 35806 USA

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

-----  
DIGIUM GENERAL PEERING AGREEMENT

PREAMBLE

For most of the history of telecommunications, the power of being able to locate and communicate with another person in a system, be it across a hall or around the world, has always centered around a centralized authority -- from a local PBX administrator to regional and national RBOCs, generally requiring fees, taxes or regulation. By contrast, DUNDi is a technology developed to provide users the freedom to communicate with each other without the necessity of any centralized

authority. This General Peering Agreement ("GPA") is used by individual parties (each, a "Participant") to allow them to build the E164 trust group for the DUNDi protocol.

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

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

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

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

The parties to this GPA agree as follows:

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

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

DUNDi protocol specification.

- (b) The terms "E.164" and "E164" mean ITU-T specification E.164 as published by the International Telecommunications Union (ITU) in May, 1997.
- (c) The term "Service" refers to any communication facility (e.g., telephone, fax, modem, etc.), identified by an E.164-compatible number, and assigned by the appropriate authority in that jurisdiction.
- (d) The term "Egress Gateway" refers an Internet facility that provides a communications path to a Service or Services that may not be directly addressable via the Internet.
- (e) The term "Route" refers to an Internet address, policies, and other characteristics defined by the DUNDi protocol and associated with the Service, or the Egress Gateway which provides access to the specified Service.
- (f) The term "Propagate" means to accept or transmit Service and/or Egress Gateway Routes only using the DUNDi protocol and the DUNDi context "e164" without regard to case, and does not apply to the exchange of information using any other protocol or context.
- (g) The term "Peering System" means the network of systems that Propagate Routes.
- (h) The term "Subscriber" means the owner of, or someone who contracts to receive, the services identified by an E.164 number.
- (i) The term "Authorizing Individual" means the Subscriber to a number who has authorized a Participant to provide Routes regarding their services via this Peering System.
- (j) The term "Route Authority" refers to a Participant that provides

an original source of said Route within the Peering System. Routes are propagated from the Route Authorities through the Peering System and may be cached at intermediate points. There may be multiple Route Authorities for any Service.

- (k) The term "Participant" (introduced above) refers to any member of the Peering System.
- (l) The term "Service Provider" refers to the carrier (e.g., exchange carrier, Internet Telephony Service Provider, or other reseller) that provides communication facilities for a particular Service to a Subscriber, Customer or other End User.
- (m) The term "Weight" refers to a numeric quality assigned to a Route as per the DUNDi protocol specification. The current Weight definitions are shown in Exhibit A.

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

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

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

- (b) A Participant may not utilize or permit the utilization of Routes which have indicated that they do not wish to receive "Unsolicited Commercial Calls" for the purpose of making an unsolicited phone call on behalf of a commercial organization.
- (c) A Participant may never utilize or permit the utilization of any DUNDi route for the purpose of making harassing phone calls.
- (d) A Party may not utilize or permit the utilization of DUNDi provided Routes for any systematic or random calling of numbers (e.g., for the purpose of locating facsimile, modem services, or systematic telemarketing).
- (e) Initial control signaling for all communication sessions that utilize Routes obtained from the Peering System must be sent from a member of the Peering System to the Service or Egress Gateway identified in the selected Route. For example, 'SIP INVITES' and IAX2 "NEW" commands must be sent from the requesting DUNDi node to the terminating Service.
- (f) A Participant may not disclose any specific Route, Service or Participant contact information obtained from the Peering System to any party outside of the Peering System except as a by-product of facilitating communication in accordance with section 2e (e.g., phone books or other databases may not be published, but the Internet addresses of the Egress Gateway or Service does not need to be obfuscated.)
- (g) The DUNDi Protocol requires that each Participant include valid contact information about itself (including information about nodes connected to each Participant). Participants may use or disclose the contact information only to ensure enforcement of legal furtherance of this Agreement.

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

such Route to be invalid or unauthorized.

- (a) A Participant may provide Routes if each Route has as its original source another member of the Peering System who has duly executed the GPA and such Routes are provided in accordance with this Agreement; provided that the Routes are not modified (e.g., with regards to existence, destination, technology or Weight); or
- (b) A Participant may provide Routes for Services with any Weight for which it is the Subscriber; or
- (c) A Participant may provide Routes for those Services whose Subscriber has authorized the Participant to do so, provided that the Participant is able to confirm that the Authorizing Individual is the Subscriber through:
  - i. a written statement of ownership from the Authorizing Individual, which the Participant believes in good faith to be accurate (e.g., a phone bill with the name of the Authorizing Individual and the number in question); or
  - ii. the Participant's own direct personal knowledge that the Authorizing Individual is the Subscriber.
- (d) A Participant may provide Routes for Services, with Weight in accordance with the Current DUNDi Specification, if it can in good faith provide an Egress Gateway to that Service on the traditional telephone network without cost to the calling party.

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

- (a) receipt of a revocation request;
- (b) receiving other notice that the Service is no longer valid; or

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

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

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

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

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

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

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

E.164 number.

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

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

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

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

PARTICIPANT WAS ADVISED OF THE POSSIBILITY THEREOF.

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

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

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

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

19. CHOICE OF LAW. This GPA and the rights and duties of the Parties

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

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

- (a) Prior to mediation or arbitration the disputing Participants shall seek informal resolution of disputes. The process shall be initiated with written notice of one Participant to the other describing the dispute with reasonable particularity followed with a written response within ten (10) days of receipt of notice. Each Participant shall promptly designate an executive with requisite authority to resolve the dispute. The informal procedure shall commence within ten (10) days of the date of response. All reasonable requests for non-privileged information reasonably related to the dispute shall be honored. If the dispute is not resolved within thirty (30) days of commencement of the procedure either Participant may proceed to mediation or arbitration pursuant to the rules set forth in (b) or (c) below.
- (b) If the dispute has not been resolved pursuant to (a) above or, if the disputing Participants fail to commence informal dispute resolution pursuant to (a) above, either Participant may, in writing and within twenty (20) days of the response date noted in (a) above, ask the other Participant to participate in a one (1) day mediation with an impartial mediator, and the other Participant shall do so. Each Participant will bear its own expenses and an equal share of the fees of the mediator. If the mediation is not successful the Participants may proceed with arbitration pursuant to (c) below.
- (c) If the dispute has not been resolved pursuant to (a) or (b) above, the dispute shall be promptly referred, no later than one (1) year from the date of original notice and subject to

applicable statute of limitations, to binding arbitration in accordance with the UNCITRAL Arbitration Rules in effect on the date of this contract. The appointing authority shall be the International Centre for Dispute Resolution. The case shall be administered by the International Centre for Dispute Resolution under its Procedures for Cases under the UNCITRAL Arbitration Rules. Each Participant shall bear its own expenses and shall share equally in fees of the arbitrator. All arbitrators shall have substantial experience in information technology and/or in the telecommunications business and shall be selected by the disputing participants in accordance with UNCITRAL Arbitration Rules. If any arbitrator, once selected is unable or unwilling to continue for any reason, replacement shall be filled via the process described above and a re-hearing shall be conducted. The disputing Participants will provide each other with all requested documents and records reasonably related to the dispute in a manner that will minimize the expense and inconvenience of both parties. Discovery will not include depositions or interrogatories except as the arbitrators expressly allow upon a showing of need. If disputes arise concerning discovery requests, the arbitrators shall have sole and complete discretion to resolve the disputes. The parties and arbitrator shall be guided in resolving discovery disputes by the Federal Rules of Civil Procedure. The Participants agree that time of the essence principles shall guide the hearing and that the arbitrator shall have the right and authority to issue monetary sanctions in the event of unreasonable delay. The arbitrator shall deliver a written opinion setting forth findings of fact and the rationale for the award within thirty (30) days following conclusion of the hearing. The award of the arbitrator, which may include legal and equitable relief, but which may not include punitive damages, will be final and binding upon the disputing Participants, and judgment may be entered upon it in accordance with applicable law in any court having jurisdiction thereof. In addition to award the arbitrator shall have the discretion to award the prevailing Participant all or part of its attorneys' fees and costs, including fees associated with arbitrator, if the arbitrator

determines that the positions taken by the other Participant on material issues of the dispute were without substantial foundation. Any conflict between the UNCITRAL Arbitration Rules and the provisions of this GPA shall be controlled by this GPA.

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

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

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

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

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

Exhibit A

Weight Range	Requirements
0-99	May only be used under authorization of Owner
100-199	May only be used by the Owner's service provider, regardless of authorization.
200-299	Reserved -- do not use for e164 context.
300-399	May only be used by the owner of the code under which the Owner's number is a part of.
400-499	May be used by any entity providing access via direct connectivity to the Public Switched Telephone Network.
500-599	May be used by any entity providing access via indirect connectivity to the Public Switched Telephone Network (e.g. Via another VoIP provider)
600-	Reserved-- do not use for e164 context.

Participant

Participant

Company:

Address:

Email:

-----  
Authorized Signature

-----  
Authorized Signature

Name:

END OF GENERAL PEERING AGREEMENT

-----

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

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

# 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.yoyodynelabs.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/${enumresult})
exten => _011.,n,Dial(DAHDI/g1/${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,,${counter})})
exten => _011.,n,EndWhile
exten => _011.,n,Dial(DAHDIG1/${EXTEN})
;
; end example 2

; example 3
;
; This example expects an ${EXTEN} that is an e.164 number (like
; 14102241145 or 437203001721)
; Search through e164.arpa and then also search through e164.org
; to see if there are any valid SIP or IAX termination capabilities.
; If none, send call out via DAHDIG1 channel 1.
;
; Start first with e164.arpa zone...
;
exten => _X.,1,Set(sipcount=${ENUMLOOKUP(+${EXTEN},sip,c)}|counter=0)
exten => _X.,2,GotoIf($["${counter}"<"${sipcount}"]?3:6)
exten => _X.,3,Set(counter=${counter}+1)
exten => _X.,4,Dial(SIP/${ENUMLOOKUP(+${EXTEN},sip,,${counter})})
exten => _X.,5,GotoIf($["${counter}"<"${sipcount}"]?3:6)
;
exten => _X.,6,Set(iaxcount=${ENUMLOOKUP(+${EXTEN},iax2,c)}|counter=0)
exten => _X.,7,GotoIf($["${counter}"<"${iaxcount}"]?8:11)
exten => _X.,8,Set(counter=${counter}+1)
exten => _X.,9,Dial(IAX2/${ENUMLOOKUP(+${EXTEN},iax2,,${counter})})
exten => _X.,10,GotoIf($["${counter}"<"${iaxcount}"]?8:11)
;
exten => _X.,11,NoOp("No valid entries in e164.arpa for ${EXTEN} - checking in e164.org")
;
; ...then also try e164.org, and look for SIP and IAX NAPTRs...
;
exten => _X.,12,Set(sipcount=${ENUMLOOKUP(+${EXTEN},sip,c,,e164.org)}|counter=0)
exten => _X.,13,GotoIf($["${counter}"<"${sipcount}"]?14:17)
exten => _X.,14,Set(counter=${counter}+1)
exten => _X.,15,Dial(SIP/${ENUMLOOKUP(+${EXTEN},sip,,${counter},e164.org)})
exten => _X.,16,GotoIf($["${counter}"<"${sipcount}"]?14:17)
;
exten => _X.,17,Set(iaxcount=${ENUMLOOKUP(+${EXTEN},iax2,c,,e164.org)}|counter=0)
exten => _X.,18,GotoIf($["${counter}"<"${iaxcount}"]?19:22)
exten => _X.,19,Set(counter=${counter}+1)
exten => _X.,20,Dial(IAX2/${ENUMLOOKUP(+${EXTEN},iax2,,${counter},e164.org)})
exten => _X.,21,GotoIf($["${counter}"<"${iaxcount}"]?19:22)
;
; ...then send out PRI.
;
exten => _X.,22,NoOp("No valid entries in e164.org for ${EXTEN} - sending out via DAHDIG1")
exten => _X.,23,Dial(DAHDIG1/${EXTEN})
;
; end example 3

```

## 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.

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)

## 9.2 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

## 9.3 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.

## 9.4 Examples

Login - Log a user into the manager interface.

```
Action: Login
Username: testuser
Secret: testsecret
```

Originate - Originate a call from a channel to an extension.

```
Action: Originate
Channel: sip/12345
Exten: 1234
Context: default
```

Originate - Originate a call from a channel to an extension without waiting for call to complete.

```
Action: Originate
Channel: sip/12345
Exten: 1234
Context: default
Async: yes
```

Redirect with ExtraChannel:

Attempted goal: Have a 'robot' program Redirect both ends of an already-connected call to a meetme room using the ExtraChannel feature through the management interface.

```
Action: Redirect
Channel: DAHDI/1-1
ExtraChannel: SIP/3064-7e00 (varies)
Exten: 680
Priority: 1
```

Where 680 is an extension that sends you to a MeetMe room.

There are a number of GUI tools that use the manager interface, please search the mailing list archives and the documentation page on the <http://www.asterisk.org> web site for more information.

## 9.5 Ensuring all modules are loaded

It is possible to connect to the manager interface before all Asterisk modules are loaded. To ensure that an application does not send AMI actions that might require a module that has not yet loaded, the application can listen for the FullyBooted manager event. It will be sent upon connection if all modules have been loaded, or as soon as loading is complete. The event:

```
Event: FullyBooted
Privilege: system,all
Status: Fully Booted
```

## 9.6 Device status reports

## 9.7 Some standard AMI headers

```
Account:                -- Account Code (Status)
AccountCode:           -- Account Code (cdr_manager)
ACL: <Y | N>           -- Does ACL exist for object ?
Action: <action>       -- Request or notification of a particular action
Address-IP:            -- IPaddress
Address-Port:          -- IP port number
Agent: <string>        -- Agent name
AMAflags:              -- AMA flag (cdr_manager, sippeers)
AnswerTime:            -- Time of answer (cdr_manager)
Append: <bool>         -- CDR userfield Append flag
Application:           -- Application to use
Async:                 -- Whether or not to use fast setup
AuthType:              -- Authentication type (for login or challenge)
                        "md5"
BillableSeconds:       -- Billable seconds for call (cdr_manager)
CallerID:              -- Caller id (name and number in Originate & cdr_manager)
CallerID:              -- CallerID number
                        Number or "<unknown>" or "unknown"
                        (should change to "<unknown>" in app_queue)
CallerID1:             -- Channel 1 CallerID (Link event)
CallerID2:             -- Channel 2 CallerID (Link event)
CallerIDName:          -- CallerID name
                        Name or "<unknown>" or "unknown"
                        (should change to "<unknown>" in app_queue)
Callgroup:             -- Call group for peer/user
CallsTaken: <num>      -- Queue status variable
Cause: <value>         -- Event change cause - "Expired"
Cause: <value>         -- Hangupcause (channel.c)
CID-CallingPres:       -- Caller ID calling presentation
Channel: <channel>     -- Channel specifier
Channel: <dialstring>  -- Dialstring in Originate
Channel: <tech/[peer/username]> -- Channel in Registry events (SIP, IAX2)
Channel: <tech>        -- Technology (SIP/IAX2 etc) in Registry events
```

```

ChannelType:          -- Tech: SIP, IAX2, DAHDI, MGCP etc
Channel1:             -- Link channel 1
Channel2:             -- Link channel 2
ChanObjectType:      -- "peer", "user"
Codecs:               -- Codec list
CodecOrder:          -- Codec order, separated with comma ","
Command:              -- Cli command to run
Context:              -- Context
Count: <num>         -- Number of callers in queue
Data:                 -- Application data
Default-addr-IP:     -- IP address to use before registration
Default-Username:    -- Username part of URI to use before registration
Destination:         -- Destination for call (Dialstring ) (dial, cdr_manager)
DestinationContext:  -- Destination context (cdr_manager)
DestinationChannel:  -- Destination channel (cdr_manager)
DestUniqueID:        -- UniqueID of destination (dial event)
Direction: <type>    -- Audio to mute (read | write | both)
Disposition:         -- Call disposition (CDR manager)
Domain: <domain>     -- DNS domain
Duration: <secs>     -- Duration of call (cdr_manager)
Dynamic: <Y | N>     -- Device registration supported?
Endtime:             -- End time stamp of call (cdr_manager)
EventList: <flag>    -- Flag being "Start", "End", "Cancelled" or "ListObject"
Events: <eventmask>  -- Eventmask filter ("on", "off", "system", "call", "log")
Exten:               -- Extension (Redirect command)
Extension:           -- Extension (Status)
Family: <string>     -- ASTdb key family
File: <filename>     -- Filename (monitor)
Format: <format>     -- Format of sound file (monitor)
From: <time>         -- Parking time (ParkedCall event)
Hint:                -- Extension hint
Incominglimit:       -- SIP Peer incoming limit
Key:                 -- ASTdb Database 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 "enc
Location:                -- Interface (whatever that is -maybe tech/name in app_c
Loginchan:               -- Login channel for agent
Logintime: <number>      -- Login time for agent
Mailbox:                 -- VM Mailbox (id@vmcontext) (mailboxstatus, mailboxcount
MD5SecretExist: <Y | N>  -- Whether secret exists in MD5 format
Membership: <string>     -- "Dynamic" or "static" member in queue
Message: <text>          -- Text message in ACKs, errors (explanation)
Mix: <bool>              -- Boolean parameter (monitor)
MOHSuggest:              -- Suggested music on hold class for peer (mohsuggest)
NewMessages: <count>    -- Count of new Mailbox messages (mailboxcount)
Newname:
ObjectName:              -- Name of object in list
OldName:                 -- Something in Rename (channel.c)
OldMessages: <count>    -- Count of old mailbox messages (mailboxcount)
Outgoinglimit:          -- SIP Peer outgoing limit
Paused: <num>           -- Queue member paused status
Peer: <tech/name>       -- "channel" specifier :-)
PeerStatus: <tech/name> -- Peer status code
                        "Unregistered", "Registered", "Lagged", "Reachable"
Penalty: <num>          -- Queue penalty
Priority:                 -- Extension priority
Privilege: <privilege>  -- AMI authorization class (system, call, log, verbose,
Pickupgroup:             -- Pickup group for peer
Position: <num>         -- Position in Queue
Queue:                   -- Queue name
Reason:                  -- "Autologoff"
Reason:                  -- "Chanunavail"
Response: <response>    -- response code, like "200 OK"
                        "Success", "Error", "Follows"
Restart:                 -- "True", "False"
RegExpire:               -- SIP registry expire
RegExpiry:               -- SIP registry expiry
Reason:                  -- Originate reason code
Seconds:                 -- Seconds (Status)
Secret: <password>      -- Authentication secret (for login)
SecretExist: <Y | N>    -- Whether secret exists
Shutdown:                -- "Uncleanly", "Cleanly"

```

```

SIP-AuthInsecure:
SIP-FromDomain:      -- Peer FromDomain
SIP-FromUser:       -- Peer FromUser
SIP-NatSupport:
SIPLastMsg:
Source:              -- Source of call (dial event, cdr_manager)
SrcUniqueID:         -- UniqueID of source (dial event)
StartTime:           -- Start time of call (cdr_manager)
State:                -- Channel state
State: <1 | 0>       -- Mute flag
Status:              -- Registration status (Registry events SIP)
Status:              -- Extension status (Extensionstate)
Status:              -- Peer status (if monitored)  ** Will change name **
                    "unknown", "lagged", "ok"
Status: <num>        -- Queue Status
Status:              -- DND status (DNDState)
Time: <sec>          -- Roundtrip time (latency)
Timeout:             -- Parking timeout time
Timeout:             -- Timeout for call setup (Originate)
Timeout: <seconds>  -- Timeout for call
Uniqueid:            -- Channel Unique ID
Uniqueid1:           -- Channel 1 Unique ID (Link event)
Uniqueid2:           -- Channel 2 Unique ID (Link event)
User:                -- Username (SIP registry)
UserField:           -- CDR userfield (cdr_manager)
Val:                 -- Value to set/read in ASTdb
Variable:            -- Variable AND value to set (multiple separated with |
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.

