

# **Asterisk Business Edition**

## **System Administrator's Manual**

**Leif Madsen  
Bill Savage  
Steven Sokol  
Jared Smith**

# **Asterisk Business Edition: System Administrator's Manual**

by Leif Madsen, Bill Savage, Steven Sokol, and Jared Smith

## **Abstract**

This is the System Administrator's Manual for Release C of Asterisk Business Edition.

DRAFT

DRAFT

# Table of Contents

Copyright .....	xii
I. Getting Started .....	1
1. Foreword .....	2
2. Introduction .....	3
3. Installing Asterisk Business Edition .....	4
Mounting the CDROM .....	4
Installing Dependencies .....	4
Installing Asterisk Business Edition from CD .....	5
Registering Asterisk Business Edition .....	7
Enabling the Asterisk GUI .....	7
Starting Asterisk Business Edition .....	8
Installing Asterisk Business Edition from RPM .....	9
Installing additional packages .....	11
Uninstalling Asterisk Business Edition .....	12
Summary .....	13
II. GUI Configuration and Operation .....	14
4. The AsteriskGUI™ .....	15
AsteriskGUI Setup .....	16
Using the AsteriskGUI .....	18
5. AsteriskGUI™ Section Descriptions .....	20
User Extensions .....	20
Conferencing .....	22
Voicemail .....	24
Call Queues .....	26
Adding Service Providers .....	28
Configure Hardware .....	32
mISDN Config .....	35
Calling Rules .....	37
Incoming Calls .....	40
Voice Menus .....	41
Time Based Rules .....	44
Call Parking .....	46
Ring Groups .....	47
Asterisk Management Options .....	49
Advanced Options .....	49
III. Reference .....	52
A. Application Reference .....	53
AddQueueMember() .....	53
ADSIProg() .....	54
AgentCallbackLogin() .....	54
AgentLogin() .....	55
AgentMonitorOutgoing() .....	55
AGI() .....	56
AMD() .....	57
Answer() .....	58
AppendCDRUserField() .....	59
Authenticate() .....	59
Background() .....	60
BackgroundDetect() .....	61
Busy() .....	61
ChangeMonitor() .....	61

ChanIsAvail()	62
ChannelRedirect()	62
ChanSpy()	63
Congestion()	64
ContinueWhile()	64
ControlPlayback()	65
DateTime()	65
DBdel()	66
DBdeltree()	66
DeadAGI()	67
Dial()	67
Dictate()	72
Directory()	73
DISA()	73
DumpChan()	74
EAGI()	75
Echo()	75
EndWhile()	75
Exec()	76
ExecIf()	76
ExitWhile()	77
ExtenSpy()	77
ExternalIVR()	78
Flash()	78
FollowMe()	79
ForkCDR()	79
GetCPEID()	80
Gosub()	80
GosubIf()	81
Goto()	81
GotoIf()	82
GotoIfTime()	83
Hangup()	83
HasNewVoicemail()	84
HasVoicemail()	85
IAX2Provision()	85
ImportVar()	86
Log()	86
LookupBlacklist()	87
LookupCIDName()	87
Macro()	88
MacroExclusive()	89
MacroExit()	89
MacroIf()	90
MailboxExists()	90
MeetMe()	91
MeetMeAdmin()	92
MeetMeCount()	94
Milliwatt()	94
MixMonitor()	95
Monitor()	95
MorseCode()	96
MP3Player()	97
MusicOnHold()	97

NBScat()	97
NoCDR()	98
NoOp()	98
Page()	99
Park()	100
ParkAndAnnounce()	100
ParkedCall()	101
PauseMonitor()	101
PauseQueueMember()	102
Pickup()	102
Playback()	103
Playtones()	103
PrivacyManager()	104
Progress()	104
Queue()	105
QueueLog()	106
Random()	107
Read()	107
ReadFile()	108
RealTime	108
RealTimeUpdate()	109
Record()	109
RemoveQueueMember()	110
ResetCDR()	110
RetryDial()	111
Return()	111
Ringing()	112
SayAlpha()	112
SayDigits()	113
SayNumber()	113
SayPhonetic()	114
SayUnixTime()	114
SendDTMF()	115
SendImage()	115
SendText()	116
SendURL()	116
Set()	117
SetAMAFlags()	117
SetCallerID()	118
SetCallerPres()	118
SetCDRUserField()	119
SetGlobalVar()	119
SetMusicOnHold()	120
SetTransferCapability()	120
SIPAddHeader()	121
SIPDtmfMode()	121
SLAStation()	122
SLATrunk()	122
SoftHangup()	122
StackPop()	123
StartMusicOnHold()	123
StopMixMonitor()	124
StopMonitor()	124
StopPlaytones()	125

StopMusicOnHold() .....	125
System() .....	126
Transfer() .....	126
TryExec() .....	126
TrySystem() .....	127
UnpauseMonitor() .....	127
UnpauseQueueMember() .....	128
UserEvent() .....	128
Verbose() .....	129
VMAuthenticate() .....	129
VoiceMail() .....	130
VoiceMailMain() .....	131
Wait() .....	131
WaitExten() .....	132
WaitForRing() .....	132
WaitForSilence() .....	133
WaitMusicOnHold() .....	133
While() .....	134
Zapateller() .....	134
ZapBarge() .....	135
ZapRAS() .....	135
ZapScan() .....	135
B. Asterisk Dialplan Functions .....	137
AGENT .....	137
ARRAY .....	137
BASE64_DECODE .....	138
BASE64_ENCODE .....	138
BLACKLIST .....	138
CALLERID .....	139
CDR .....	139
CHANNEL .....	140
CHECK_MD5 .....	141
CHECKSIPDOMAIN .....	142
CURL .....	142
CUT .....	142
DB .....	143
DB_DELETE .....	143
DB_EXISTS .....	144
DUNDILOOKUP .....	144
ENUMLOOKUP .....	144
ENV .....	144
EVAL .....	145
EXISTS .....	145
FIELDQTY .....	145
FILTER .....	146
GLOBAL .....	146
GROUP .....	146
GROUP_COUNT .....	147
GROUP_LIST .....	147
GROUP_MATCH_COUNT .....	147
IAXPEER .....	148
IF .....	148
IFTIME .....	149
ISNULL .....	149

KEYPADHASH .....	149
LANGUAGE .....	150
LEN .....	150
MATH .....	150
MD5 .....	151
MUSICCLASS .....	151
QUEUE_MEMBER_COUNT .....	151
QUEUE_MEMBER_LIST .....	152
QUEUE_WAITING_COUNT .....	152
QUEUEAGENTCOUNT .....	152
QUOTE .....	153
RAND .....	153
REALTIME .....	153
REGEX .....	154
SET .....	154
SHA1 .....	154
SIP_HEADER .....	154
SIPCHANINFO .....	155
SIPPEER .....	155
SORT .....	156
SPEECH .....	156
SPEECH_ENGINE .....	156
SPEECH_GRAMMAR .....	157
SPEECH_SCORE .....	157
SPEECH_TEXT .....	157
SPRINTF .....	157
STAT .....	158
STRFTIME .....	158
STRPTIME .....	159
TIMEOUT .....	159
TXTCIDNAME .....	160
URIDECODE .....	160
URIENCODE .....	160
VMCOUNT .....	161
C. Command-line Reference .....	162
bang .....	162
abort halt .....	162
add .....	162
ael .....	163
agent .....	163
agi .....	164
answer .....	164
autoanswer .....	165
cdr status .....	165
clear profile .....	165
console .....	165
convert .....	166
core .....	166
database .....	169
debug .....	170
dialplan .....	170
dial .....	172
dnsmgr .....	172
dont include .....	172



dump agihtml .....	172
dundi .....	172
extensions reload .....	174
features show .....	174
file convert .....	174
flash .....	174
funcdevstate list .....	175
group show channels .....	175
help .....	175
http show status .....	175
iax2 .....	175
include context .....	177
indication .....	178
init keys .....	178
keys .....	178
load .....	179
local .....	179
logger .....	179
manager .....	179
meetme .....	180
mgcp .....	181
mixmonitor .....	181
module .....	181
moh .....	182
mute .....	182
no debug channel .....	183
odbc .....	183
originate .....	183
osp .....	183
oss .....	184
pri .....	184
queue .....	185
realtime .....	185
reload .....	185
remove .....	186
restart .....	186
rtcp .....	186
rtp .....	187
save .....	187
say .....	187
send .....	187
set .....	188
show .....	188
sip .....	188
skinny .....	190
sla .....	190
soft .....	191
stop .....	191
stun .....	191
transcoder .....	192
transfer .....	192
udptl .....	192
unload .....	192
unmute .....	193