## 9.8 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:

### 9.8.1 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.

### 9.8.2 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 "http show status" at the Asterisk CLI.

examples:

```
http://localhost:8088/asterisk/manager?action=login&username=foo&secret=bar
```

This logs you into the manager interface's "HTML" view. Once you're logged in, Asterisk stores a cookie on your browser (valid for the length of httptimeout) which is used to connect to the same session.

```
http://localhost:8088/asterisk/rawman?action=status
```

Assuming you've already logged into manager, this URI will give you a "raw" manager output for the "status" command.

```
http://localhost:8088/asterisk/mxml?action=status
```

This will give you the same status view but represented as AJAX data, theoretically compatible with RICO (<http://www.openrico.org>).

```
http://localhost:8088/asterisk/static/ajamdemo.html
```

If you have enabled static content support and have done a make install, Asterisk will serve up a demo page which presents a live, but very basic, "astman" like interface. You can login with your username/secret for manager and have a basic view of channels as well as transfer and hangup calls. It's only tested in Firefox, but could probably be made to run in other browsers as well.

A sample library (astman.js) is included to help ease the creation of manager HTML interfaces.

Note that for the demo, there is no need for *any* external web server.

### 9.8.3 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 CDR Fields

- accountcode: What account number to use, (string, 20 characters)

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

In some cases, uniqueid is appended:

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

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 Variables

If the channel has a CDR, that CDR has its own set of variables which can be accessed just like channel variables. The following builtin variables are available.

<code>\${CDR(clid)}</code>	Caller ID
<code>\${CDR(src)}</code>	Source
<code>\${CDR(dst)}</code>	Destination
<code>\${CDR(dcontext)}</code>	Destination context
<code>\${CDR(channel)}</code>	Channel name
<code>\${CDR(dstchannel)}</code>	Destination channel
<code>\${CDR(lastapp)}</code>	Last app executed
<code>\${CDR(lastdata)}</code>	Last app's arguments
<code>\${CDR(start)}</code>	Time the call started.
<code>\${CDR(answer)}</code>	Time the call was answered.
<code>\${CDR(end)}</code>	Time the call ended.
<code>\${CDR(duration)}</code>	Duration of the call.
<code>\${CDR(billsec)}</code>	Duration of the call once it was answered.
<code>\${CDR(disposition)}</code>	ANSWERED, NO ANSWER, BUSY
<code>\${CDR(amaflags)}</code>	DOCUMENTATION, BILL, IGNORE etc
<code>\${CDR(accountcode)}</code>	The channel's account code.
<code>\${CDR(uniqueid)}</code>	The channel's unique id.
<code>\${CDR(userfield)}</code>	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.

## 10.4 Storage Backends

### 10.4.1 Microsoft SQL Server

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.

#### 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] (150)      NOT NULL ,
    [userfield]    [varchar] (255)      NOT NULL
)

```

Start asterisk in verbose mode, you should see that asterisk logs a connection to the database and will now record every call to the database when it's complete.

## TDS, using cdr\_tds

Compile, configure, and install the latest FreeTDS package:

```
tar -zxvf freetds-0.62.4.tar.gz &&
cd freetds-0.62.4 &&
./configure --prefix=/usr --with-tdsver=7.0
make &&
make install
```

Compile, or recompile, asterisk so that it will now add support for cdr\_tds.

```
make clean && ./configure --with-tds &&
make update &&
make &&
make install
```

Only install one database connector. Do not confuse asterisk by using both ODBC (cdr\_odbc) and FreeTDS (cdr\_tds). This command will erase the contents of cdr\_odbc.conf

```
[ -f /etc/asterisk/cdr_odbc.conf ] > /etc/asterisk/cdr_odbc.conf
```

Setup cdr\_tds configuration files. These are working samples from my system. You will need to modify for your setup. Define your usernames and passwords here, secure file as well.

```
                                /etc/asterisk/cdr_tds.conf
[global]
hostname=192.168.1.25
port=1433
dbname=voipdb
user=voipdbuser
password=voipdbpass
charset=BIG5
```

And finally, create the 'cdr' table in your mssql database.

```
CREATE TABLE cdr (
  [accountcode] [varchar] (20) NULL ,
  [src] [varchar] (80) NULL ,
  [dst] [varchar] (80) NULL ,
  [dcontext] [varchar] (80) NULL ,
  [clid] [varchar] (80) NULL ,
  [channel] [varchar] (80) NULL ,
  [dstchannel] [varchar] (80) NULL ,
  [lastapp] [varchar] (80) NULL ,
  [lastdata] [varchar] (80) NULL ,
  [start] [datetime] NULL ,
  [answer] [datetime] NULL ,
  [end] [datetime] NULL ,
  [duration] [int] NULL ,
  [billsec] [int] NULL ,
```

```

        [disposition]    [varchar] (20)          NULL ,
        [amaflags]       [varchar] (16)         NULL ,
        [uniqueid]       [varchar] (150)        NULL ,
        [userfield]     [varchar] (256)        NULL
    )

```

Start asterisk in verbose mode, you should see that asterisk logs a connection to the database and will now record every call to the database when it's complete.

## 10.4.2 MySQL

### ODBC

Using MySQL for CDR records is supported by using ODBC and the cdr-odbc module.

### Native

Alternatively, there is a native MySQL CDR module.

To use it, configure the module in cdr\_mysql.conf. Create a table called cdr under the database name you will be using the following schema.

```

CREATE TABLE cdr (
    calldate datetime NOT NULL default '0000-00-00 00:00:00',
    clid varchar(80) NOT NULL default '',
    src varchar(80) NOT NULL default '',
    dst varchar(80) NOT NULL default '',
    dcontext varchar(80) NOT NULL default '',
    channel varchar(80) NOT NULL default '',
    dstchannel varchar(80) NOT NULL default '',
    lastapp varchar(80) NOT NULL default '',
    lastdata varchar(80) NOT NULL default '',
    duration int(11) NOT NULL default '0',
    billsec int(11) NOT NULL default '0',
    disposition varchar(45) NOT NULL default '',
    amaflags int(11) NOT NULL default '0',
    accountcode varchar(20) NOT NULL default '',
    uniqueid varchar(32) NOT NULL default '',
    userfield varchar(255) NOT NULL default ''
);

```

## 10.4.3 PostgreSQL

If you want to go directly to postgresql database, and have the cdr\_pgsql.so compiled you can use the following sample setup. On Debian, before compiling asterisk, just install libpqxx-dev. Other distros will likely have a similiar package.

Once you have the compile done, copy the sample cdr\_pgsql.conf file or create your own.

Here is a sample:

```
/etc/asterisk/cdr_pgsql.conf
; Sample Asterisk config file for CDR logging to PostgreSQL
[global]
hostname=localhost
port=5432
dbname=asterisk
password=password
user=postgres
table=cdr
```

Now create a table in postgresql for your cdrs

```
CREATE TABLE cdr (
    calldate      timestamp      NOT NULL ,
    clid          varchar (80)    NOT NULL ,
    src           varchar (80)    NOT NULL ,
    dst           varchar (80)    NOT NULL ,
    dcontext     varchar (80)    NOT NULL ,
    channel      varchar (80)    NOT NULL ,
    dstchannel   varchar (80)    NOT NULL ,
    lastapp      varchar (80)    NOT NULL ,
    lastdata     varchar (80)    NOT NULL ,
    duration     int             NOT NULL ,
    billsec     int             NOT NULL ,
    disposition  varchar (45)    NOT NULL ,
    amaflags     int             NOT NULL ,
    accountcode  varchar (20)    NOT NULL ,
    uniqueid    varchar (150)    NOT NULL ,
    userfield    varchar (255)   NOT NULL
);
```

#### 10.4.4 SQLite 2

SQLite version 2 is supported in cdr\_sqlite.

#### 10.4.5 SQLite 3

SQLite version 3 is supported in cdr\_sqlite3\_custom.

#### 10.4.6 RADIUS

##### What is needed

- FreeRADIUS server

- Radiusclient-ng library
- Asterisk PBX

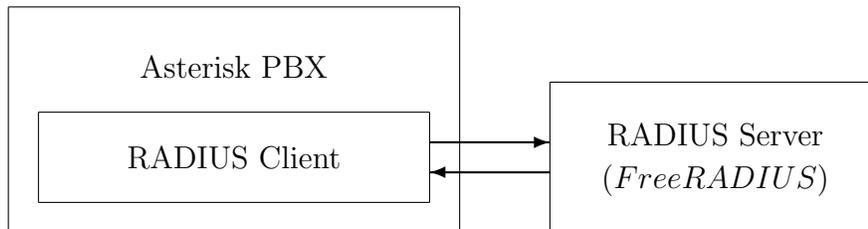


Figure 10.1: Asterisk/RADIUS Integration

### Installation of the Radiusclient library

Download the sources from <http://developer.berlios.de/projects/radiusclient-ng/>

Untar the source tarball:

```
root@localhost:/usr/local/src# tar xvfz radiusclient-ng-0.5.2.tar.gz
```

Compile and install the library:

```
root@localhost:/usr/local/src# cd radiusclient-ng-0.5.2
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# ./configure
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# make
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# make install
```

### Configuration of the Radiusclient library

By default all the configuration files of the radiusclient library will be in `/usr/local/etc/radiusclient-ng` directory.

File "radiusclient.conf" Open the file and find lines containing the following:

```
authserver localhost
```

This is the hostname or IP address of the RADIUS server used for authentication. You will have to change this unless the server is running on the same host as your Asterisk PBX.

acctserver localhost

This is the hostname or IP address of the RADIUS server used for accounting. You will have to change this unless the server is running on the same host as your Asterisk PBX.

#### **File "servers"**

RADIUS protocol uses simple access control mechanism based on shared secrets that allows RADIUS servers to limit access from RADIUS clients.

A RADIUS server is configured with a secret string and only RADIUS clients that have the same secret will be accepted.

You need to configure a shared secret for each server you have configured in radiusclient.conf file in the previous step. The shared secrets are stored in /usr/local/etc/radiusclient-ng/servers file.

Each line contains hostname of a RADIUS server and shared secret used in communication with that server. The two values are separated by white spaces. Configure shared secrets for every RADIUS server you are going to use.

#### **File "dictionary"**

Asterisk uses some attributes that are not included in the dictionary of radiusclient library, therefore it is necessary to add them. A file called dictionary.digium (kept in the contrib dir) was created to list all new attributes used by Asterisk. Add to the end of the main dictionary file /usr/local/etc/radiusclient-ng/dictionary the line:

```
$INCLUDE /path/to/dictionary.digium
```

### **Install FreeRADIUS Server (Version 1.1.1)**

Download sources tarball from:

```
http://freeradius.org/
```

Untar, configure, build, and install the server:

```
root@localhost:/usr/local/src# tar xvfz freeradius-1.1.1.tar.gz
root@localhost:/usr/local/src# cd freeradius-1.1.1
root@localhost"/usr/local/src/freeradius-1.1.1# ./configure
root@localhost"/usr/local/src/freeradius-1.1.1# make
root@localhost"/usr/local/src/freeradius-1.1.1# make install
```

All the configuration files of FreeRADIUS server will be in /usr/local/etc/raddb directory.

## Configuration of the FreeRADIUS Server

There are several files that have to be modified to configure the RADIUS server. These are presented next.

File "clients.conf"

File `/usr/local/etc/raddb/clients.conf` contains description of RADIUS clients that are allowed to use the server. For each of the clients you need to specify its hostname or IP address and also a shared secret. The shared secret must be the same string you configured in radiusclient library.

Example:

```
client myhost {
    secret = mysecret
    shortname = foo
}
```

This fragment allows access from RADIUS clients on "myhost" if they use "mysecret" as the shared secret. The file already contains an entry for localhost (127.0.0.1), so if you are running the RADIUS server on the same host as your Asterisk server, then modify the existing entry instead, replacing the default password.

File "dictionary"

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/radacct` directory in form of plain text files. The server will create one file for each hostname in the directory. The following example shows how the log files look like.

Asterisk now generates Call Detail Records. See `/include/asterisk/cdr.h` for all the fields which are recorded. By default, records in comma separated values will be created in `/var/log/asterisk/cdr-csv`.

The configuration file for `cdr_radius.so` module is `/etc/asterisk/cdr.conf`

This is where you can set CDR related parameters as well as the path to the `radiusclient-ng` library configuration file.

### Logged Values

"Asterisk-Acc-Code",	The account name of detail records
"Asterisk-Src",	
"Asterisk-Dst",	
"Asterisk-Dst-Ctx",	The destination context
"Asterisk-Clid",	
"Asterisk-Chan",	The channel
"Asterisk-Dst-Chan",	(if applicable)
"Asterisk-Last-App",	Last application run on the channel
"Asterisk-Last-Data",	Argument to the last channel
"Asterisk-Start-Time",	
"Asterisk-Answer-Time",	
"Asterisk-End-Time",	
"Asterisk-Duration",	Duration is the whole length that the entire call lasted. ie. call rx'd to hangup "end time" minus "start time"
"Asterisk-Bill-Sec",	The duration that a call was up after other end answered which will be <= to duration "end time" minus "answer time"
"Asterisk-Disposition",	ANSWERED, NO ANSWER, BUSY
"Asterisk-AMA-Flags",	DOCUMENTATION, BILL, IGNORE etc, specified on a per channel basis like accountcode.
"Asterisk-Unique-ID",	Unique call identifier
"Asterisk-User-Field"	User field set via SetCDRUserField

# Chapter 11

## CEL: Channel Event Logging

### 11.1 Design Goals

CEL, or Channel Event Logging, has been written with the hopes that it will help solve some of the problems that were difficult to address in CDR records. Some difficulties in CDR generation are the fact that the CDR record stores three events: the "Start" time, the "Answer" time, and the "End" time. Billing time is usually the difference between "Answer" and "End", and total call duration was the difference in time from "Start" to "End". The trouble with this direct and simple approach is the fact that calls can be transferred, put on hold, conferenced, forwarded, etc. In general, those doing billing applications in Asterisk find they have to do all sorts of very creative things to overcome the shortcomings of CDR records, often supplementing the CDR records with AGI scripts and manager event filters.

The fundamental assumption is that the Channel is the fundamental communication object in asterisk, which basically provides a communication channel between two communication ports. It makes sense to have an event system aimed at recording important events on channels. Each event is attached to a channel, like ANSWER or HANGUP. Some events are meant to connect two or more channels, like the BRIDGE\_START event. Some events, like BLINDTRANSFER, are initiated by one channel, but affect two others. These events use the Peer field, like BRIDGE would, to point to the target channel.

The design philosophy of CEL is to generate event data that can grouped together to form a billing record. This may not be a simple task, but we

hope to provide a few different examples that could be used as a basis for those involved in this effort.

There are definite parallels between Manager events and CEL events, but there are some differences. Some events that are generated by CEL are not generated by the Manager interface (yet). CEL is optimized for databases, and Manager events are not. The focus of CEL is billing. The Manager interface is targeted to real-time monitoring and control of asterisk.

To give the reader a feel for the complexities involved in billing, please take note of the following sequence of events:

Remember that 150, 151, and 152 are all Zap extension numbers, and their respective devices are Zap/50, Zap/51, and Zap/52.

152 dials 151; 151 answers. 152 parks 151; 152 hangs up. 150 picks up the park (dials 701). 150 and 151 converse. 151 flashes hook; dials 152, talks to 152, then 151 flashes hook again for 3-way conference. 151 converses with the other two for a while, then hangs up. 150 and 152 keep conversing, then hang up. 150 hangs up first.(not that it matters).

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

*Note that the actual CEL events below are in CSV format and do not include the ;;; and text after that which gives a description of what the event represents.*

```
"EV_CHAN_START","2007-05-09 12:46:16","fxs.52","152","","","s","extension","Zap/52-1","","","DOCUMENTATION","","1178736376.3","","" ;;; 152 takes the phone off-hook
```

```
"EV_APP_START","2007-05-09 12:46:18","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51-30-TtWw","DOCUMENTATION","","1178736376.3" ;;; 152 finishes dialing 151
```

```
"EV_CHAN_START","2007-05-09 12:46:18","fxs.51","151","","","s","extension","Zap/51-1","","","DOCUMENTATION","","1178736378.4","","" ;;; 151 channel created, starts ringing
```

*(151 is ringing)*

```
"EV_ANSWER","2007-05-09 12:46:19","","151","152","","","151","extension","Zap/51-1","AppDial","(Outgoing Line)","DOCUMENTATION","","1178736378.4","","" ;;; 151 answers
```

```
"EV_ANSWER","2007-05-09 12:46:19","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51-30-TtWw","DOCUMENTATION","","1178736376.3","","" ;;; so does 152 (???)
```