voicemail .....	193
zap .....	193
D. AGI Reference .....	195
ANSWER .....	195
CHANNEL STATUS .....	195
DATABASE DEL .....	195
DATABASE DELTREE .....	196
DATABASE GET .....	196
DATABASE PUT .....	196
EXEC .....	197
GET DATA .....	197
GET FULL VARIABLE .....	197
GET OPTION .....	198
GET VARIABLE .....	198
HANGUP .....	198
NoOp .....	199
RECEIVE CHAR .....	199
RECORD FILE .....	199
SAY ALPHA .....	200
SAY DATE .....	200
SAY DATETIME .....	200
SAY DIGITS .....	201
SAY NUMBER .....	201
SAY PHONETIC .....	202
SAY TIME .....	202
SEND IMAGE .....	202
SEND TEXT .....	203
SET AUTOHANGUP .....	203
SET CALLERID .....	203
SET CONTEXT .....	204
SET EXTENSION .....	204
SET MUSIC ON .....	204
SET PRIORITY .....	205
SET VARIABLE .....	205
STREAM FILE .....	205
TDD MODE .....	206
VERBOSE .....	206
WAIT FOR DIGIT .....	206
E. Asterisk Manager Interface Actions .....	207
AbsoluteTimeout .....	207
AgentCallbackLogin .....	207
AgentLogoff .....	208
Agents .....	209
ChangeMonitor .....	210
Command .....	211
DBGet .....	212
DBPut .....	212
Events .....	213
ExtensionState .....	214
GetConfig .....	215
GetVar .....	215
Hangup .....	216
IAXNetstats .....	217
IAXPeers .....	217

ListCommands .....	218
Logoff .....	218
MailboxCount .....	219
MailboxStatus .....	219
MeetmeMute .....	220
MeetMeUnmute .....	221
Monitor .....	221
Originate .....	222
Park .....	223
ParkedCalls .....	224
PauseMonitor .....	225
Ping .....	226
PlayDTMF .....	226
QueueAdd .....	227
QueuePause .....	228
QueueRemove .....	229
QueueStatus .....	229
Queues .....	231
Redirect .....	231
SIPpeers .....	232
SIPShowPeer .....	233
SetCDRUserField .....	235
SetVar .....	235
Status .....	236
StopMonitor .....	237
UnpauseMonitor .....	238
UpdateConfig .....	238
UserEvent .....	239
WaitEvent .....	240
ZapDNDOff .....	241
ZapDNDon .....	241
ZapDialOffhook .....	242
ZapHangup .....	242
ZapRestart .....	243
ZapShowChannels .....	243
ZapTransfer .....	244

# Copyright

Table 1.



Digium, Inc.

445 Jan Davis Drive Huntsville, AL 35806, United States

Main Number: 1.256.428.6000

Tech Support: 1.256.428.6161

U.S. Toll Free: 1.877.344.4861

Sales: 1.256.428.6262

[vwww.digium.com](http://vwww.digium.com)

[www.asterisk.org](http://www.asterisk.org)

[www.asterisknow.org](http://www.asterisknow.org)

Copyright © 2007 Digium, Inc.

All rights reserved.

Portions of this manual are Copyright © 2005-2007 O'Reilly Media, Inc. Used with permission.

No part of this publication may be copied, distributed, transmitted, transcribed, stored in a retrieval system, or translated into any human or computer language without the prior written permission of Digium, Inc.

Digium, Inc. has made every effort to ensure that the instructions contained in this document are adequate and error free. The manufacturer will, if necessary, explain issues which may not be covered by this documentation. The manufacturer's liability for any errors in the documents is limited to the correction of errors and the aforementioned advisory services.

This document has been prepared for use by professional and properly trained personnel, and the customer assumes full responsibility when using it.

Adobe and Acrobat are registered trademarks, and Acrobat Reader is a trademark of Adobe Systems Incorporated.

Asterisk and Digium are registered trademarks and Asterisk Business Edition, AsteriskNOW, AsteriskGUI are trademarks of Digium, Inc. Any other trademarks mentioned in the document are the property of their respective owners.

# Part I. Getting Started

DRAFT

# Chapter 1. Foreword

Dear Asterisk User,

Thank you for purchasing Asterisk Business Edition. I consider myself quite fortunate to have been able to seed and participate in a project as fundamental and as wide-reaching as Asterisk. From its humble beginnings as a Linux-based PBX, originally written for my little startup company, the open source development model has allowed an enormous and growing community of users, developers, and customers to launch Asterisk into an industry-shattering, world-wide phenomenon.

Digium's primary purpose is to advance Asterisk development and deployments. While few question the developmental advantages of the open source model, not all businesses are prepared to operate a system as fundamental to their business as their phone system without more traditional product warranties, more support, and a clear commercial license when incorporated with proprietary components. Asterisk Business Edition builds upon the open source model, by providing the same core software, under more traditional licensing terms, with more substantial software warranties and support to address this need.

Your purchase of Asterisk Business Edition not only entitles you to first-class technical support from Digium's team of core Asterisk developers and support technicians, but also directly benefits the Asterisk project at large by supporting many of the programmers who contribute to its stability, interoperability and features on a daily basis.

On behalf of everyone at Digium, I again thank you for your purchase and encourage you to contact us with any Asterisk-related questions or needs you might have in the future.

Mark Spencer

Original Author and Project Manager, Asterisk

CTO, Digium

# Chapter 2. Introduction

Asterisk Business Edition provides the enterprise environment with a tested and stable edition of Asterisk for mission critical applications. Also, developers and turnkey providers who require a non-GPL version can confidentially distribute Asterisk Business Edition within a product or development platform.

Asterisk Business Edition allows the following advantages:

- You have the most stable, tested version of Asterisk available
- The product includes a full one-year warranty
- A new installer is provided that includes a Linux distribution customized for Asterisk Business Edition, making it easy to install Linux and Asterisk
- Technical Support is included for installation and critical issues
- The product comes with a commercial license for your protection
- Free updates and support are provided for a full year and can be renewed for a nominal fee
- Customers have access to a special Asterisk Business Edition portal on the Digium website for obtaining troubleshooting info, support, partner products, updates, and supplemental downloads
- Customers have access to a special Asterisk Business Edition portal on the Digium website for the latest listing of hardware and software
- The product comes with a commercial license for your protection
- Several software partner products are included, either as trials or with limited number of channels, such as Text-to-Speech by Cepstral™ and Speech Recognition by Lumenvox™
- A range of Partner products are certified for operation with Asterisk Business Edition. Please see our website under Ecosystem for a complete list of certified compatible products.

Major new releases of Asterisk Business Edition will be provided approximately twice per year. Each new version is fully regression tested using a test suite of over 1300 test cases, including functionality, stress testing, and certification with Digium and Partner products. Minor releases are provided as needed to address critical issues. All product modifications and revisions are risk-analyzed to maximize stability and reliability of the product.

# Chapter 3. Installing Asterisk Business Edition

In this chapter we will explore how to install Asterisk from the `install.sh` script found on the Asterisk Business Edition (ABE) installation disk. The supported operating systems include RedHat Enterprises Linux (RHEL) 4 & 5, and Fedora Core (FC) 6 & 7. The instructions in this chapter will assume that you've installed a bare minimal configuration set and that you will be installing the necessary packages for Asterisk Business Edition with `yum`.

## Mounting the CDROM

After you have installed one of the supported operating systems for Asterisk Business Edition (ABE), you can then insert one of the two discs into your cdrom. Disc 1 contains ABE for 32-bit architectures such as Intel P4 and AMD Athlon XP. Disc 2 contains ABE for 64-bit architectures such as Intel Xeon and AMD Opteron.

Next, mount the cdrom using the following commands.