```
"EV_BRIDGE_START","2007-05-09 12:46:20","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51-30-TtWw","DOCUMENTATION","","1178736376.3","","","Zap/51-1" ;;; 152 and 151 are bridged
```

*(151 and 152 are conversing)*

"EV\_BRIDGE\_END","2007-05-09 12:46:25","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51—30—TtWw","DOCUMENTATION","","1178736376.3","","" ;; after 5 seconds, the bridge ends (152 dials #700?)

"EV\_BRIDGE\_START","2007-05-09 12:46:25","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51—30—TtWw","DOCUMENTATION","","1178736376.3","","Zap/51-1" ;; extraneous 0-second bridge?

"EV\_BRIDGE\_END","2007-05-09 12:46:25","fxs.52","152","152","","","151","extension","Zap/52-1","Dial","Zap/51—30—TtWw","DOCUMENTATION","","1178736376.3","","" ;;

"EV\_PARK\_START","2007-05-09 12:46:27","","151","152","","","extension","Zap/51-1","Parked Call","","DOCUMENTATION","","1178736378.4","","" ;; 151 is parked

"EV\_HANGUP","2007-05-09 12:46:29","fxs.52","152","152","","","h","extension","Zap/52-1","","","DOCUMENTATION","","1178736376.3","","" ;; 152 hangs up 2 sec later

"EV\_CHAN\_END","2007-05-09 12:46:29","fxs.52","152","152","","","h","extension","Zap/52-1","","","DOCUMENTATION","","1178736376.3","","" ;; 152's channel goes away

*(151 is parked and listening to MOH! now, 150 picks up, and dials 701)*

"EV\_CHAN\_START","2007-05-09 12:47:08","fxs.50","150","","","s","extension","Zap/50-1","","","DOCUMENTATION","","1178736428.5","","" ;; 150 picks up the phone, dials 701

"EV\_PARK\_END","2007-05-09 12:47:11","","151","152","","","extension","Zap/51-1","Parked Call","","DOCUMENTATION","","1178736378.4","","" ;; 151's park comes to end

"EV\_ANSWER","2007-05-09 12:47:11","fxs.50","150","150","","","701","extension","Zap/50-1","ParkedCall","701","DOCUMENTATION","","1178736428.5","","" ;; 150 gets answer (twice)

"EV\_ANSWER","2007-05-09 12:47:12","fxs.50","150","150","","","701","extension","Zap/50-1","ParkedCall","701","DOCUMENTATION","","1178736428.5","","" ;;

"EV\_BRIDGE\_START","2007-05-09 12:47:12","fxs.50","150","150","","","701","extension","Zap/50-1","ParkedCall","701","DOCUMENTATION","","1178736428.5","","Zap/51-1" ;; bridge begins between 150 and recently parked 151

*(150 and 151 are conversing, then 151 hits flash)*

"EV\_CHAN\_START","2007-05-09 12:47:51","fxs.51","151","","","s","extension","Zap/51-2","","","DOCUMENTATION","","1178736471.6","","" ;; 39 seconds later, 51-2 channel is created. (151 flashes hook)

"EV\_HOOKFLASH","2007-05-09 12:47:51","","151","152","","","extension","Zap/51-1","Bridged Call","Zap/50-1","DOCUMENTATION","","1178736378.4","","Zap/51-2" ;; a marker to record that 151 flashed the hook

"EV\_BRIDGE\_END","2007-05-09 12:47:51","fxs.50","150","150","","","701","extension","Zap/50-1","ParkedCall","701","DOCUMENTATION","","1178736428.5","","Zap/51-1" ;; bridge ends between 150 and 151

"EV\_BRIDGE\_START","2007-05-09 12:47:51","fxs.50","150","150","","","701","extension","Zap/50-1","ParkedCall","701","DOCUMENTATION","","1178736428.5","","Zap/51-1" ;; 0-second bridge from 150 to ? 150 gets no sound at all

"EV\_BRIDGE\_END","2007-05-09 12:47:51","fxs.50","150","150","","","701","extension","Zap/50-1","ParkedCall","701","DOCUMENTATION","","1178736428.5","","Zap/51-1" ;;

"EV\_BRIDGE\_START","2007-05-09 12:47:51","fxs.50","150","150","","","701","extension","Zap/50-1","ParkedCall","701","DOCUMENTATION","","1178736428.5","","Zap/51-1" ;; bridge start on 150

*(151 has dialtone after hitting flash; dials 152)*

"EV\_APP\_START", "2007-05-09 12:47:55", "fxs.51", "151", "151", "", "", "152", "extension", "Zap/51-2", "Dial", "Zap/52—30—TtWw", "DOCUMENTATION", "", "1178736471.6", "", "" ;; 151-2 dials 152 after 4 seconds

"EV\_CHAN\_START", "2007-05-09 12:47:55", "fxs.52", "152", "", "", "s", "extension", "Zap/52-1", "", "", "DOCUMENTATION", "", "1178736475.7", "", "" ;; 152 channel created to ring 152.

*(152 ringing)*

"EV\_ANSWER", "2007-05-09 12:47:58", "", "152", "151", "", "", "152", "extension", "Zap/52-1", "AppDial", "(Outgoing Line)", "DOCUMENTATION", "", "1178736475.7", "", "" ;; 3 seconds later, 152 answers

"EV\_ANSWER", "2007-05-09 12:47:58", "fxs.51", "151", "151", "", "", "152", "extension", "Zap/51-2", "Dial", "Zap/52—30—TtWw", "DOCUMENTATION", "", "1178736471.6", "", "" ;; ... and 151-2 also answers

"EV\_BRIDGE\_START", "2007-05-09 12:47:59", "fxs.51", "151", "151", "", "", "152", "extension", "Zap/51-2", "Dial", "Zap/52—30—TtWw", "DOCUMENTATION", "", "1178736471.6", "", "Zap/51-1" ;; 1 second later, bridge formed betw. 151-2 and 151

*(152 answers, 151 and 152 converging; 150 is listening to silence; 151 hits flash again... to start a 3way)*

"EV\_3WAY\_START", "2007-05-09 12:48:58", "", "151", "152", "", "", "extension", "Zap/51-1", "Bridged Call", "Zap/50-1", "DOCUMENTATION", "", "1178736378.4", "", "Zap/51-2" ;; another hook-flash to begin a 3-way conference

"EV\_BRIDGE\_END", "2007-05-09 12:48:58", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; - almost 1 minute later, the bridge ends (151 flashes hook again)

"EV\_BRIDGE\_START", "2007-05-09 12:48:58", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; 0-second bridge at 150. (3 way conf formed)

"EV\_BRIDGE\_END", "2007-05-09 12:48:58", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;;

"EV\_BRIDGE\_START", "2007-05-09 12:48:58", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; bridge starts for 150

*(3way now, then 151 hangs up.)*

"EV\_BRIDGE\_END", "2007-05-09 12:49:26", "fxs.50", "150", "150", "", "", "701", "extension", "Zap/50-1", "ParkedCall", "701", "DOCUMENTATION", "", "1178736428.5", "", "Zap/51-1" ;; 28 seconds later, bridge ends

"EV\_HANGUP", "2007-05-09 12:49:26", "", "151", "152", "", "", "extension", "Zap/51-1", "Bridged Call", "Zap/50-1", "DOCUMENTATION", "", "1178736378.4", "", "" ;; 151 hangs up, leaves 150 and 152 connected

"EV\_CHAN\_END", "2007-05-09 12:49:26", "", "151", "152", "", "", "extension", "Zap/51-1", "Bridged Call", "Zap/50-1", "DOCUMENTATION", "", "1178736378.4", "", "" ;; 151 channel ends

"EV\_CHAN\_END", "2007-05-09 12:49:26", "fxs.51", "151", "151", "", "", "h", "extension", "Zap/51-2<ZOMBIE>", "", "", "DOCUMENTATION", "", "1178736428.5", "", "" ;; 152-2 channel ends (zombie)

*(just 150 and 152 now)*

```

"EV_BRIDGE_END","2007-05-09 12:50:13","fxs.50","150","150","","","152","extension","Zap/50-1","Dial","Zap/52-30-TtWw","DOCUMENTATION","","1178736471.6","","" ;; 47
sec later, the bridge from 150 to 152 ends

"EV_HANGUP","2007-05-09 12:50:13","","152","151","","","extension","Zap/52-1","Bridged Call","Zap/50-1","DOCUMENTATION","","1178736475.7","","" ;; 152 hangs up

"EV_CHAN_END","2007-05-09 12:50:13","","152","151","","","extension","Zap/52-1","Bridged Call","Zap/50-1","DOCUMENTATION","","1178736475.7","","" ;; 152 channel ends

"EV_HANGUP","2007-05-09 12:50:13","fxs.50","150","150","","","h","extension","Zap/50-1","","","DOCUMENTATION","","1178736471.6","","" ;; 150 hangs up

"EV_CHAN_END","2007-05-09 12:50:13","fxs.50","150","150","","","h","extension","Zap/50-1","","","DOCUMENTATION","","1178736471.6","","" ;; 150 ends

```

In terms of Manager events, the above Events correspond to the following 80 Manager events:

```

Event: Newchannel
Privilege: call,all
Channel: Zap/52-1
State: Rsvrd
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801102.5

Event: Newcallerid
Privilege: call,all
Channel: Zap/52-1
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801102.5
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newcallerid
Privilege: call,all
Channel: Zap/52-1
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801102.5
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newstate
Privilege: call,all
Channel: Zap/52-1
State: Ring
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801102.5

Event: Newexten
Privilege: call,all
Channel: Zap/52-1
Context: extension
Extension: 151
Priority: 1

```

Application: Set  
AppData: CDR(myvar)=zingo  
Uniqueid: 1178801102.5

Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: 151  
Priority: 2  
Application: Dial  
AppData: Zap/51|30|TtWw  
Uniqueid: 1178801102.5

Event: Newchannel  
Privilege: call,all  
Channel: Zap/51-1  
State: Rsrvd  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801108.6

Event: Newstate  
Privilege: call,all  
Channel: Zap/51-1  
State: Ringing  
CallerIDNum: 152  
CallerIDName: fxs.52  
Uniqueid: 1178801108.6

Event: Dial  
Privilege: call,all  
SubEvent: Begin  
Source: Zap/52-1  
Destination: Zap/51-1  
CallerIDNum: 152  
CallerIDName: fxs.52  
SrcUniqueID: 1178801102.5  
DestUniqueID: 1178801108.6

Event: Newcallerid  
Privilege: call,all  
Channel: Zap/51-1  
CallerIDNum: 151  
CallerIDName: <Unknown>  
Uniqueid: 1178801108.6  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newstate  
Privilege: call,all  
Channel: Zap/52-1  
State: Ringing  
CallerIDNum: 152  
CallerIDName: fxs.52  
Uniqueid: 1178801102.5

Event: Newstate

Privilege: call,all  
Channel: Zap/51-1  
State: Up  
CallerIDNum: 151  
CallerIDName: <unknown>  
Uniqueid: 1178801108.6

Event: Newstate  
Privilege: call,all  
Channel: Zap/52-1  
State: Up  
CallerIDNum: 152  
CallerIDName: fxs.52  
Uniqueid: 1178801102.5

Event: Link  
Privilege: call,all  
Channel1: Zap/52-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801102.5  
Uniqueid2: 1178801108.6  
CallerID1: 152  
CallerID2: 151

Event: Unlink  
Privilege: call,all  
Channel1: Zap/52-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801102.5  
Uniqueid2: 1178801108.6  
CallerID1: 152  
CallerID2: 151

Event: Link  
Privilege: call,all  
Channel1: Zap/52-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801102.5  
Uniqueid2: 1178801108.6  
CallerID1: 152  
CallerID2: 151

Event: Unlink  
Privilege: call,all  
Channel1: Zap/52-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801102.5  
Uniqueid2: 1178801108.6  
CallerID1: 152  
CallerID2: 151

Event: ParkedCall  
Privilege: call,all  
Exten: 701  
Channel: Zap/51-1  
From: Zap/52-1  
Timeout: 45

CallerIDNum: 151  
CallerIDName: <unknown>

Event: Dial  
Privilege: call,all  
SubEvent: End  
Channel: Zap/52-1  
DialStatus: ANSWER

Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: h  
Priority: 1  
Application: Goto  
AppData: label1  
Uniqueid: 1178801102.5

Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: h  
Priority: 4  
Application: Goto  
AppData: label2  
Uniqueid: 1178801102.5

Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: h  
Priority: 2  
Application: NoOp  
AppData: In Hangup! myvar is zingo and accountcode is billsec is 26 and duration is 40 and end is 2007-05-10 06:45  
Uniqueid: 1178801102.5

Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: h  
Priority: 3  
Application: Goto  
AppData: label3  
Uniqueid: 1178801102.5

Event: Newexten  
Privilege: call,all  
Channel: Zap/52-1  
Context: extension  
Extension: h  
Priority: 5  
Application: NoOp  
AppData: More Hangup message after hopping around"

Uniqueid: 1178801102.5

Event: Hangup  
Privilege: call,all  
Channel: Zap/52-1  
Uniqueid: 1178801102.5  
Cause: 16  
Cause-txt: Normal Clearing

Event: Newchannel  
Privilege: call,all  
Channel: Zap/50-1  
State: Rsrvd  
CallerIDNum: 150  
CallerIDName: fxs.50  
Uniqueid: 1178801162.7

Event: Newcallerid  
Privilege: call,all  
Channel: Zap/50-1  
CallerIDNum: 150  
CallerIDName: fxs.50  
Uniqueid: 1178801162.7  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newcallerid  
Privilege: call,all  
Channel: Zap/50-1  
CallerIDNum: 150  
CallerIDName: fxs.50  
Uniqueid: 1178801162.7  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newstate  
Privilege: call,all  
Channel: Zap/50-1  
State: Ring  
CallerIDNum: 150  
CallerIDName: fxs.50  
Uniqueid: 1178801162.7

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: 701  
Priority: 1  
Application: ParkedCall  
AppData: 701  
Uniqueid: 1178801162.7

Event: UnParkedCall  
Privilege: call,all  
Exten: 701  
Channel: Zap/51-1  
From: Zap/50-1  
CallerIDNum: 151

CallerIDName: <unknown>

Event: Newstate  
Privilege: call,all  
Channel: Zap/50-1  
State: Up  
CallerIDNum: 150  
CallerIDName: fxs.50  
Uniqueid: 1178801162.7

Event: Link  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Newchannel  
Privilege: call,all  
Channel: Zap/51-2  
State: Rsrvd  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8

Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Link  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Link  
Privilege: call,all  
Channel1: Zap/50-1

Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Newcallerid  
Privilege: call,all  
Channel: Zap/51-2  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newcallerid  
Privilege: call,all  
Channel: Zap/51-2  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newstate  
Privilege: call,all  
Channel: Zap/51-2  
State: Ring  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8

Event: Newexten  
Privilege: call,all  
Channel: Zap/51-2  
Context: extension  
Extension: 152  
Priority: 1  
Application: Set  
AppData: CDR(myvar)=zingo  
Uniqueid: 1178801218.8

Event: Newexten  
Privilege: call,all  
Channel: Zap/51-2  
Context: extension  
Extension: 152  
Priority: 2  
Application: Dial  
AppData: Zap/52|30|TtWw  
Uniqueid: 1178801218.8

Event: Newchannel  
Privilege: call,all  
Channel: Zap/52-1  
State: Rsrvd  
CallerIDNum: 152  
CallerIDName: fxs.52  
Uniqueid: 1178801223.9

Event: Newstate  
Privilege: call,all  
Channel: Zap/52-1  
State: Ringing  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801223.9

Event: Dial  
Privilege: call,all  
SubEvent: Begin  
Source: Zap/51-2  
Destination: Zap/52-1  
CallerIDNum: 151  
CallerIDName: fxs.51  
SrcUniqueID: 1178801218.8  
DestUniqueID: 1178801223.9

Event: Newcallerid  
Privilege: call,all  
Channel: Zap/52-1  
CallerIDNum: 152  
CallerIDName: <Unknown>  
Uniqueid: 1178801223.9  
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newstate  
Privilege: call,all  
Channel: Zap/51-2  
State: Ringing  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8

Event: Newstate  
Privilege: call,all  
Channel: Zap/52-1  
State: Up  
CallerIDNum: 152  
CallerIDName: <unknown>  
Uniqueid: 1178801223.9

Event: Newstate  
Privilege: call,all  
Channel: Zap/51-2  
State: Up  
CallerIDNum: 151  
CallerIDName: fxs.51  
Uniqueid: 1178801218.8

Event: Link  
Privilege: call,all  
Channel1: Zap/51-2  
Channel2: Zap/52-1  
Uniqueid1: 1178801218.8  
Uniqueid2: 1178801223.9

CallerID1: 151  
CallerID2: 152

Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Link  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Link  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/51-1  
Uniqueid1: 1178801162.7  
Uniqueid2: 1178801108.6  
CallerID1: 150  
CallerID2: 151

Event: Hangup  
Privilege: call,all  
Channel: Zap/51-1  
Uniqueid: 1178801108.6  
Cause: 16  
Cause-txt: Normal Clearing

Event: Newexten  
Privilege: call,all

Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 1  
Application: Goto  
AppData: label1  
Uniqueid: 1178801162.7

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 4  
Application: Goto  
AppData: label2  
Uniqueid: 1178801162.7

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 2  
Application: NoOp  
AppData: In Hangup! myvar is and accountcode is billsec is 0 and duration is 0 and end is 2007-05-10 06:48:37.  
Uniqueid: 1178801162.7

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 3  
Application: Goto  
AppData: label3  
Uniqueid: 1178801162.7

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 5  
Application: NoOp  
AppData: More Hangup message after hopping around"  
Uniqueid: 1178801162.7

Event: Masquerade  
Privilege: call,all  
Clone: Zap/50-1  
CloneState: Up  
Original: Zap/51-2  
OriginalState: Up

Event: Rename  
Privilege: call,all

Oldname: Zap/50-1  
Newname: Zap/50-1<MASQ>  
Uniqueid: 1178801162.7

Event: Rename  
Privilege: call,all  
Oldname: Zap/51-2  
Newname: Zap/50-1  
Uniqueid: 1178801218.8

Event: Rename  
Privilege: call,all  
Oldname: Zap/50-1<MASQ>  
Newname: Zap/51-2<ZOMBIE>  
Uniqueid: 1178801162.7

Event: Hangup  
Privilege: call,all  
Channel: Zap/51-2<ZOMBIE>  
Uniqueid: 1178801162.7  
Cause: 0  
Cause-txt: Unknown

Event: Unlink  
Privilege: call,all  
Channel1: Zap/50-1  
Channel2: Zap/52-1  
Uniqueid1: 1178801218.8  
Uniqueid2: 1178801223.9  
CallerID1: 150  
CallerID2: 152

Event: Hangup  
Privilege: call,all  
Channel: Zap/52-1  
Uniqueid: 1178801223.9  
Cause: 16  
Cause-txt: Normal Clearing

Event: Dial  
Privilege: call,all  
SubEvent: End  
Channel: Zap/50-1  
DialStatus: ANSWER

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 1  
Application: Goto  
AppData: label1  
Uniqueid: 1178801218.8

Event: Newexten  
Privilege: call,all

Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 4  
Application: Goto  
AppData: label2  
Uniqueid: 1178801218.8

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 2  
Application: NoOp  
AppData: In Hangup! myvar is and accountcode is billsec is 90 and duration is 94 and end is 2007-05-10 06:48:37.  
Uniqueid: 1178801218.8

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 3  
Application: Goto  
AppData: label3  
Uniqueid: 1178801218.8

Event: Newexten  
Privilege: call,all  
Channel: Zap/50-1  
Context: extension  
Extension: h  
Priority: 5  
Application: NoOp  
AppData: More Hangup message after hopping around"  
Uniqueid: 1178801218.8

Event: Hangup  
Privilege: call,all  
Channel: Zap/50-1  
Uniqueid: 1178801218.8  
Cause: 16  
Cause-txt: Normal Clearing

And, humorously enough, the above 80 manager events, or 42 CEL events, correspond to the following two CDR records (at the moment!):

""fxs.52" <152>","152","h","extension","Zap/52-1","Zap/51-1","NoOp","More Hangup message after hopping around""","2007-05-09 17:35:56","2007-05-09 17:36:20","2007-05-09 17:36:36","40","16","ANSWERED","DOCUMENTATION","","","1178753756.0",""

""fxs.50" <150>","150","152","extension","Zap/50-1","Zap/51-1","NoOp","More Hangup message after hopping around""","2007-05-09 17:37:59","2007-05-09 17:38:06","2007-05-09 17:39:11","72","65","ANSWERED","DOCUMENTATION","","","1178753871.3",""

## 11.2 Events & Fields

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

See table 11.1 for a list of events raised by CEL and table 11.2 for the list of fields passed for each CEL event.

Event	Description
CHAN_START	The time a channel was created
CHAN_END	The time a channel was terminated
ANSWER	The time a channel was answered (ie, phone taken off-hook, etc)
HANGUP	The time at which a hangup occurred.
CONF_ENTER	The time a channel was connected into a conference room
CONF_EXIT	The time a channel was removed from a conference room
CONF_START	The time the first person enters a conference
CONF_END	The time the last person left a conf (and turned out the lights?)
APP_START	The time a tracked application was started
APP_END	the time a tracked application ended
PARK_START	The time a call was parked
PARK_END	unpark event
BRIDGE_START	The time a bridge is started
BRIDGE_END	The time a bridge is ended
3WAY_START	When a 3-way conf starts (usually via attended xfer)
3WAY_END	When one or all exit a 3-way conf
BLINDTRANSFER	When a blind transfer is initiated
ATTENDEDTRANSFER	When an attended transfer is initiated
TRANSFER	Generic transfer initiated; not used yet...?
HOOKFLASH	So far, when a hookflash event occurs on a Zap interface
USER_EVENT	these are triggered from the dialplan, and have a name given by the user.

Table 11.1: List of CEL Events

Field	Description
eventtype	The name of the event; see the above list; each is prefixed with "EV_".
eventtime	The time the event happened
cidname	CID name field
cidnum	CID number field
cidani	CID ANI field
cidrdnis	CID RDNIS field
ciddnid	CID DNID field
exten	The extension in the dialplan
context	The context in the dialplan
channname	The name assigned to the channel in which the event took place
appname	The name of the current application
appdata	The arguments that will be handed to that application
amaflags	The AMA flags associated with the event; user assignable.
accountcode	A user assigned datum (string)
uniqueid	Each Channel instance gets a unique ID associated with it.
userfield	A user assigned datum (string)
linkedid	the per-call id, spans several events, possibly.
peer	For bridge or other 2-channel events, this would be the other channel name

Table 11.2: List of CEL Event Fields

## 11.3 Applications & Functions

### 11.3.1 CEL Function

\*\*\*\* THIS IS NO LONGER TRUE. REWRITE. \*\*\*\*

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

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

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

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

<code>cidname</code>	<code>userfield</code>
<code>cidnum</code>	<code>amaflags</code>
<code>cidani</code>	<code>cidrdnis</code>
<code>ciddnid</code>	<code>appdata</code>
<code>exten</code>	<code>accountcode</code>
<code>context</code>	<code>uniqueid</code>
<code>channname</code>	<code>appname</code>
<code>peer</code>	<code>eventtime</code>
<code>eventtype</code>	

### 11.3.2 CELGenUserEvent Application

This application allows the dialplan to insert custom events into the event stream.

For more information, in the CLI, type: `core show application CELGenUserEvent`

Its arguments take this format:

```
CELGenUserEvent(eventname)
```

Please note that there is no restrictions on the name supplied. If it happens to match a standard CEL event name, it will look like that event was generated. This could be a blessing or a curse!

## 11.4 Configuration Files

- `cel.conf`

## 11.5 Generating Billing Information

\*\*\* This is the Next Big Task \*\*\*

## 11.6 Storage Backends

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

### 11.6.1 Microsoft SQL Server

Asterisk can currently store Channel Events into an MSSQL database in two different ways: cel\_odbc or cel\_tds

Channel Event Records can be stored using unixODBC (which requires the FreeTDS package) [cel\_odbc] or directly by using just the FreeTDS package [cel\_tds] The following provide some examples known to get asterisk working with mssql.

NOTE: Only choose one db connector.

#### ODBC using cel\_odbc

Compile, configure, and install the latest unixODBC package:

```
tar -zxvf unixODBC-2.2.9.tar.gz &&
cd unixODBC-2.2.9 &&
./configure --sysconfdir=/etc --prefix=/usr --disable-gui &&
make &&
make install
```

Compile, configure, and install the latest FreeTDS package:

```
tar -zxvf freetds-0.62.4.tar.gz &&
cd freetds-0.62.4 &&
./configure --prefix=/usr --with-tdsver=7.0 \
            --with-unixodbc=/usr/lib &&
make && make install
```

Compile, or recompile, asterisk so that it will now add support for cel\_odbc.

```
make clean && ./configure --with-odbc &&
make update &&
make &&
make install
```

Setup odbc configuration files. These are working examples from my system. You will need to modify for your setup. You are not required to store usernames or passwords here.

```
/etc/odbcinst.ini
[FreeTDS]
Description      = FreeTDS ODBC driver for MSSQL
Driver           = /usr/lib/libtdsodbc.so
Setup            = /usr/lib/libtdsS.so
FileUsage        = 1
```

```
/etc/odbc.ini
[MSSQL-asterisk]
description      = Asterisk ODBC for MSSQL
driver           = FreeTDS
server          = 192.168.1.25
port             = 1433
database        = voipdb
tds_version     = 7.0
language        = us_english
```

Only install one database connector. Do not confuse asterisk by using both ODBC (cel\_odbc) and FreeTDS (cel\_tds). This command will erase the contents of cel\_tds.conf

```
[ -f /etc/asterisk/cel_tds.conf ] > /etc/asterisk/cel_tds.conf
```

NOTE: unixODBC requires the freeTDS package, but asterisk does not call freeTDS directly.

Now set up cel\_odbc configuration files. These are working samples from my system. You will need to modify for your setup. Define your usernames and passwords here, secure file as well.

```
/etc/asterisk/cel_odbc.conf
```

```

[global]
dsn=MSSQL-asterisk
username=voipdbuser
password=voipdbpass
loguniqueid=yes

```

And finally, create the 'cel' table in your mssql database.

```

CREATE TABLE cel (
[eventtype]      [varchar] (30)          NOT NULL ,
[eventtime]      [datetime]             NOT NULL ,
[cidname]        [varchar] (80)         NOT NULL ,
[cidnum]         [varchar] (80)         NOT NULL ,
[cidani]         [varchar] (80)         NOT NULL ,
[cidrdsn]        [varchar] (80)         NOT NULL ,
[ciddnid]        [varchar] (80)         NOT NULL ,
[exten]          [varchar] (80)         NOT NULL ,
[context]        [varchar] (80)         NOT NULL ,
[channname]      [varchar] (80)         NOT NULL ,
[appname]        [varchar] (80)         NOT NULL ,
[appdata]        [varchar] (80)         NOT NULL ,
[amaflags]       [int]                  NOT NULL ,
[accountcode]    [varchar] (20)         NOT NULL ,
[uniqueid]       [varchar] (32)         NOT NULL ,
[peer]           [varchar] (80)         NOT NULL ,
[userfield]      [varchar] (255)        NOT NULL
)

```

Start asterisk in verbose mode, you should see that asterisk logs a connection to the database and will now record every desired channel event at the moment it occurs.

### FreeTDS, using cel\_tds

Compile, configure, and install the latest FreeTDS package:

```

tar -zxvf freetds-0.62.4.tar.gz &&
cd freetds-0.62.4 &&
./configure --prefix=/usr --with-tdsver=7.0
make &&

```

```
make install
```

Compile, or recompile, asterisk so that it will now add support for cel\_tds.

```
make clean && ./configure --with-tds &&  
make update &&  
make &&  
make install
```

Only install one database connector. Do not confuse asterisk by using both ODBC (cel\_odbc) and FreeTDS (cel\_tds). This command will erase the contents of cel\_odbc.conf

```
[ -f /etc/asterisk/cel_odbc.conf ] > /etc/asterisk/cel_odbc.conf
```

Setup cel\_tds configuration files. These are working samples from my system. You will need to modify for your setup. Define your usernames and passwords here, secure file as well.

```
                /etc/asterisk/cel_tds.conf  
[global]  
hostname=192.168.1.25  
port=1433  
dbname=voipdb  
user=voipdbuser  
password=voipdbpass  
charset=BIG5
```

And finally, create the 'cel' table in your mssql database.

```
CREATE TABLE cel (  
[eventtype]      [varchar] (30)          NULL ,  
                [eventtime]      [datetime]          NULL ,  
                [cidname]        [varchar] (80)         NULL ,  
                [cidnum]         [varchar] (80)         NULL ,  
                [cidani]         [varchar] (80)         NULL ,  
                [cidrdnis]       [varchar] (80)         NULL ,  
                [ciddnid]        [varchar] (80)         NULL ,  
                [exten]          [varchar] (80)         NULL ,  
                [context]        [varchar] (80)         NULL ,  
                [channname]      [varchar] (80)         NULL ,
```

```

        [appname]      [varchar] (80)      NULL ,
        [appdata]     [varchar] (80)      NULL ,
        [amaflags]    [varchar] (16)      NULL ,
        [accountcode] [varchar] (20)      NULL ,
        [uniqueid]    [varchar] (32)      NULL ,
        [userfield]   [varchar] (255)     NULL ,
        [peer]        [varchar] (80)      NULL
    )

```

Start asterisk in verbose mode, you should see that asterisk logs a connection to the database and will now record every call to the database when it's complete.

## 11.6.2 MySQL

Using MySQL for Channel Event records is supported by using ODBC and the cel\_odbc module.

## 11.6.3 PostgreSQL

If you want to go directly to postgresql database, and have the cel\_pgsql.so compiled you can use the following sample setup. On Debian, before compiling asterisk, just install libpqxx-dev. Other distros will likely have a similiar package.

Once you have the compile done, copy the sample cel\_pgsql.conf file or create your own.

Here is a sample:

```

/etc/asterisk/cel_pgsql.conf
; Sample Asterisk config file for CEL logging to PostgreSQL
[global]
hostname=localhost
port=5432
dbname=asterisk
password=password
user=postgres
table=cel

```

Now create a table in postgresql for your cels

```

CREATE TABLE cel (
    id          serial ,
    eventtype   varchar (30)          NOT NULL ,
    eventtime   timestamp            NOT NULL ,
    userdeftype varchar(255)          NOT NULL ,
    cid_name    varchar (80)          NOT NULL ,
    cid_num     varchar (80)          NOT NULL ,
    cid_ani     varchar (80)          NOT NULL ,
    cid_rdnis   varchar (80)          NOT NULL ,
    cid_dnid    varchar (80)          NOT NULL ,
    exten       varchar (80)          NOT NULL ,
    context     varchar (80)          NOT NULL ,
    channame    varchar (80)          NOT NULL ,
    appname     varchar (80)          NOT NULL ,
    appdata     varchar (80)          NOT NULL ,
    amaflags    int                  NOT NULL ,
    accountcode varchar (20)          NOT NULL ,
    peeraccount  varchar (20)          NOT NULL ,
    uniqueid    varchar (150)         NOT NULL ,
    linkedid    varchar (150)         NOT NULL ,
    userfield   varchar (255)         NOT NULL ,
    peer        varchar (80)          NOT NULL
);

```

### 11.6.4 SQLite 3

SQLite version 3 is supported in cel\_sqlite3\_custom.

### 11.6.5 RADIUS

#### What is needed

- FreeRADIUS server
- Radiusclient-ng library
- Asterisk PBX

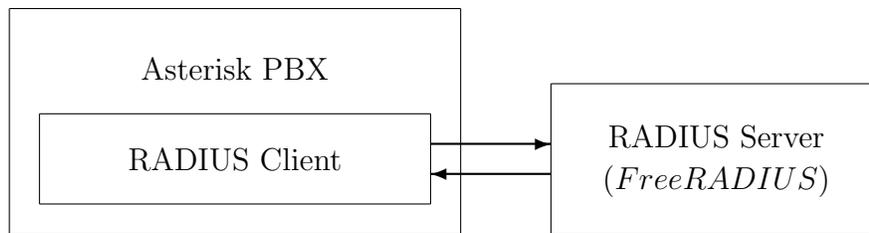


Figure 11.1: Asterisk/RADIUS Integration

### 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-0.5.5.1.tar.gz
```

Compile and install the library.

```
root@localhost:/usr/local/src# cd radiusclient-ng-0.5.5.1
root@localhost:/usr/local/src/radiusclient-ng-0.5.5.1# ./configure
root@localhost:/usr/local/src/radiusclient-ng-0.5.5.1# make
root@localhost:/usr/local/src/radiusclient-ng-0.5.5.1# 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/cel.h` for all the fields which are recorded. By default, records in comma separated values will be created in `/var/log/asterisk/cel-csv`.

The configuration file for `cel_radius.so` module is :

`/etc/asterisk/cel.conf` This is where you can set CEL related parameters as well as the path to the `radiusclient-ng` library configuration file.

## Logged Values

"Asterisk-Acc-Code",	The account name of detail records
"Asterisk-CidName",	
"Asterisk-CidNum",	
"Asterisk-Cidani",	
"Asterisk-Cidrdnis",	
"Asterisk-Ciddnid",	
"Asterisk-Exten",	
"Asterisk-Context",	The destination context
"Asterisk-Channame",	The channel name
"Asterisk-Appname",	Last application run on the channel
"Asterisk-App-Data",	Argument to the last channel
"Asterisk-Event-Time",	
"Asterisk-Event-Type",	
"Asterisk-AMA-Flags",	DOCUMENTATION, BILL, IGNORE etc, specified on a per channel basis like accountcode.
"Asterisk-Unique-ID",	Unique call identifier
"Asterisk-User-Field"	User field set via <code>SetCELUserField</code>
"Asterisk-Peer"	Name of the Peer for 2-channel events (like bridge)

# Chapter 12

## Voicemail

### 12.1 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
<code>msgnum</code>	<code>int(11)</code>	Yes		NULL	
<code>dir</code>	<code>varchar(80)</code>	Yes	MUL	NULL	
<code>context</code>	<code>varchar(80)</code>	Yes		NULL	
<code>macrocontext</code>	<code>varchar(80)</code>	Yes		NULL	
<code>callerid</code>	<code>varchar(40)</code>	Yes		NULL	
<code>origtime</code>	<code>varchar(40)</code>	Yes		NULL	
<code>duration</code>	<code>varchar(20)</code>	Yes		NULL	
<code>flag</code>	<code>varchar(8)</code>	Yes		NULL	
<code>mailboxuser</code>	<code>varchar(80)</code>	Yes		NULL	
<code>mailboxcontext</code>	<code>varchar(80)</code>	Yes		NULL	
<code>recording</code>	<code>longblob</code>	Yes		NULL	

Table 12.1: `voicemessages` Table Schema

The database name (from `/etc/asterisk/res_odbc.conf`) is in the `odbcstorage` variable in the general section of `voicemail.conf`.

You may modify the `voicemessages` table name by using `odbctable=table_name` in `voicemail.conf`.

## 12.2 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.

### 12.2.1 Installation Notes

#### University of Washington IMAP C-Client

If you do not have the University of Washington's IMAP c-client installed on your system, you will need to download the c-client source distribution (<http://www.washington.edu/imap/>) and compile it. Asterisk supports the 2007 version of c-client as there appears to be issues with older versions which cause Asterisk to crash in certain scenarios. It is highly recommended that you utilize a current version of the c-client libraries. Additionally, `mail-expunge-full` is enabled in the 2006 and later versions.

Note that Asterisk only uses the 'c-client' portion of the UW IMAP toolkit, but building it also builds an IMAP server and various other utilities. Because of this, the build instructions for the IMAP toolkit are somewhat complicated and can lead to confusion about what is needed.

If you are going to be connecting Asterisk to an existing IMAP server, then you don't need to care about the server or utilities in the IMAP toolkit

at all. If you want to also install the UW IMAPD server, that is outside the scope of this document.

Building the c-client library is fairly straightforward; for example, on a Debian system there are two possibilities:

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"
```

Additionally, you may wish to build on a 64-bit machine, in which case you need to add `-fPIC` to `EXTRACFLAGS`. So, building on a 64-bit machine with SSL support would look something like:

```
$ make slx EXTRACFLAGS="-fPIC -I/usr/include/openssl"
```

Or without SSL support:

```
$ make slx SSLTYPE=none EXTRACFLAGS=-fPIC
```

Once this completes you can proceed with the Asterisk build; there is no need to run `'make install'`.

### Compiling Asterisk

Configure with `./configure --with-imap=/usr/src/imap` or wherever you built the UWashing IMAP Toolkit. This directory will be searched for a source installation. If no source installation is found there, then a package installation of the IMAP c-client will be searched for in this directory. If one is not found, then configure will fail.

A second configure option is to not specify a directory (i.e. `./configure --with-imap`). This will assume that you have the `imap-2007e` source installed in the `../imap` directory relative to the Asterisk source. If you do not have

this source, then configure will default to the "system" option defined in the next paragraph

A third option is `./configure --with-imap=system`. This will assume that you have installed a dynamically linked version of the c-client library (most likely via a package provided by your distro). This will attempt to link against `-lc-client` and will search for c-client headers in your include path starting with the `imap` directory, and upon failure, in the `c-client` directory.

When you run 'make menuselect', choose 'Voicemail Build Options' and the `IMAP_STORAGE` option should be available for selection.

After selecting the `IMAP_STORAGE` option, use the 'x' key to exit menuselect and save your changes, and the build/install Asterisk normally.

## 12.2.2 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>
imapfolder=<IMAP folder to store messages to>
imapgreetings=<yes or no>
greetingsfolder=<IMAP folder to store greetings in if imapgreetings is enabled>
expungeonhangup=<yes or no>
authuser=<username>
authpassword=<password>
opentimeout=<TCP open timeout in seconds>
closetimeout=<TCP close timeout in seconds>
readtimeout=<TCP read timeout in seconds>
writetimeout=<TCP write timeout in seconds>
```

The "imapfolder" can be used to specify an alternative folder on your IMAP server to store voicemails in. If not specified, the default folder 'IN-BOX' will be used.

The "imapgreetings" parameter can be enabled in order to store voicemail greetings on the IMAP server. If disabled, then they will be stored on the local file system as normal.

The "greetingsfolder" can be set to store greetings on the IMAP server when "imapgreetings" is enabled in an alternative folder than that set by "imapfolder" or the default folder for voicemails.

The "expungeonhangup" flag is used to determine if the voicemail system should expunge all messages marked for deletion when the user hangs up the phone.

Each mailbox definition should also have `imapuser=<imap username>`. For example:

```
4123=>4123,James Rothenberger,jar@onebiztone.com,,attach=yes|imapuser=jar
```

The directives "authuser" and "authpassword" are not needed when using Kerberos. They are defined to allow Asterisk to authenticate as a single user that has access to all mailboxes as an alternative to Kerberos.

### 12.2.3 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.

Please note that it is not recommended to store your voicemails in the top level folder where your users will keep their emails, especially if there are a large number of emails. A large number of emails in the same folder(s) that you're storing your voicemails could cause a large delay as Asterisk must parse through all the emails. For example a mailbox with 100 emails in it could take up to 60 seconds to receive a response.

### 12.2.4 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.

### 12.2.5 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).

This version is *\*not\** recommended for Asterisk.

### **UW IMAP-2006**

This version supports UIDPLUS, which allows UID\_EXPUNGE capabilities. This feature allow the system to expunge ONLY pertinent messages, instead of the default behavior, which is to expunge ALL messages marked for deletion when EXPUNGE is called. The IMAP storage mechanism is this version of Asterisk will check if the UID\_EXPUNGE feature is supported by the server, and use it if possible.

This version is *\*not\** recommended for Asterisk.

### **UW IMAP-2007**

This is the currently recommended version for use with Asterisk.

### **Cyrus IMAP**

Cyrus IMAP server v2.3.3 has been tested using a hierarchy delimiter of '/'.

## **12.2.6 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.

## **12.2.7 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 13

## SMS

### 13.1 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](http://www.etsi.org). Landline SMS is starting to be available in various parts of Europe, and is available from BT in the UK. However, Asterisk would allow gateways to be created in other locations such as the US, and use of SMS capable phones such as the Magic Messenger. SMS works using analogue or ISDN lines.

### 13.2 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.

### 13.3 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).

### 13.4 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

## 13.5 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.

## 13.6 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("someapptohandleincoming sms ${EXTEN}")
exten = _X.,3,Hangup
; Mobile originated, RX. This is receiving a message from a device, e.g.
; a Magic Messenger on a sip extension
; Running an app after receipt of the text allows the app to find all messages
; in the queue and handle them, e.g. sending them to the public SMSC
; The app may be something like smsq --process=somecommand --queue=${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

```

smsmtx 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./08005875290,1,Goto(smsmtx,${EXTEN},1)
exten = _X./_80058752[0-8]0,1,Goto(smsmtx,${EXTEN}-${CALLERID(num):8:1},1)

```

Alternatively, if you have the correct national prefix on incoming CLI, e.g. using dahdi\_hfc, you might use:

```

exten = _X./08005875290,1,Goto(smsmtx,${EXTEN},1)
exten = _X./_080058752[0-8]0,1,Goto(smsmtx,${EXTEN}-${CALLERID(num):9:1},1)

```

smsmorx is normally accessed by a call from a local sip device connected to a Magic Messenger. It could however be that you are operating Asterisk as a message centre for calls from outside. Either way, you look at the called number and goto smsmorx. In the UK, the SMSC number that would be dialed is 1709400X where X is the caller sub address. As such typical usage in extension.config at the point of handling a call from a sip phone is:

```

exten = 17094009,1,Goto(smsmorx,${CALLERID(num)},1)
exten = _1709400[0-8],1,Goto(smsmorx,${CALLERID(num)}--{EXTEN:7:1},1)

```

## 13.7 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 mtttx calls  
Default is Local/ and the queue name without -X suffix  
--mttttx-callerid  
Specify the callerid for mtttx 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, `smq` checks for pending messages and generates an outgoing anyway

Note that when `smq` 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 `smq` 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

## 13.8 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.

## 13.9 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 14

## Queues

### 14.1 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.

### 14.2 Configuring Call Queues

#### 14.2.1 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 |
| [dispatch] |
| music=default |
| context=dispatch |
| strategy=ringall |
| joinempty=strict |
| leavewhenempty=strict |
|-----|

```

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

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

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

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

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

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

## 14.2.2 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/IfYouKnowYourPartysExtensionYouMayDialItAtAnyTime);
    WaitExten(0.1);
    Background(digium/Otherwise);
    WaitExten(0.1);
    Background(digium/ForSalesPleasePress2);
    WaitExten(0.2);
    Background(digium/ForCustomerServicePleasePress3);
    WaitExten(0.2);
    Background(digium/ForAllOtherDepartmentsPleasePress4);
    WaitExten(0.2);
    Background(digium/ToSpeakWithAnOperatorPleasePress0AtAnyTime);
    if( ${attempts} < 2 ) {
        WaitExten(0.3);
        Background(digium/ToHearTheseOptionsRepeatedPleaseHold);
    }
    WaitExten(5);
    if( ${attempts} < 2 ) goto repeat;
    Background(digium/YouHaveMadeNoSelection);
    Background(digium/ThisCallWillBeEnded);
    Background(goodbye);
    Hangup();
}
}

```

## The Contexts referenced from the queues.conf file

```

context sales {

    0 => goto dispatch,s,1;
    8 => Voicemail(${SALESVM});

    s => {
        Ringing();
        Wait(2);
    }
}

```

```

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

```

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

```

context customerservice {
    0 => {
        SetCIDName(CSVTrans);
        goto dispatch|s|1;
    }
    8 => Voicemail(${CUSTSERVVM});

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

```

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

```

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

```

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

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

First, you set `Queue_MAX_PENALTY` to a value, then you call `Queue(<queue-name>,option,...)` (see the Queue application documentation for details)

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

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

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

### 14.2.3 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,0${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-addrmove(dispatch,10,${EXTEN});
        &queue-success(${EXTEN});
    }

    // Brittanica Spears
    _[IO]6165 => {
        &queue-addrmove(dispatch,20,${EXTEN});
        &queue-success(${EXTEN});
    }
}
```

```

// Rock Hudson
_[IO]6170 => {
    &queue-addremove(sales-general,10,${EXTEN});
    &queue-addremove(customerservice,20,${EXTEN});
    &queue-addremove(dispatch,30,${EXTEN});
    &queue-success(${EXTEN});
}

// Saline Dye-on
_[IO]6070 => {
    &queue-addremove(sales-general,20,${EXTEN});
    &queue-addremove(customerservice,30,${EXTEN});
    &queue-addremove(dispatch,30,${EXTEN});
    &queue-success(${EXTEN});
}
}

```

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

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

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

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

```

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

```

The queue-addremove macro is defined in this manner:

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

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

## 14.2.4 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-ecstasy);

Set(CALLERID(name)=EmptySupQ);
Set(Queue_MAX_PENALTY=10);
Queue(support-dispatch,t);
Set(Queue_MAX_PENALTY=20);
Queue(support-dispatch,t);
Set(Queue_MAX_PENALTY=0); // means no max
Queue(support-dispatch,t);
goto dispatch,s,1;
}
6121 => &callagent(${RAQUEL},${EXTEN});
6165 => &callagent(${SPEARS},${EXTEN});
6170 => &callagent(${ROCK},${EXTEN});
6070 => &callagent(${SALINE},${EXTEN});
}

```

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

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

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

```

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

```

```
}  
}
```

In the callagent macro above, the  $\{\text{exten}\}$  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  $\{\text{GROUP\_COUNT}()\}$  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.

## 14.2.5 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)
```

## 14.2.6 Caveats

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

## 14.3 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|ringtime)**

The caller was connected to an agent. Hold time represents the amount of time the caller was on hold. The bridged channel unique ID contains the unique ID of the queue member channel that is taking the call. This is useful when trying to link recording filenames to a particular call in the queue. Ringtime represents the time the queue members phone was ringing prior to being answered.

**ENTERQUEUE(url|callerid)**

A call has entered the queue. URL (if specified) and Caller\*ID are placed in the log.

**EXITEMPTY(position|origposition|waittime)**

The caller was exited from the queue forcefully because the queue had no reachable members and it's configured to do that to callers when there are no reachable members. The position is the caller's position in the queue when they hungup, the origposition is the original position the caller was when they first entered the queue, and the waittime is how long the call had been waiting in the queue at the time of disconnect.

**EXITWITHKEY(key|position|origposition|waittime)**

The caller elected to use a menu key to exit the queue. The key and the caller's position in the queue are recorded. The caller's entry position and amount of time waited is also recorded.

**EXITWITHTIMEOUT(position|origposition|waittime)**

The caller was on hold too long and the timeout expired. The position in the queue when the timeout occurred, the entry position, and the amount of time waited are logged.

**QUEUESTART**

The queueing system has been started for the first time this session.

**RINGNOANSWER(ringtime)**

After trying for ringtime ms to connect to the available queue member, the attempt ended without the member picking up the call. Bad queue member!

**SYSCOMPAT**

A call was answered by an agent, but the call was dropped because the channels were not compatible.

**TRANSFER(extension|context|holdtime|calltime|origposition)**

Caller was transferred to a different extension. Context and extension are recorded. The caller's hold time and the length of the call are both recorded, as is the caller's entry position at the time of the transfer. PLEASE remember

that transfers performed by SIP UA's by way of a reinvite may not always be caught by Asterisk and trigger off this event. The only way to be 100% sure that you will get this event when a transfer is performed by a queue member is to use the built-in transfer functionality of Asterisk.

# Chapter 15

## Phone Provisioning

### 15.1 Introduction

Asterisk includes basic phone provisioning support through the `res_phoneprov` module. The current implementation is based on a templating system using Asterisk dialplan function and variable substitution and obtains information to substitute into those templates from `phoneprov.conf` and `users.conf`. A profile and set of templates is provided for provisioning Polycom phones. Note that `res_phoneprov` is currently limited to provisioning a single user per device.

### 15.2 Configuration of `phoneprov.conf`

The configuration file, `phoneprov.conf`, is used to set up the built-in variables `SERVER` and `SERVER_PORT`, to define a default phone profile to use, and to define different phone profiles available for provisioning.

#### 15.2.1 The `[general]` section

Below is a sample of the general section of `phoneprov.conf`:

```
[general]
;serveriface=eth0
;serveraddr=192.168.1.1
;serverport=5060
default_profile=polycom
```

By default, `res_phoneprov` will set the `SERVER` variable to the IP address on the server that the requesting phone uses to contact the asterisk HTTP server. The `SERVER_PORT` variable will default to the **`bindport`** setting in `sip.conf`.

Should the defaults be insufficient, there are two choices for overriding the default setting of the `SERVER` variable. If the IP address of the server is known, or the hostname resolvable by the phones, the appropriate **`server-addr`** value should be set. Alternatively, the network interface that the server listens on can be set by specifying a **`serveriface`** and `SERVER` will be set to the IP address of that interface. Only one of these options should be set.

The default `SERVER_PORT` variable can be overridden by setting the **`serverport`**. If **`bindport`** is not set in `sip.conf` and `serverport` is not specified, it is set to a default value of 5060.

Any user set for auto-provisioning in `users.conf` without a specified profile will be assumed to belong to the profile set with **`default_profile`**.

## 15.2.2 Creating phone profiles

A phone profile is basically a list of files that a particular group of phones needs to function. For most phone types there are files that are identical for all phones (firmware, for instance) as well as a configuration file that is specific to individual phones. `res_phoneprov` breaks these two groups of files into static files and dynamic files, respectively. A sample profile:

```
[polycom]
staticdir => configs/
mime_type => text/xml
setvar => CUSTOM_CONFIG=/var/lib/asterisk/phoneprov/configs/custom.cfg
static_file => bootrom.ld,application/octet-stream
static_file => bootrom.ver,plain/text
static_file => sip.ld,application/octet-stream
static_file => sip.ver,plain/text
static_file => sip.cfg
static_file => custom.cfg
${TOLOWER(${MAC})}.cfg => 000000000000.cfg
${TOLOWER(${MAC})}-phone.cfg => 000000000000-phone.cfg
config/${TOLOWER(${MAC})} => polycom.xml
${TOLOWER(${MAC})}-directory.xml => 000000000000-directory.xml
```

A **`static_file`** is set by specifying the file name, relative to `AST\_DATA\_DIR/phoneprov`. The mime-type of the file can optionally be specified after a comma. If **`staticdir`** is set, all static files will be relative to the subdirectory of `AST_DATA_DIR/phoneprov` specified.

Since phone-specific config files generally have file names based on phone-specific data, dynamic filenames in `res_phoneprov` can be defined with Asterisk dialplan function and variable substitution. In the above example, `_${TOLOWER}_${MAC}.cfg => 000000000000.cfg` would define a relative URI to be served that matches the format of `MACADDRESS.cfg`, all lower case. A request for that file would then point to the template found at `AST_DATA_DIR/phoneprov/000000000000.cfg`. The template can be followed by a comma and mime-type. Notice that the dynamic filename (URI) can contain directories. Since these files are dynamically generated, the config file itself does not reside on the filesystem—only the template. To view the generated config file, open it in a web browser. If the config file is XML, Firefox should display it. Some browsers will require viewing the source of the page requested.

A default mime-type for the profile can be defined by setting **mime-type**. If a custom variable is required for a template, it can be specified with **setvar**. Variable substitution on this value is done while building the route list, so `_${USERNAME}` would expand to the username of the `users.conf` user that registers the dynamic filename.

NOTE: Any dialplan function that is used for generation of dynamic file names MUST be loaded before `res_phoneprov`. Add `"preload => module-name.so"` to `modules.conf` for required functions. In the example above, `"preload => func_strings.so"` would be required.

## 15.3 Configuration of `users.conf`

The `asterisk-gui` sets up extensions, SIP/IAX2 peers, and a host of other settings. User-specific settings are stored in `users.conf`. If the `asterisk-gui` is not being used, manual entries to `users.conf` can be made.

### 15.3.1 The [general] section

There are only two settings in the general section of `users.conf` that apply to phone provisioning: `localextenlength` which maps to template variable `EXTENSION_LENGTH` and `vmexten` which maps to the `VOICEMAIL_EXTEN` variable.

## 15.3.2 Individual Users

To enable auto-provisioning of a phone, the user in `users.conf` needs to have:

```
...
autoprov=yes
macaddress=deadbeef4dad
profile=polycom
```

The profile is optional if a `default_profile` is set in `phoneprov.conf`. The following is a sample `users.conf` entry, with the template variables commented next to the settings:

```
[6001]
callwaiting = yes
context = numberplan-custom-1
hasagent = no
hasdirectory = yes
hasiax = no
hasmanager = no
hassip = yes
hasvoicemail = yes
host = dynamic
mailbox = 6001
threewaycalling = yes
deletevoicemail = no
autoprov = yes
profile = polycom
directmedia = no
nat = no
fullname = User Two ; ${DISPLAY_NAME}
secret = test ; ${SECRET}
username = 6001 ; ${USERNAME}
macaddress = deadbeef4dad ; ${MAC}
label = 6001 ; ${LABEL}
cid_number = 6001 ; ${CALLERID}
```

The variables above, are the user-specific variables that can be substituted into dynamic filenames and config templates.

## 15.4 Templates

Configuration templates are a generic way to configure phones with text-based configuration files. Templates can use any loaded dialplan function and all of the variables created by `phoneprov.conf` and `users.conf`. A short example is the included `000000000000.cfg` Polycom template:

```

<?xml version="1.0" standalone="yes"?>
<APPLICATION
  APP_FILE_PATH="sip.ld"
  CONFIG_FILES="${IF(${STAT(e|${CUSTOM_CONFIG})} ? "custom.cfg,
")}config/${TOLOWER(${MAC})}, sip.cfg"
  MISC_FILES="" LOG_FILE_DIRECTORY=""
/>

```

This template uses dialplan functions, expressions, and a couple of variables to generate a config file to instruct the Polycom where to pull other needed config files. If a phone with MAC address 0xDEADBEEF4DAD requests this config file, and the filename that is stored in variable CUSTOM\_CONFIG does not exist, then the generated output would be:

```

<?xml version="1.0" standalone="yes"?>
<APPLICATION
  APP_FILE_PATH="sip.ld"
  CONFIG_FILES="config/deadbeef4dad, sip.cfg"
  MISC_FILES="" LOG_FILE_DIRECTORY=""
/>

```

The Polycom phone would then download both sip.cfg (which would be registered in `phoneprov.conf` as a static file) and `config/deadbeef4dad` (which would be registered as a dynamic file pointing to another template, `polycom.xml`).