### Note

If you've previously created the `/mnt/cdrom/` directory, then you do not need to create it again



### Warning

If you have multiple cdrom devices connected to your system, the location of your cdrom may be different.

```
# mkdir /mnt/cdrom
# mount /dev/cdrom /mnt/cdrom
mount: block device /dev/cdrom is write-protected, mounting read-only
```

## Installing Dependencies

In order to install Asterisk Business Edition, you will need to install a few dependencies in order for the software to compile and install the appropriate modules. For a base install with none of the optional modules such as the *Speex codec*, *Ogg Vorbis format*, *mISDN*, *Cepstral Text-to-Speech*, or the *Transnexus Open Settlement Protocol*, then you will need the following packages:

- Perl
- libcurl
- unixODBC
- ALSA
- GNU C compiler
- make
- Kernel sources



- Newt development packages



## Note

When you install the kernel sources, your system may install the latest released version of the kernel sources which may not necessarily match your currently running kernel. Be sure to install the appropriate kernel sources for your currently running kernel, or upgrade to the latest kernel and reboot your system before installing Asterisk Business Edition.

Install the necessary dependencies using the **yum** application with the following command from your system console:

```
# yum install perl curl alsa-lib gcc make unixODBC kernel-devel newt-devel
```

You will then be prompted to confirm installation of the packages along with any dependencies required by the above applications or libraries. Confirm you wish to install the packages and continue.

See Installing additional packages for more information about installing dependencies for the optional packages.

# Installing Asterisk Business Edition from CD

Change to the directory where you mounted your cdrom (/mnt/cdrom/) and execute the `install.sh` script.

```
# cd /mnt/cdrom
# ./install.sh
```

After starting the script, you will be asked a series of questions asking you which packages you want to install. We will explore a basic installation with no additional packages installed. If you wish to install the optional packages now (those selected as 'n' in the following script), be sure you have all the necessary dependencies installed (see Installing additional packages). If you choose not to install them now, you can still install them at a later time.

```
*****
*****  ASTERISK BUSINESS EDITION C.1.0  *****
*****
```

```
Would you like to install the open-source codec Speex? (Y/n) n
Would you like to install support for Ogg Vorbis playback? (Y/n) n
***** Checking for required packages *****
```

After the required packages have been verified to be installed, then you will be prompted to read acknowledge you agree to the Digium End-User Purchase and License Agreement. Then you will be prompted about a few more packages you may wish to install. The options shown below are for a basic installation.

```
Do you agree to the terms of the EULA? (y/n) y

Do you wish to install sample configs? (Y/n) y
Would you like to install mISDN support for Asterisk? (Y/n) n
Would you like to install the Cepstral Text-to-Speech connector
for Asterisk? (Y/n) n
Would you like to install Transnexus Open Settlement Protocol (OSP)
Extras? (Y/n) n
```

Then the Zaptel drivers will be built and installed, followed by installation of the Asterisk Business Edition packages you have selected.

```

***** ZAPTEL BUILT SUCCESSFULLY *****

***** Removing any existing Asterisk RPMS *****
*****          It could be a minute          *****

***** Attempting to install RPMS *****

Preparing... #####
asterisk-core #####
libpri #####
libtonezone #####
asterisk-doc #####
asterisk-curl #####
asterisk-alsa #####
asterisk-odbc #####
asterisk-gui #####
asterisk-configs #####
Preparing... #####
asterisk #####

***** Successfully installed Asterisk Business Edition RPMS *****

Would you like to install extra Asterisk sounds? (Y/n) y
Would you like Asterisk to start on boot? (Y/n) y
*****
Asterisk and Zaptel have been added to your services
service <asterisk or zaptel> {start|stop|restart|status}

Be sure to edit /etc/sysconfig/zaptel to refine the modules you wish to load
*****
Would you like to register your product? (y/n)

```

After the packages have finished building and installing you can choose to register your copy of Asterisk Business Edition, or continue on with the installation and register Asterisk after by running the **register** script on the Asterisk installation cdrom. For more information about registering your product, see Registering Asterisk Business Edition.

```

*****
Thank You For Installing Asterisk Business Edition
  To use the Asterisk GUI:
    1. Create or uncomment default account in /etc/asterisk/manager.conf
    2. Set 'enabled' and 'webenabled' to 'yes' in /etc/asterisk/manager.conf
    3. Set 'enabled' to 'yes' in /etc/asterisk/http.conf
*****
Would you like to view the Asterisk Business Edition README? (y/n)

```

And voila, Asterisk has been installed! If you are new to Asterisk Business Edition, or you are upgrading from a previous release, please peruse the README file.