`res_phoneprov` also registers its own dialplan function: `PP_EACH_USER`. This function was designed to be able to print out a particular string for each user that `res_phoneprov` knows about. An example use of this function is the template for a Polycom contact directory:

```

<?xml version="1.0" standalone="yes"?>
<directory>
  <item_list>
    ${PP_EACH_USER(<item><fn>${DISPLAY_NAME}</fn><ct>${CALLERID}</ct><bw>1</bw></item>|${MAC})}
  </item_list>
</directory>

```

`PP_EACH_USER` takes two arguments. The first is the string to be printed for each user. Any variables that are to be substituted need to be in the format `%{VARIABLE}` so that Asterisk doesn't try to substitute the variable immediately before it is passed to `PP_EACH_USER`. The second, optional, argument is a MAC address to exclude from the list iterated over (so, in this case, a phone won't be listed in its own contact directory).

## 15.5 Putting it all together

Make sure that `manager.conf` has:

```
[general]
enabled = yes
webenabled = yes
```

and that `http.conf` has:

```
[general]
enabled = yes
bindaddr = 192.168.1.1 ; Your IP here ;-)
bindport = 8088 ; Or port 80 if it is the only http server running on the machine
```

With `phoneprov.conf` and `users.conf` in place, start Asterisk. From the CLI, type "http show status". An example output:

```
HTTP Server Status:
Prefix: /asterisk
Server Enabled and Bound to 192.168.1.1:8088

Enabled URI's:
/asterisk/httpstatus => Asterisk HTTP General Status
/asterisk/phoneprov/... => Asterisk HTTP Phone Provisioning Tool
/asterisk/manmanager => HTML Manager Event Interface
/asterisk/rawman => Raw HTTP Manager Event Interface
/asterisk/static/... => Asterisk HTTP Static Delivery
/asterisk/mxml => XML Manager Event Interface

Enabled Redirects:
None.

POST mappings:
None.
```

There should be a `phoneprov` URI listed. Next, from the CLI, type "phoneprov show routes" and verify that the information there is correct. An example output for Polycom phones would look like:

```
Static routes

Relative URI          Physical location
sip.ver               configs/sip.ver
sip.ld                configs/sip.ld
bootrom.ver           configs/bootrom.ver
sip.cfg               configs/sip.cfg
bootrom.ld            configs/bootrom.ld
custom.cfg             configs/custom.cfg

Dynamic routes
```

Relative URI	Template
deadbeef4dad.cfg	000000000000.cfg
deadbeef4dad-directory.xml	000000000000-directory.xml
deadbeef4dad-phone.cfg	000000000000-phone.cfg
config/deadbeef4dad	polycom.xml

With the above examples, the phones would be pointed to `http://192.168.1.1:8080/asterisk/phoneprov` for pulling config files. Templates would all be placed in `AST_DATA_DIR/phoneprov` and static files would be placed in `AST_DATA_DIR/phoneprov/configs`. Examples of valid URIs would be:

- `http://192.168.1.1:8080/asterisk/phoneprov/sip.cfg`
- `http://192.168.1.1:8080/asterisk/phoneprov/deadbeef4dad.cfg`
- `http://192.168.1.1:8080/asterisk/phoneprov/config/deadbeef4dad`

# Chapter 16

## Calendaring

### 16.1 Introduction

The Asterisk Calendaring API aims to be a generic interface for integrating Asterisk with various calendaring technologies. The goal is to be able to support reading and writing of calendar events as well as allowing notification of pending events through the Asterisk dialplan.

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

### 16.2 Configuring Asterisk Calendaring

All asterisk calendaring modules are configured through `calender.conf`. Each calendar module can define its own set of required parameters in addition to the parameters available to all calendar types. An effort has been made to keep all options the same in all calendaring modules, but some options will diverge over time as features are added to each module.

An example `calendar.conf` might look like:

```
[calendar_joe]
type = ical
url = https://example.com/home/jdoe/Calendar
user = jdoe
secret = mysecret
refresh = 15
```

```
timeframe = 600
autoremind = 10
channel = SIP/joe
context = calendar_event_notify
extension = s
waittime = 30
```

## 16.2.1 Module-independent settings

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

**autoremind** This allows the overriding of any alarms that may or may not be set for a calendar event. It is specified in minutes.

**refresh** How often to refresh the calendar data; specified in minutes.

**timeframe** How far into the future each calendar refresh should look. This is the amount of data that will be visible to queries from the dialplan. This setting should always be greater than or equal to the refresh setting or events may be missed. It is specified in minutes.

**channel** The channel that should be used for making the notification attempt.

**waittime** How long to wait, in seconds, for the channel to answer a notification attempt.

There are two ways to specify how to handle a notification. One option is providing a context and extension, while the other is providing an application and the arguments to that application. One (and only one) of these options should be provided.

**context** The context of the extension to connect to the notification channel

**extension** The extension to connect to the notification. Note that the priority will always be 1.

or

**app** The dialplan application to execute upon the answer of a notification

**appdata** The data to pass to the notification dialplan application

## 16.2.2 Module-dependent settings

Connection-related options are specific to each module. Currently, all modules take a url, user, and secret for configuration and no other module-specific settings have been implemented. At this time, no support for HTTP redirects has been implemented, so it is important to specify the correct URL—paying attention to any trailing slashes that may be necessary.

## 16.3 Dialplan functions

### 16.3.1 Read functions

The simplest dialplan query is the `CALENDAR_BUSY` query. It takes a single option, the name of the calendar defined, and returns "1" for busy (including tentatively busy) and "0" for not busy.

For more information about a calendar event, a combination of `CALENDAR_QUERY` and `CALENDAR_QUERY_RESULT` is used. `CALENDAR_QUERY` takes the calendar name and optionally a start and end time in "unix time" (seconds from unix epoch). It returns an id that can be passed to `CALENDAR_QUERY_RESULT` along with a field name to return the data in that field. If multiple events are returned in the query, the number of the event in the list can be specified as well. The available fields to return are:

**summary** A short summary of the event

**description** The full description of the event

**organizer** Who organized the event

**location** Where the event is located

**calendar** The name of the calendar from `calendar.conf`

**uid** The unique identifier associated with the event

**start** The start of the event in seconds since Unix epoch

**end** The end of the event in seconds since Unix epoch

**busystate** The busy state 0=Free, 1=Tentative, 2=Busy

**attendees** A comma separated list of attendees as stored in the event and may include prefixes such as "mailto:".

When an event notification is sent to the dial plan, the CALENDAR.-EVENT function may be used to return the information about the event that is causing the notification. The fields that can be returned are the same as those from CALENDAR\_QUERY\_RESULT.

## 16.3.2 Write functions

To write an event to a calendar, the CALENDAR\_WRITE function is used. This function takes a calendar name and also uses the same fields as CALENDAR\_QUERY\_RESULT. As a write function, it takes a set of comma-separated values that are in the same order as the specified fields. For example:

```
CALENDAR_WRITE(mycalendar,summary,organizer,start,end,busystate)=
  "My event","mailto:jdoe@example.com",228383580,228383640,1)
```

## 16.4 Dialplan Examples

### 16.4.1 Office hours

A common business PBX scenario is would be executing dialplan logic based on when the business is open and the phones staffed. If the business is closed for holidays, it is sometimes desirable to play a message to the caller stating why the business is closed.

The standard way to do this in asterisk has been doing a series of GotoIfTime statements or time-based include statements. Either way can be tedious and requires someone with access to edit asterisk config files.

With calendaring, the administrator only needs to set up a calendar that contains the various holidays or even recurring events specifying the office hours. A custom greeting filename could even be contained in the description field for playback. For example:

```
[incoming]
exten => 5555551212,1,Answer
exten => 5555551212,n,GotoIf(${CALENDAR_BUSY(officehours)}?closed:attendant,s,1)
exten => 5555551212,n(closed),Set(id=${CALENDAR_QUERY(office,${EPOCH},${EPOCH})})
exten => 5555551212,n,Set(soundfile=${CALENDAR_QUERY_RESULT(${id},description)})
exten => 5555551212,n,Playback(${${ISNULL(soundfile)} ? generic-closed :: ${soundfile}})
exten => 5555551212,n,Hangup
```

## 16.4.2 Meeting reminders

One useful application of Asterisk Calendaring is the ability to execute dialplan logic based on an event notification. Most calendaring technologies allow a user to set an alarm for an event. If these alarms are set on a calendar that Asterisk is monitoring and the calendar is set up for event notification via `calendar.conf`, then Asterisk will execute notify the specified channel at the time of the alarm. If an overridden notification time is set with the `autorereminder` setting, then the notification would happen at that time instead.

The following example demonstrates the set up for a simple event notification that plays back a generic message followed by the time of the upcoming meeting. `calendar.conf`.

```
[calendar_joe]
type = ical
url = https://example.com/home/jdoe/Calendar
user = jdoe
secret = mysecret
refresh = 15
timeframe = 600
autorereminder = 10
channel = SIP/joe
context = calendar_event_notify
extension = s
waittime = 30

[calendar_event_notify]
exten => s,1,Answer
exten => s,n,Playback(you-have-a-meeting-at)
exten => s,n,SayUnixTime(${CALENDAR_EVENT(start)})
exten => s,n,Hangup
```

## 16.4.3 Writing an event

Both CalDAV and Exchange calendar servers support creating new events. The following example demonstrates writing a log of a call to a calendar.

```
[incoming]
exten => 6000,1,Set(start=${EPOCH})
exten => 6000,n,Dial(SIP/joe)
exten => h,1,Set(end=${EPOCH})
exten => h,n,Set(CALENDAR_WRITE(calendar_joe,summary,start,end)=Call from ${CALLERID(all)},{start},{end})
```

# Chapter 17

## Security Framework

### 17.1 Introduction

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

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

## 17.2 Framework Overview

This section discusses the architecture of the Asterisk modifications being proposed.

There are two main components that we propose for the initial implementation of the security framework:

- Security Event Generation
- Security Event Logger

### 17.2.1 Security Event Generation

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

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

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

### 17.2.2 Security Event Logger

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

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

```
security_log => security
```

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

## 17.3 Events to Log

(-) required

(+) optional

### Invalid Account ID

- (-) Local address family/IP address/port/transport
- (-) Remote address family/IP address/port/transport
- (-) Service (SIP, AMI, IAX2, ...)
- (-) System Name
- (+) Module
- (+) Account ID (username, etc)
- (+) Session ID (CallID, etc)
- (+) Session timestamp (required if Session ID present)
- (-) Event timestamp (sub-second precision)

### Failed ACL match

- > everything from invalid account ID
- (+) Name of ACL (when we have named ACLs)

### Invalid Challenge/Response

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

### Invalid Password

- > everything from invalid account ID

### Successful Authentication

- > informational event
- > everything from invalid account ID

Invalid formatting of Request

- > everything from invalid account ID
- > account ID optional
- (-) Request Type
- (+) Request parameters

Session Limit Reached (such as a call limit)

- > everything from invalid account ID

Memory Limit Reached

- > everything from invalid account ID

Maximum Load Average Reached

- > everything from invalid account ID

Request Not Allowed

- > everything from invalid account ID
- (-) Request Type
- (+) Request parameters

Request Not Supported

- > everything from invalid account ID
- (-) Request Type

Authentication Method Not Allowed

- > everything from invalid account ID
- (-) Authentication Method attempted

In dialog message from unexpected host

- > everything from invalid account ID
- (-) expected host

## 17.4 Security Log File Format

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

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

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

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

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

IE: SecurityEvent

Content: This is the security event sub-type.

Values: FailedACL, InvalidAccountID, SessionLimit, MemoryLimit, LoadAverageLimit, RequestNotSupported, RequestNotAllowed, AuthMethodNotAllowed, ReqBadFormat, UnexpectedAddress, ChallengeResponseFailed, InvalidPassword

IE: EventVersion

Content: This is a numeric value that indicates when updates are made to the content of the event.

Values: Monotonically increasing integer, starting at 1

IE: Service

Content: This is the Asterisk service that generated the event.

Values: TEST, SIP, AMI

IE: Module

Content: This is the Asterisk module that generated the event.

Values: chan\_sip

IE: AccountID

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

IE: SessionID

Content: This is a string used to identify the session associated with the

event. The format of the session identifier is specific to the service. In the case of SIP, this would be the Call-ID.

IE: SessionTV

Content: The time that the session associated with the SessionID started.

Values: <seconds>-<microseconds> since epoch

IE: ACLName

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

IE: LocalAddress

Content: This is the local address that was contacted for the related event.

Values: <Address Family>/<Transport>/<Address>/<Port>

Examples:

-> IPV4/UDP/192.168.1.1/5060

-> IPV4/TCP/192.168.1.1/5038

IE: RemoteAddress

Content: This is the remote address associated with the event.

Examples:

-> IPV4/UDP/192.168.1.2/5060

-> IPV4/TCP/192.168.1.2/5038

IE: ExpectedAddress

Content: This is the address that was expected to be the remote address.

Examples:

-> IPV4/UDP/192.168.1.2/5060

-> IPV4/TCP/192.168.1.2/5038

IE: EventTV

Content: This is the timestamp of when the event occurred.

Values: <seconds>-<microseconds> since epoch

IE: RequestType

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

IE: RequestParams

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

IE: AuthMethod

Content: This is a service specific string that represents an authentication method that was used or requested.

IE: Challenge

Content: This is a service specific string that represents the challenge provided to a user attempting challenge/response authentication.

IE: Response

Content: This is a service specific string that represents the response received from a user attempting challenge/response authentication.

IE: ExpectedResponse

Content: This is a service specific string that represents the response that was expected to be received from a user attempting challenge/response authentication.

# Chapter 18

## Secure Calls

### 18.1 Introduction

Asterisk supports a channel-agnostic method for handling secure call requirements. Since there is no single meaning of what constitutes a "secure call," Asterisk allows the administrator the control to define "secure" for themselves via the dialplan and channel-specific configuration files.

### 18.2 Channel-specific configuration

Currently the IAX2 and SIP channels support the call security features in Asterisk. Both channel-specific configuration files (`iax2.conf` and `sip.conf`) support the `encryption=yes` setting. For IAX2, this setting causes Asterisk to offer encryption when placing or receiving a call. To force encryption with IAX2, the `forceencrypt=yes` option is required. Due to limitations of SDP, `encryption=yes` in `sip.conf` results in a call with only a secure media offer, therefore `forceencrypt=yes` would be redundant in `sip.conf`.

If a peer is defined as requiring encryption but the endpoint does not support it, the call will fail with a HANGUPCAUSE of 58 (bearer capability does not exist).

## 18.3 Security-based dialplan branching

Each channel that supports secure signaling or media can implement a CHANNEL read callback function that specifies whether or not that channel meets the specified criteria. Currently, `chan_iax2` and `chan_sip` implement these callbacks. Channels that do not support secure media or signaling will return an empty string when queried. For example, to only allow an inbound call that has both secure signaling and media, see the following example.

```
exten => 123,1,GotoIf("${CHANNEL(secure_signaling)}" = "")?fail)
exten => 123,n,GotoIf("${CHANNEL(secure_media)}" = "")?fail)
exten => 123,n,Dial(SIP/123)
exten => 123,n,Hangup
exten => 123,n(fail),Playback(vm-goodbye)
exten => 123,n,Hangup
```

## 18.4 Forcing bridged channels to be secure

Administrators can force outbound channels that are to be bridged to a calling channel to conform to secure media and signaling policies. For example, to first make a call attempt that has both secure signaling and media, but gracefully fall back to non-secure signaling and media see the following example:

```
exten => 123,1,NoOp(We got a call)
exten => 123,n,Set(CHANNEL(secure_bridge_signaling)=1)
exten => 123,n,Set(CHANNEL(secure_bridge_media)=1)
exten => 123,n,Dial(SIP/somebody)
exten => 123,n,NoOp(HANGUPCAUSE=${HANGUPCAUSE})
exten => 123,n,GotoIf("${HANGUPCAUSE}"="58"?encrypt_fail)
exten => 123,n,Hangup

; notify user that retrying via insecure channel (user-provided prompt)
exten => 123,n(encrypt_fail),Playback(secure-call-fail-retry)
exten => 123,n,Set(CHANNEL(secure_bridge_signaling)=0)
exten => 123,n,Set(CHANNEL(secure_bridge_media)=0)
exten => 123,n,Dial(SIP/somebody)
exten => 123,n,Hangup
```

# Chapter 19

## Manipulating Party ID Information

### 19.1 Introduction

This chapter aims to explain how to use some of the features available to manipulate party ID information. It will not delve into specific channel configuration options described in the respective sample configuration files. The party ID information can consist of Caller ID, Connected Line ID, redirecting to party ID information, and redirecting from party ID information. Meticulous control is needed particularly when interoperating between different channel technologies.

- Caller ID: The Caller ID information describes who is originating a call.
- Connected Line ID: The Connected Line ID information describes who is connected to the other end of a call while a call is established. Unlike Caller ID, the connected line information can change over the life of a call when call transfers are performed. The connected line information can also change in either direction because either end could transfer the call. For ISDN it is known as Connected Line Identification Presentation (COLP), Connected Line Identification Restriction (COLR), and Explicit Call Transfer (ECT). For SIP it is known either as P-Asserted-Identity or Remote-Party-Id.
- Redirecting information: When a call is forwarded, the call originator is informed that the call is redirecting-to a new destination. The new

destination is also informed that the incoming call is redirecting-from the forwarding party. A call can be forwarded repeatedly until a new destination answers it or a forwarding limit is reached.

## 19.2 Tools available

Asterisk contains several tools for manipulating the party ID information for a call. Additional information can be found by using the 'core show function' or 'core show application' console commands at the Asterisk CLI. The following list identifies some of the more common tools for manipulating the party ID information:

- CALLERID(datatype[,caller-id])
- CONNECTEDLINE(datatype[,i])
- REDIRECTING(datatype[,i])
- Dial() and Queue() dialplan application 'I' option
- Interception macros
- Channel driver specific configuration options.

### 19.2.1 CALLERID dialplan function

The CALLERID function has been around for quite a while and its use is straightforward. It is used to examine and alter the caller information that came into the dialplan with the call. Then the call with it's caller information passes on to the destination using the Dial() or Queue() application.