In the next few sections we are going to explain Registering Asterisk Business Edition, Enabling the Asterisk GUI, and Starting Asterisk Business Edition.

# Registering Asterisk Business Edition

Before we get too involved, Asterisk Business Edition requires registration and activation using the code supplied on the included certificate. In order to register your system with Digium using the information on the certificate, we need to execute the **register** application located on your installation CD.

```
# cd /mnt/cdrom
# ./register          ( pass '-v' for verbose
                     mode if you have any trouble)
```



## Note

Once you activate and register your copy of Asterisk Business Edition, it will not be eligible for return or refund, so be sure you have carefully read the End User License Agreement before activation.

To run the registration application, your PC must have Internet access to contact the Digium license server. Port 433 is used by default and must be open through any firewalls to complete the automated activation process. The PC must also have at least one Ethernet device installed.

```
Select the following:
  1. Digium Products
--> 1. Asterisk Business Edition
```

Enter your Activation Code

The registration application will then ask you for the following information:

- First Name
- Last Name
- Company/Organization
- Mailing Address
- Phone Number
- Email Address

Supplying the correct email address is important. The one specified during registration gives you access to the Asterisk Business Edition Portal, <http://be.digium.com>. This area of the Digium website provides a multitude of documentation, updates and support for Asterisk Business Edition.

And now you should have a successfully registered product!

## Enabling the Asterisk GUI

To enable the Asterisk GUI interface, modify the `/etc/asterisk/manager.conf` file to allow the web interface to communicate with Asterisk.

Set `enabled` and `webenabled` to `yes` in the `[general]` section of the file:

```
[general]
displaysystemname = yes
enabled = yes
webenabled = yes
port = 5038
```

Then uncomment or modify the [admin] account near the bottom of the file.

```
;*****
;* Asterisk Business Edition GUI users: *
;*          uncomment and/or modify the account below *
;*          to access the GUI *
;*****
;
[admin]
secret = mysecret
deny=0.0.0.0/0.0.0.0
permit=0.0.0.0/0.0.0.0
read = system,call,log,verbose,command,agent,user,config
write = system,call,log,verbose,command,agent,user,config
```

Then modify the /etc/asterisk/httpd.conf file to enable the mini-HTTP server by setting enabled to yes in the [general] section.

```
[general]
;
; Whether HTTP interface is enabled or not. Default is no.
;
enabled=yes
```

Now you are ready to start Asterisk and get started configuring your system. In Starting Asterisk Business Edition, we will show you how to get your system started. For more information about configuring your Asterisk system via the web interface, see Part II: GUI Configuration and Operation.

## Starting Asterisk Business Edition

Now that you've successfully installed and registered Asterisk Business Edition, you are ready to let it loose. To get started, run the following two commands.

```
# service zaptel start
# service asterisk start
```

Before loading the Zaptel modules with the service application, modify /etc/sysconfig/zaptel so that it reflects your installed hardware. If you have no hardware installed, then comment out all MODULES lines except the line that loads the ztdummy driver. Commented lines start with a #, while uncommented lines have no #. After modifying /etc/sysconfig/zaptel, then start Zaptel.

```
# service zaptel start
```

```
Loading zaptel framework: [ OK ]
Waiting for zap to come online...OK
Loading zaptel hardware modules: ztdummy.
Running ztcfg: [ OK ]
```

After loading Zaptel, you can start Asterisk in a similar manner:

```
# service asterisk start
```

```
Starting asterisk: [ OK ]
```

Once Asterisk starts successfully you can connect to the Asterisk command line interface (CLI) by running **asterisk -rvvvv**.

```
# asterisk -rvvv
```

```
Asterisk Business Edition C.1.0, Copyright (C) 1999 - 2007 Digium, Inc.  
and others.
```

```
Created by Mark Spencer
```

```
Thank you for using Business Edition. This Software is provided by Digium Inc  
under license. Please refer to the license agreement provided with the
```

```
Software.
```

```
=====
== Parsing '/etc/asterisk/asterisk.conf': Found
== Asterisk Business Edition Host-ID: xx:xx:xx:xx:xx:xx
== Found license 'ABE-XXXXXXXXXXXX' providing 128 calls
== Found total of 128 Business Edition calls
== Parsing '/etc/asterisk/extconfig.conf': Found
Connected to Asterisk C.1.0 currently running on localhost (pid = 28110)
localhost*CLI>
```

And that's it. You've successfully started Asterisk Business Edition!