The CALLERID information is passed during the initial call setup. However, depending on the channel technology, the caller name may be delayed. Q.SIG is an example where the caller name may be delayed so your dialplan may need to wait for it.

### 19.2.2 CONNECTEDLINE dialplan function

The CONNECTEDLINE function does the opposite of the CALLERID function. CONNECTEDLINE can be used to setup connected line information to

be sent when the call is answered. You can use it to send new connected line information to the remote party on the channel when a call is transferred. The CONNECTEDLINE information is passed when the call is answered and when the call is transferred.

Since the connected line information can be sent while a call is connected, you may need to prevent the channel driver from acting on a partial update. The 'i' option is used to inhibit the channel driver from sending the changed information immediately.

### 19.2.3 REDIRECTING dialplan function

The REDIRECTING function allows you to report information about forwarded/deflected calls to the caller and to the new destination. The use of the REDIRECTING function is the most complicated of the party information functions.

The REDIRECTING information is passed during the initial call setup and while the call is being routed through the network. Since the redirecting information is sent before a call is answered, you need to prevent the channel driver from acting on a partial update. The 'i' option is used to inhibit the channel driver from sending the changed information immediately.

The incoming call may have already been redirected. An incoming call has already been redirected if the REDIRECTING(count) is not zero. (Alternate indications are if the REDIRECTING(from-num-valid) is non-zero or if the REDIRECTING(from-num) is not empty.)

There are several things to do when a call is forwarded by the dialplan:

- Setup the REDIRECTING(to-xxx) values to be sent to the caller.
- Setup the REDIRECTING(from-xxx) values to be sent to the new destination.
- Increment the REDIRECTING(count).
- Set the REDIRECTING(reason).
- Dial() the new destination.

## 19.2.4 Special REDIRECTING considerations for ISDN

Special considerations for Q.SIG and ISDN point-to-point links are needed to make the DivertingLegInformation1, DivertingLegInformation2, and DivertingLegInformation3 messages operate properly.

You should manually send the COLR of the redirected-to party for an incoming redirected call if the incoming call could experience further redirects. For `chan_misdn`, just set the `REDIRECTING(to-num,i) = ${EXTEN}` and set the `REDIRECTING(to-num-pres)` to the COLR. For `chan_dahdi`, just set the `REDIRECTING(to-num,i) = CALLERID(dnid)` and set the `REDIRECTING(to-num-pres)` to the COLR. (Setting the `REDIRECTING(to-num,i)` value may not be necessary since the channel driver has already attempted to preset that value for automatic generation of the needed DivertingLegInformation3 message.)

For redirected calls out a trunk line, you need to use the 'i' option on all of the REDIRECTING statements before dialing the redirected-to party. The call will update the redirecting-to presentation (COLR) when it becomes available.

## 19.2.5 Dial() and Queue() dialplan application 'I' option

In the dialplan applications `Dial()` and `Queue()`, the 'I' option is a brute force option to block connected line and redirecting information updates while the application is running. Blocking the updates prevents the update from overwriting any `CONNECTEDLINE` or `REDIRECTING` values you may have setup before running the application.

The option blocks all redirecting updates since they should only happen before a call is answered. The option only blocks the connected line update from the initial answer. Connected line updates resulting from call transfers happen after the application has completed. Better control of connected line and redirecting information is obtained using the interception macros.

## 19.2.6 Interception macros

The interception macros give the administrator an opportunity to alter connected line and redirecting information before the channel driver is given the information. If the macro does not change a value then that is what is going to be passed to the channel driver.

The tag string available in CALLERID, CONNECTEDLINE, and REDIRECTING is useful for the interception macros to provide some information about where the information originally came from.

The 'i' option of the CONNECTEDLINE dialplan function should always be used in the CONNECTED\_LINE interception macros. The interception macro always passes the connected line information on to the channel driver when the macro exits. Similarly, the 'i' option of the REDIRECTING dialplan function should always be used in the REDIRECTING interception macros.

`#{REDIRECTING_CALLEE_SEND_MACRO}`

Macro to call before sending a redirecting update to the callee. This macro may never be needed since the redirecting updates should only go from the callee to the caller direction. It is available for completeness.

`#{REDIRECTING_CALLEE_SEND_MACRO_ARGS}`

Arguments to pass to `#{REDIRECTING_CALLEE_SEND_MACRO}`.

`#{REDIRECTING_CALLER_SEND_MACRO}`

Macro to call before sending a redirecting update to the caller.

`#{REDIRECTING_CALLER_SEND_MACRO_ARGS}`

Arguments to pass to `#{REDIRECTING_CALLER_SEND_MACRO}`.

`#{CONNECTED_LINE_CALLEE_SEND_MACRO}`

Macro to call before sending a connected line update to the callee.

`#{CONNECTED_LINE_CALLEE_SEND_MACRO_ARGS}`

Arguments to pass to `#{CONNECTED_LINE_CALLEE_SEND_MACRO}`.

`#{CONNECTED_LINE_CALLER_SEND_MACRO}`

Macro to call before sending a connected line update to the caller.

`#{CONNECTED_LINE_CALLER_SEND_MACRO_ARGS}`

Arguments to pass to `#{CONNECTED_LINE_CALLER_SEND_MACRO}`.

## 19.3 Manipulation examples

The following examples show several common scenarios in which you may need to manipulate party ID information from the dialplan.

### 19.3.1 Simple recording playback

```
exten => 1000,1,NoOp
; The CONNECTEDLINE information is sent when the call is answered.
exten => 1000,n,Set(CONNECTEDLINE(name,i)="Company Name")
exten => 1000,n,Set(CONNECTEDLINE(name-pres,i)=allowed)
exten => 1000,n,Set(CONNECTEDLINE(num,i)=5551212)
exten => 1000,n,Set(CONNECTEDLINE(num-pres)=allowed)
exten => 1000,n,Answer
exten => 1000,n,Playback(tt-weasels)
exten => 1000,n,Hangup
```

### 19.3.2 Straightforward dial through

```
exten => 1000,1,NoOp
; The CONNECTEDLINE information is sent when the call is answered.
exten => 1000,n,Set(CONNECTEDLINE(name,i)="Company Name")
exten => 1000,n,Set(CONNECTEDLINE(name-pres,i)=allowed)
exten => 1000,n,Set(CONNECTEDLINE(num,i)=5551212)
exten => 1000,n,Set(CONNECTEDLINE(num-pres)=allowed)
; The I option prevents overwriting the CONNECTEDLINE information
; set above when the call is answered.
exten => 1000,n,Dial(SIP/1000,20,I)
exten => 1000,n,Hangup
```

### 19.3.3 Use of interception macro

```
[macro-add_pfx]
; ARG1 is the prefix to add.
; ARG2 is the number of digits at the end to add the prefix to.
; When the macro ends the CONNECTEDLINE data is passed to the
; channel driver.
exten => s,1,NoOp(Add prefix to connected line)
exten => s,n,Set(NOPREFIX=${CONNECTEDLINE(number):-${ARG2}})
exten => s,n,Set(CONNECTEDLINE(num,i)=${ARG1}${NOPREFIX})
exten => s,n,MacroExit

exten => 1000,1,NoOp
```

```

exten => 1000,n,Set(__CONNECTED_LINE_CALLER_SEND_MACRO=add_pfx)
exten => 1000,n,Set(__CONNECTED_LINE_CALLER_SEND_MACRO_ARGS=45,4)
exten => 1000,n,Dial(SIP/1000,20)
exten => 1000,n,Hangup

```

### 19.3.4 Simple redirection

```

exten => 1000,1,NoOp
; For Q.SIG or ISDN point-to-point we should determine the COLR for this
; extension and send it if the call was redirected here.
exten => 1000,n,GotoIf($[${REDIRECTING(count)}>0]?redirected:notredirected)
exten => 1000,n(redirected),Set(REDIRECTING(to-num,i)=${CALLERID(dnid)})
exten => 1000,n,Set(REDIRECTING(to-num-pres)=allowed)
exten => 1000,n(notredirected),NoOp
; Determine that the destination has forwarded the call.
; ...
exten => 1000,n,Set(REDIRECTING(from-num,i)=1000)
exten => 1000,n,Set(REDIRECTING(from-num-pres,i)=allowed)
exten => 1000,n,Set(REDIRECTING(to-num,i)=2000)
; The DivertingLegInformation3 message is needed because at this point
; we do not know the presentation (COLR) setting of the redirecting-to
; party.
exten => 1000,n,Set(REDIRECTING(count,i)=[${REDIRECTING(count)} + 1])
exten => 1000,n,Set(REDIRECTING(reason,i)=cfu)
; The call will update the redirecting-to presentation (COLR) when it
; becomes available with a redirecting update.
exten => 1000,n,Dial(DAHD1/g1/2000,20)
exten => 1000,n,Hangup

```

## 19.4 Ideas for usage

The following is a list of ideas in which the manipulation of party ID information would be beneficial.

- IVR that updates connected name on each selection made.
- Disguise the true number of an individual with a generic company number.

- Use interception macros to make outbound connected number E.164 formatted.
- You can do a lot more in an interception macro than just manipulate party information...

## 19.5 Troubleshooting tips

- For CONNECTEDLINE and REDIRECTING, check the usage of the 'i' option.
- Check channel configuration settings. The default settings may not be what you want or expect.
- Check packet captures. Your equipment may not support what Asterisk sends.

## 19.6 For further reading...

- Relevant ETSI ISDN redirecting specification: EN 300 207-1
- Relevant ETSI ISDN COLP specification: EN 300 097-1
- Relevant ETSI ISDN ECT specification: EN 300 369-1
- Relevant Q.SIG ISDN redirecting specification: ECMA-174
- Relevant Q.SIG ISDN COLP specification: ECMA-148
- Relevant Q.SIG ISDN ECT specification: ECMA-178
- Relevant SIP RFC for P-Asserted-Id: RFC3325
- The expired draft (draft-ietf-sip-privacy-04.txt) defines Remote-Party-Id. Since Remote-Party-Id has not made it into an RFC at this time, its use is non-standard by definition.

# Chapter 20

## Call Completion Supplementary Services

### 20.1 Introduction

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

### 20.2 What is CCSS?

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

## 20.3 Glossary of Terms

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

- **CCBS:** Call Completion on Busy Subscriber. When a call fails because the recipient's phone is busy, the caller will have the opportunity to request CCBS. When the recipient's phone is no longer busy, the caller will be alerted. The means by which the caller is alerted is dependent upon the type of agent used by the caller.
- **CCNR:** Call Completion on No Response. When a call fails because the recipient does not answer the phone, the caller will have the opportunity to request CCNR. When the recipient's phone becomes busy and then is no longer busy, the caller will be alerted. The means by which the caller is alerted is dependent upon the type of the agent used by the caller.
- **Agent:** The agent is the entity within Asterisk that communicates with and acts on behalf of the calling party.
- **Monitor:** The monitor is the entity within Asterisk that communicates with and monitors the status of the called party.
- **Generic Agent:** A generic agent is an agent that uses protocol-agnostic methods to communicate with the caller. Generic agents should only be used for phones, and never should be used for "trunks."
- **Generic Monitor:** A generic monitor is a monitor that uses protocol-agnostic methods to monitor the status of the called party. Like with generic agents, generic monitors should only be used for phones.
- **Native Agent:** The opposite of a generic agent. A native agent uses protocol-specific messages to communicate with the calling party. Native agents may be used for both phones and trunks, but it must be known ahead of time that the device with which Asterisk is communicating supports the necessary signaling.
- **Native Monitor:** The opposite of a generic monitor. A native monitor uses protocol-specific messages to subscribe to and receive notification

of the status of the called party. Native monitors may be used for both phones and trunks, but it must be known ahead of time that the device with which Asterisk is communicating supports the necessary signaling.

- Offer: An offer of CC refers to the notification received by the caller that he may request CC.
- Request: When the caller decides that he would like to subscribe to CC, he will make a request for CC. Furthermore, the term may refer to any outstanding requests made by callers.
- Recall: When the caller attempts to call the recipient after being alerted that the recipient is available, this action is referred to as a "recall."

## 20.4 The CC Process

### 20.4.1 The Initial Call

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

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

### 20.4.2 Requesting CC

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

With generic agents, the `CallCompletionRequest` application must be called in order to request CC. There are two different ways in which this may be called. It may either be called before the caller hangs up during the initial call, or the caller may hang up from the initial call and dial an

extension which calls the `CallCompletionRequest` application. If the second method is used, then the caller will have until the `cc_offer_timer` expires to request CC.

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

### **20.4.3 Cancelling CC**

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

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

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

### **20.4.4 Monitoring the Called Party**

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

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

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

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

## 20.4.5 Alerting the Caller

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

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

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

## 20.4.6 If the Caller is unavailable

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

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

## 20.4.7 The CC recall

The calling party will make its recall to the same extension that was dialed. Asterisk will provide a channel variable, `CC_INTERFACES`, to be used as an

argument to the Dial application for CC recalls. It is strongly recommended that you use this channel variable during a CC recall. Listed are two reasons:

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

Note that even if your call gets routed through local channels, the `CC_-INTERFACES` variable will be populated with the appropriate values for that specific extension. When the called parties are dialed, it is expected that a called party will answer, since Asterisk had previously determined that the party was available. However, it is possible that the called party may choose not to respond to the call, or he could have become busy again. In such a situation, the calling party must re-request CC if he wishes to still be alerted when the calling party has become available.

## 20.5 Miscellaneous Information and Tips

- Be aware when using a generic agent that the `max_cc_agents` configuration parameter is ignored. The main driving reason for this is that the mechanism for cancelling CC when using a generic agent would become much more potentially confusing to execute. By limiting a calling party to having a single request, there is only ever a single request to be cancelled, making the process simple.
- Keep in mind that no matter what CC agent type is being used, a CC request can only be made for the latest call issued.
- If available timers are running on multiple called parties, it is possible that one of the timers may expire before the others do. If such a situation occurs, then the interface on which the timer expired will cease to be monitored. If, though, one of the other called parties becomes available before his available timer expires, the called party whose available timer had previously expired will still be included in the `CC_INTERFACES` channel variable on the recall.

- It is strongly recommended that lots of thought is placed into the settings of the CC timers. Our general recommendation is that timers for phones should be set shorter than those for trunks. The reason for this is that it makes it less likely for a link in the middle of a network to cause CC to fail.
- CC can potentially be a memory hog if used irresponsibly. The following are recommendations to help curb the amount of resources required by the CC engine. First, limit the maximum number of CC requests in the system using the `cc_max_requests` option in `ccss.conf`. Second, set the `cc_offer_timer` low for your callers. Since it is likely that most calls will not result in a CC request, it is a good idea to set this value to something low so that information for calls does not stick around in memory for long. The final thing that can be done is to conditionally set the `cc_agent_policy` to "never" using the `CALLCOMPLETION` dialplan function. By doing this, no CC information will be kept around after the call completes.
- It is possible to request CCNR on answered calls. The reason for this is that it is impossible to know whether a call that is answered has actually been answered by a person or by something such as voicemail or some other IVR.
- Not all channel drivers have had the ability to set CC config parameters in their configuration files added yet. At the time of this writing (2009 Oct), only `chan_sip` has had this ability added, with short-term plans to add this to `chan_dahdi` as well. It is possible to set CC configuration parameters for other channel types, though. For these channel types, the setting of the parameters can only be accomplished using the `CALLCOMPLETION` dialplan function.
- It is documented in many places that generic agents and monitors can only be used for phones. In most cases, however, Asterisk has no way of distinguishing between a phone and a trunk itself. The result is that Asterisk will happily let you violate the advice given and allow you to set up a trunk with a generic monitor or agent. While this will not cause anything catastrophic to occur, the behavior will most definitely not be what you want.

- At the time of this writing (2009 Oct), Asterisk is the only known SIP stack to write an implementation of draft-ietf-bliss-call-completion-04. As a result, it is recommended that for your SIP phones, use a generic agent and monitor. For SIP trunks, you will only be able to use CC if the other end is terminated by another Asterisk server running version 1.8 or later.
- If the Dial application is called multiple times by a single extension, CC will only be offered to the caller for the parties called by the first instantiation of Dial.
- If a phone forwards a call, then CC may only be requested for the phone that executed the call forward. CC may not be requested for the phone to which the call was forwarded.
- CC is currently only supported by the Dial application. Queue, Followme, and Page do not support CC because it is not particularly useful for those applications.
- Generic CC relies heavily on accurate device state reporting. In particular, when using SIP phones it is vital to be sure that device state is updated properly when using them. In order to facilitate proper device state handling, be sure to set `callcounter=yes` for all peers and to set `limitonpeers=yes` in the general section of `sip.conf`
- When using SIP CC (i.e. native CC over SIP), it is important that your `minexpiry` and `maxexpiry` values allow for available timers to run as little or as long as they are configured. When an Asterisk server requests call completion over SIP, it sends a SUBSCRIBE message with an Expires header set to the number of seconds that the available timer should run. If the Asterisk server that receives this SUBSCRIBE has a `maxexpiry` set lower than what is in the received Expires header, then the available timer will only run for `maxexpiry` seconds.
- As with all Asterisk components, CC is not perfect. If you should find a bug or wish to enhance the feature, please open an issue on <https://issues.asterisk.org>. If writing an enhancement, please be sure to include a patch for the enhancement, or else the issue will be closed.

## 20.6 Simple Example of generic call completion

The following is an incredibly bare-bones example sip.conf and dialplan to show basic usage of generic call completion. It is likely that if you have a more complex setup, you will need to make use of items like the CALLCOMPLETION dialplan function or the CC\_INTERFACES channel variable.

First, let's establish a very simple sip.conf to use for this

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

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

Now, let's write a simple dialplan

```
[phone_calls]

exten => 1000,1,Dial(SIP/Mark,20)
exten => 1000,n,Hangup

exten => 2000,1,Dial(SIP/Richard,20)
exten => 2000,n,Hangup

exten => 30,1,CallCompletionRequest
exten => 30,n,Hangup

exten => 31,1,CallCompletionCancel
exten => 31,n,Hangup
```

- Scenario 1: Mark picks up his phone and dials Richard by dialing 2000. Richard is currently on a call, so Mark hears a busy signal. Mark then

hangs up, picks up the phone and dials 30 to call the CallCompletion-Request application. After some time, Richard finishes his call and hangs up. Mark is automatically called back by Asterisk. When Mark picks up his phone, Asterisk will dial extension 2000 for him.

- Scenario 2: Richard picks up his phone and dials Mark by dialing 1000. Mark has stepped away from his desk, and so he is unable to answer the phone within the 20 second dial timeout. Richard hangs up, picks the phone back up and then dials 30 to request call completion. Mark gets back to his desk and dials somebody's number. When Mark finishes the call, Asterisk detects that Mark's phone has had some activity and has become available again and rings Richard's phone. Once Richard picks up, Asterisk automatically dials extension 1000 for him.
- Scenario 3: Much like scenario 1, Mark calls Richard and Richard is busy. Mark hangs up, picks the phone back up and then dials 30 to request call completion. After a little while, Mark realizes he doesn't actually need to talk to Richard, so he dials 31 to cancel call completion. When Richard becomes free, Mark will not automatically be redialed by Asterisk.
- Scenario 4: Richard calls Mark, but Mark is busy. About thirty seconds later, Richard decides that he should perhaps request call completion. However, since Richard's phone has the default `cc_offer_timer` of 20 seconds, he has run out of time to request call completion. He instead must attempt to dial Mark again manually. If Mark is still busy, Richard can attempt to request call completion on this second call instead.
- Scenario 5: Mark calls Richard, and Richard is busy. Mark requests call completion. Richard does not finish his current call for another 2 hours (7200 seconds). Since Mark has the default `ccbs_available_timer` of 4800 seconds set, Mark will not be automatically recalled by Asterisk when Richard finishes his call.
- Scenario 6: Mark calls Richard, and Richard does not respond within the 20 second dial timeout. Mark requests call completion. Richard does not use his phone again for another 4 hours (144000 seconds). Since Mark has the default `ccnr_available_timer` of 7200 seconds set, Mark will not be automatically recalled by Asterisk when Richard finishes his call.

# Chapter 21

## Packet Loss Concealment

### 21.1 What is PLC?

PLC stands for Packet Loss Concealment. PLC describes any method of generating new audio data when packet loss is detected. In Asterisk, there are two main flavors of PLC, generic and native. Generic PLC is a method of generating audio data on signed linear audio streams. Signed linear audio, often abbreviated "slin," is required since it is a raw format that has no companding, compression, or other transformations applied. Native PLC is used by specific codec implementations, such as iLBC and Speex, which generates the new audio in the codec's native format. Native PLC happens automatically when using a codec that supports native PLC. Generic PLC requires specific configuration options to be used and will be the focus of this document.

### 21.2 How does Asterisk detect packet loss?

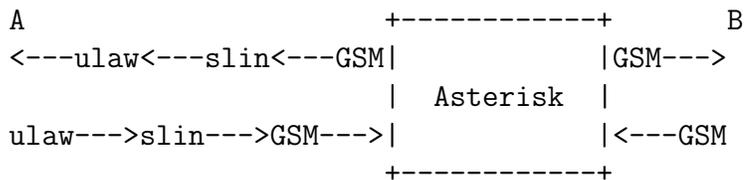
Oddly, Asterisk does not detect packet loss when reading audio in. In order to detect packet loss, one must have a jitter buffer in use on the channel on which Asterisk is going to write missing audio using PLC. When a jitter buffer is in use, audio that is to be written to the channel is fed into the jitterbuffer. When the time comes to write audio to the channel, a bridge will request that the jitter buffer gives a frame of audio to the bridge so that the audio may be written. If audio is requested from the jitter buffer but the

jitter buffer is unable to give enough audio to the bridge, then the jitter buffer will return an interpolation frame. This frame contains no actual audio data and indicates the number of samples of audio that should be inserted into the frame.

## 21.3 A bit of background on translation

As stated in the introduction, generic PLC can only be used on slin audio. The majority of audio communication is not done in slin, but rather using lower bandwidth codecs. This means that for PLC to be used, there must be a translation step involving slin on the write path of a channel. This means that PLC cannot be used if the codecs on either side of the bridge are the same or do not require a translation to slin in order to translate between them. For instance, a ulaw <-> ulaw call will not use PLC since no translation is required. In addition, a ulaw <-> alaw call will also not use PLC since the translation path does not include any step involving slin. One item of note is that slin must be present on the write path of a channel since that is the path where PLC is applied. Consider that Asterisk is bridging channels A and B. A uses ulaw for audio and B uses GSM. This translation involves slin, so things are shaping up well for PLC. Consider, however if Asterisk sets up the translation paths like so:

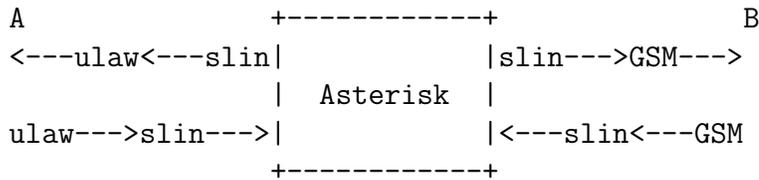
Fig. 1



The arrows indicate the direction of audio flow. Each channel has a write path (the top arrow) and a read path (the bottom arrow). In this setup, PLC can be used when sending audio to A, but it cannot be used when sending audio to B. The reason is simple, the write path to A's channel contains a slin step, but the write path to B contains no slin step. Such a translation setup is perfectly valid, and Asterisk can potentially set up such a path depending

on circumstances. When we use PLC, however, we want slin audio to be present on the write paths of both A and B. A visual representation of what we want is the following:

Fig. 2



In this scenario, the write paths for both A and B begin with slin, and so PLC may be applied to either channel. This translation behavior has, in the past been doable with the `transcode_via_sln` option in `asterisk.conf`. Recent changes to the PLC code have also made the `genericplc` option in `codecs.conf` imply the `transcode_via_sln` option. The result is that by enabling `genericplc` in `codecs.conf`, the translation path set up in Fig. 2 should automatically be used.

## 21.4 Additional restrictions and caveats

One restriction that has not been spelled out so far but that has been hinted at is the presence of a bridge. The term bridge in this sense means two channels exchanging audio with one another. A bridge is required because use of a jitter buffer is a prerequisite for using PLC, and a jitter buffer is only used when bridging two channels. This means that one-legged calls, (e.g. calls to voicemail, to an IVR, to an extension that just plays back audio) will not use PLC. In addition, MeetMe and ConfBridge calls will not use PLC. It should be obvious, but it bears mentioning, that PLC cannot be used when using a technology's native bridging functionality. For instance, if two SIP channels can exchange RTP directly, then Asterisk will never be able to process the audio in the first place. Since translation of audio is a requirement for using PLC, and translation will not allow for a native bridge to be created, this is something that is not likely to be an issue, though. Since a jitter buffer is a requirement in order to use PLC, it should be noted that

simply enabling the jitter buffer via the `jbenable` option may not be enough. For instance, if bridging two SIP channels together, the default behavior will not be to enable jitter buffers on either channel. The rationale is that the jitter will be handled at the endpoints to which Asterisk is sending the audio. In order to ensure that a jitter buffer is used in all cases, one must enable the `jbforce` option for channel types on which PLC is desired.

## 21.5 Summary

The following are all required for PLC to be used:

- Enable `genericplc` in the `plc` section of `codecs.conf`
- Enable (and potentially force) jitter buffers on channels
- Two channels must be bridged together for PLC to be used (no Meetme or one-legged calls)
- The audio must be translated between the two channels and must have `slin` as a step in the translation process.

## 21.6 Protip

One of the restrictions mentioned is that PLC will only be used when two audio channels are bridged together. Through the use of Local channels, you can create a bridge even if the call is, for all intents and purposes, one-legged. By using a combination of the `/n` and `/j` suffixes for a Local channel, one can ensure that the Local channel is not optimized out of the talk path and that a jitter buffer is applied to the Local channel as well. Consider the following simple dialplan:

```
[example]
exten => 1,1,Playback(tt-weasels)
exten => 2,1,Dial(Local/1@example/nj)
```

When dialing extension 1, PLC cannot be used because there will be only a single channel involved. When dialing extension 2, however, Asterisk will

create a bridge between the incoming channel and the Local channel, thus allowing PLC to be used.

# Chapter 22

## Sounds Packages

### 22.1 Introduction

Asterisk utilizes a variety of sound prompts that are available in several file formats and languages. Multiple languages and formats can be installed on the same system, and Asterisk will utilize prompts from languages installed, and will automatically pick the least CPU intensive format that is available on the system (based on codecs in use, in addition to the codec and format modules installed and available).

In addition to the prompts available with Asterisk, you can create your own sets of prompts and utilize them as well. This document will tell you how the prompts available for Asterisk are created so that the prompts you create can be as close and consistent in the quality and volume levels as those shipped with Asterisk.

### 22.2 Getting The Sounds Tools

The sounds tools are available in the publicly accessible repotools repository. You can check these tools out with Subversion via the following command:

```
# svn co http://svn.asterisk.org/svn/repotools
```

The sound tools are available in the subdirectory `sound_tools/` which contains the following directories:

- `audiofilter`

- makeg722
- scripts

## 22.3 About The Sounds Tools

The following sections will describe the sound tools in more detail and explain what they are used for in the sounds package creation process.

### 22.3.1 audiofilter

The audiofilter application is used to "tune" the sound files in such a way that they sound good when being used while in a compressed format. The values in the scripts for creating the sound files supplied in repotools is essentially a high-pass filter that drops out audio below 100Hz (or so).

(There is an ITU specification that states for 8KHz audio that is being compressed frequencies below a certain threshold should be removed because they make the resulting compressed audio sound worse than it should.)

The audiofilter application is used by the 'converter' script located in the scripts subdirectory of repotools/sound.tools. The values being passed to the audiofilter application is as follows:

```
audiofilter -n 0.86916 -1.73829 0.86916 -d 1.00000 -1.74152 0.77536
```

The two options -n and -d are 'numerator' and 'denominator'. Per the author, Jean-Marc Valin, "These values are filter coefficients (-n means numerator, -d is denominator) expressed in the z-transform domain. There represent an elliptic filter that I designed with Octave such that 'the result sounds good'."

### 22.3.2 makeg722

The makeg722 application is used by the 'converters' script to generate the G.722 sound files that are shipped with Asterisk. It starts with the RAW sound files and then converts them to G.722.

### 22.3.3 scripts

The scripts folder is where all the magic happens. These are the scripts that the Asterisk open source team use to build the packaged audio files for the various formats that are distributed with Asterisk.

- `chkcore` - used to check that the contents of `core-sounds-<lang>.txt` are in sync
- `chkextra` - same as above, but checks the extra sound files
- `mkcore` - script used to generate the core sounds packages
- `mkextra` - script used to generate the extra sounds packages
- `mkmoh` - script used to generate the music on hold packages
- `converters` - script used to convert the master files to various formats

# Chapter 23

## Development

### 23.1 Backtrace

This document is intended to provide information on how to obtain the backtraces required on the asterisk bug tracker, available at <https://issues.asterisk.org>. The information is required by developers to help fix problem with bugs of any kind. Backtraces provide information about what was wrong when a program crashed; in our case, Asterisk. There are two kind of backtraces (aka 'bt') which are useful: bt and bt full.

First of all, when you start Asterisk, you **MUST** start it with option -g. This tells Asterisk to produce a core file if it crashes.

If you start Asterisk with the safe\_asterisk script, it automatically starts using the option -g.

If you're not sure if Asterisk is running with the -g option, type the following command in your shell:

```
debian:/tmp# ps aux | grep asterisk
root      17832  0.0  1.2  2348  788 pts/1    S   Aug12   0:00 /bin/sh /usr/sbin/safe_asterisk
root      26686  0.0  2.8 15544 1744 pts/1    S   Aug13   0:02 asterisk -vvvg -c
[...]
```

The interesting information is located in the last column.

Second, your copy of Asterisk must have been built without optimization or the backtrace will be (nearly) unusable. This can be done by selecting the 'DONT\_OPTIMIZE' option in the Compiler Flags submenu in the 'make menuselect' tree before building Asterisk.

After Asterisk crashes, a core file will be "dumped" in your /tmp/ directory. To make sure it's really there, you can just type the following command in your shell:

```
debian:/tmp# ls -l /tmp/core.*
-rw----- 1 root root 10592256 Aug 12 19:40 /tmp/core.26252
-rw----- 1 root root 9924608 Aug 12 20:12 /tmp/core.26340
-rw----- 1 root root 10862592 Aug 12 20:14 /tmp/core.26374
-rw----- 1 root root 9105408 Aug 12 20:19 /tmp/core.26426
-rw----- 1 root root 9441280 Aug 12 20:20 /tmp/core.26462
-rw----- 1 root root 8331264 Aug 13 00:32 /tmp/core.26647
debian:/tmp#
```

In the event that there are multiple core files present (as in the above example), it is important to look at the file timestamps in order to determine which one you really intend to look at.

Now that we've verified the core file has been written to disk, the final part is to extract 'bt' from the core file. Core files are pretty big, don't be scared, it's normal.

**NOTE: Don't attach core files on the bug tracker, we only need the bt and bt full.**

For extraction, we use a really nice tool, called gdb. To verify that you have gdb installed on your system:

```
debian:/tmp# gdb -v
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-linux".
debian:/tmp#
```

Which is great, we can continue. If you don't have gdb installed, go install gdb.

Now load the core file in gdb, as follows:

```
debian:/tmp# gdb asterisk /tmp/core.26252
[...]
(You would see a lot of output here.)
[...]
Reading symbols from /usr/lib/asterisk/modules/app_externalivr.so...done.
Loaded symbols for /usr/lib/asterisk/modules/app_externalivr.so
#0 0x29b45d7e in ?? ()
(gdb)
```

Now at the gdb prompt, type: bt You would see output similar to:

```

(gdb) bt
#0 0x29b45d7e in ?? ()
#1 0x08180bf8 in ?? ()
#2 0xbcdf5a58 in ?? ()
#3 0x08180bf8 in ?? ()
#4 0xbcdf5a60 in ?? ()
#5 0x08180bf8 in ?? ()
#6 0x180bf894 in ?? ()
#7 0x0bf80008 in ?? ()
#8 0x180b0818 in ?? ()
#9 0x08068008 in ast_stopstream (tmp=0x40758d38) at file.c:180
#10 0x000000a0 in ?? ()
#11 0x000000a0 in ?? ()
#12 0x00000000 in ?? ()
#13 0x407513c3 in confcall_careful_stream (conf=0x8180bf8, filename=0x8181de8 "DAHDI/pseudo-1324221520") at app_meet
#14 0x40751332 in streamconfthread (args=0x8180bf8) at app_meetme.c:1965
#15 0xbcdf5be0 in ?? ()
#16 0x40028e51 in pthread_start_thread () from /lib/libpthread.so.0
#17 0x401ec92a in clone () from /lib/libc.so.6
(gdb)

```

The bt's output is the information that we need on the bug tracker.

Now do a bt full as follows:

```

(gdb) bt full
#0 0x29b45d7e in ?? ()
No symbol table info available.
#1 0x08180bf8 in ?? ()
No symbol table info available.
#2 0xbcdf5a58 in ?? ()
No symbol table info available.
#3 0x08180bf8 in ?? ()
No symbol table info available.
#4 0xbcdf5a60 in ?? ()
No symbol table info available.
#5 0x08180bf8 in ?? ()
No symbol table info available.
#6 0x180bf894 in ?? ()
No symbol table info available.
#7 0x0bf80008 in ?? ()
No symbol table info available.
#8 0x180b0818 in ?? ()
No symbol table info available.
#9 0x08068008 in ast_stopstream (tmp=0x40758d38) at file.c:180
No locals.
#10 0x000000a0 in ?? ()
No symbol table info available.
#11 0x000000a0 in ?? ()
No symbol table info available.
#12 0x00000000 in ?? ()
No symbol table info available.
#13 0x407513c3 in confcall_careful_stream (conf=0x8180bf8, filename=0x8181de8 "DAHDI/pseudo-1324221520") at app_meet
    f = (struct ast_frame *) 0x8180bf8
    trans = (struct ast_trans_pvt *) 0x0
#14 0x40751332 in streamconfthread (args=0x8180bf8) at app_meetme.c:1965
No locals.

```

```

#15 0xbcdfbe0 in ?? ()
No symbol table info available.
#16 0x40028e51 in pthread_start_thread () from /lib/libpthread.so.0
No symbol table info available.
#17 0x401ec92a in clone () from /lib/libc.so.6
No symbol table info available.
(gdb)

```

We also need gdb's output. That output gives more details compared to the simple "bt". So we recommend that you use bt full instead of bt. But, if you could include both, we appreciate that.

The final "extraction" would be to know all traces by all threads. Even if asterisk runs on the same thread for each call, it could have created some new threads.

To make sure we have the correct information, just do: (gdb) thread apply all bt

```

Thread 1 (process 26252):
#0 0x29b45d7e in ?? ()
#1 0x08180bf8 in ?? ()
#2 0xbcdf8fa58 in ?? ()
#3 0x08180bf8 in ?? ()
#4 0xbcdf8fa60 in ?? ()
#5 0x08180bf8 in ?? ()
#6 0x180bf894 in ?? ()
#7 0x0bf80008 in ?? ()
#8 0x180b0818 in ?? ()
#9 0x08068008 in ast_stopstream (tmp=0x40758d38) at file.c:180
#10 0x000000a0 in ?? ()
#11 0x000000a0 in ?? ()
#12 0x00000000 in ?? ()
#13 0x407513c3 in confcall_careful_stream (conf=0x8180bf8, filename=0x8181de8 "DAHDI/pseudo-1324221520") at app_meetme.c:1965
#14 0x40751332 in streamconfthread (args=0x8180bf8) at app_meetme.c:1965
#15 0xbcdfbe0 in ?? ()
#16 0x40028e51 in pthread_start_thread () from /lib/libpthread.so.0
#17 0x401ec92a in clone () from /lib/libc.so.6
(gdb)

```

That output tells us crucial information about each thread.

Now, just create an output.txt file and dump your "bt full" (and/or "bt") ALONG WITH "thread apply all bt" into it.

Note: Please ATTACH your output, DO NOT paste it as a note.

And you're ready for upload on the bug tracker.

If you have questions or comments regarding this documentation, feel free to pass by the #asterisk-bugs channel on irc.freenode.net.