## Installing Asterisk Business Edition from RPM

In this section we will provide instructions to install Asterisk Business Edition manually from the RPM packages. You can also perform a guided installation of Asterisk Business Edition. For more information see Installing Asterisk Business Edition from CD.

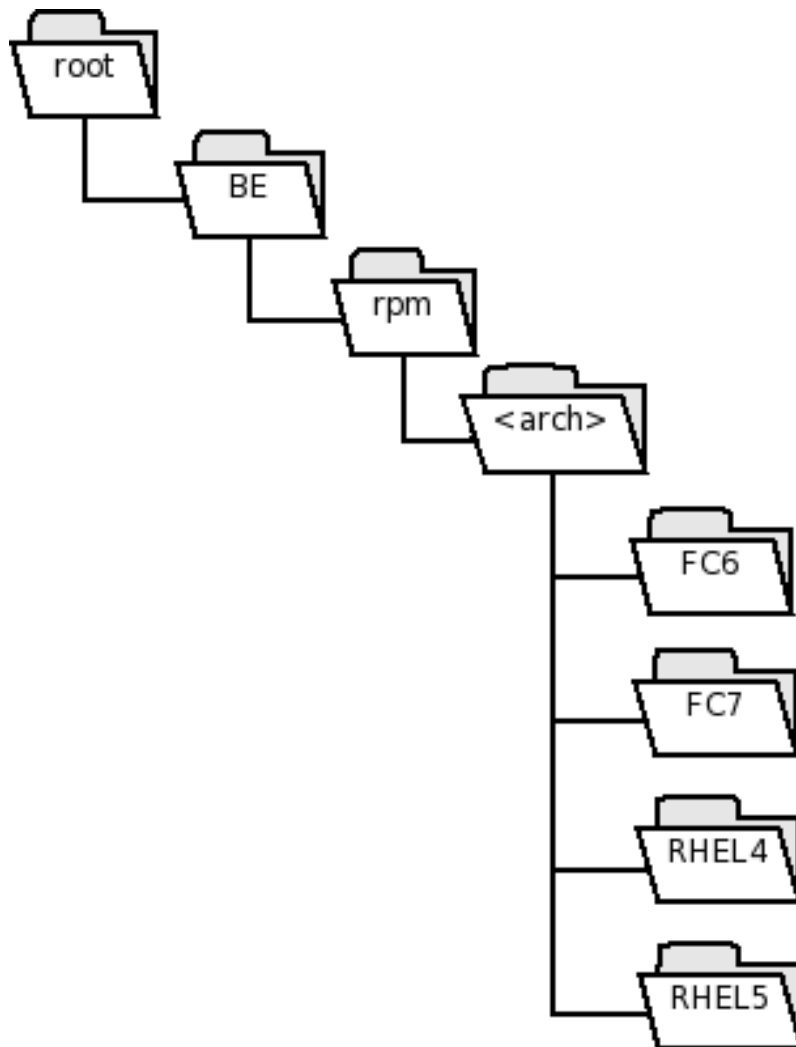


### Note

You must have superuser privileges in order to update/delete/install packages with **yum** or **rpm**.

You will find the RPMs supplied on the two Asterisk Business Edition CDs. Be sure to select the correct disk for the architecture you'll be installing to (disk 1 contains installation for 32-bit platforms, and disk 2 for 64-bit platforms).

The following figure shows the location of the RPMs on the Asterisk Business Edition CDs. The folder labeled <arch> will either be named i386 or x86\_64 for 32-bit and 64-bit operating systems respectfully.

**Figure 3.1. RPM location tree**

## Installing with *yum*

To install a package:

```
# yum install <packagename>
```

To update a package (or all packages on the system):

```
# yum update <packagename>
```

To search for a package:

```
# yum search <packagename>
```

If yum complains about the GPG key not being imported, then then you must import the keys before you will be able to install anything. To import a key, you use the "rpm --import <key>" command.

```
# rpm --import /usr/share/rhn/*GPG-KEY*
```

## Install with *rpm*

To install an RPM, use the "`rpm -ivh <rpmfilename>`" command:

```
# rpm -ivh asterisk-core-C.1.0_RHEL5.x86_64.rpm
```

## Installing additional packages

There are several additional packages you can install for Asterisk Business Edition in order to give you added functionality from external applications. We have given examples of the full filename for the RHEL5, 64-bit platform, although the commands could also be written as:

```
rpm -ivh <packagename>-<version>*.rpm
```

Where *<packagename>* would be something like `asterisk-ogg`, or `asterisk-misdn`, and *<version>* would be the version of Asterisk Business Edition you are installing, for example, `C.1.0`.

## Speex

You will need to install the Speex codec libraries before installing the `asterisk-speex` RPM. You can find the RPMs for Fedora Core 6, 7 and RedHat Enterprise Linux 4, 5 at <http://dag.wieers.com/rpm/packages/speex/>.

To install the Speex libraries on RHEL5, 64-bit platform, install `wget` and `libogg` (a dependency of Speex) with the following commands:



### Warning

Be sure to download the correct RPM for your system!

```
# yum install wget libogg
# cd /usr/src/
# wget \
http://dag.wieers.com/rpm/packages/speex/speex-<VERSION>.el5.rf.x86_64.rpm
# rpm -Uvh speex*.rpm
```

Once you've installed the dependencies, you can now install the `asterisk-speex` package. On the RHEL5, 64-bit platform execute:

```
# rpm -ivh asterisk-speex-C.1.0_RHEL5.x86_64.rpm
```

## OggVorbis Support

You will need to install the `libogg` and `libvorbis` libraries before installing the `asterisk-ogg` RPM. Try installing with **yum**:

```
# yum install libogg libvorbis
```

Then install the `asterisk-ogg` RPM. On RHEL5, 64-bit platform, you'd do:

```
# rpm -ivh asterisk-ogg-C.1.0_RHEL5.x86_64.rpm
```

## mISDN Support

The `asterisk-misdn` package does not have any additional dependencies. On the RHEL5, 64-bit platform you would install:

```
# rpm -ivh asterisk-misdn-C.1.0_RHEL5.x86_64.rpm
```

Please use the Asterisk GUI to configure your digital cards. If you wish to configure and use `chan_misdn.so` in Asterisk without the Asterisk GUI, you will need to do the following:

- Install your Digium B410P according to your install guide and reboot.
- As root, type

```
mISDN scan; mISDN config; mISDN start
```

and hit <return>

## Cepstral Support

The `asterisk-cepstral` package does not have any additional dependencies. On the RHEL5, 64-bit platform you would install:

```
# rpm -ivh asterisk-cepstral-C.1.0_RHEL5.x84_64.rpm
```

## Open Settlement Protocol (OSP) Support

The `asterisk-osp` package does not have any additional dependencies. On the RHEL5, 64-bit platform you would install:

```
# rpm -ivh asterisk-osp-C.1.0_RHEL5.x86_64.rpm
```

## Uninstalling Asterisk Business Edition

To remove Asterisk Business Edition, execute the following command:

```
# rpm -e asterisk \  
    asterisk-core \  
    asterisk-curl \  
    asterisk-odbc \  
    asterisk-configs \  
    asterisk-doc \  
    asterisk-alsa \  
    asterisk-gui \  
    libpri \  
    libtonezone
```

If you had Asterisk and Zaptel run at startup, remove the startup scripts from the `/etc/init.d` directory.

```
# rm /etc/init.d/asterisk  
# rm /etc/init.d/zaptel
```



## Summary

After this chapter you should have a fully installed Asterisk system and are now ready to start configuring. The rest of this book will guide you into getting aquanted with the exciting world of Asterisk. In Part II: GUI Configuration and Operation we'll make use of the Asterisk GUI interface to provision our system. In Part III: Manual Configuration and Operation, we'll delve into the heart of Asterisk and start exploring the tools you'll need to build any custom applications you can think of.

## **Part II. GUI Configuration and Operation**

DRAFT

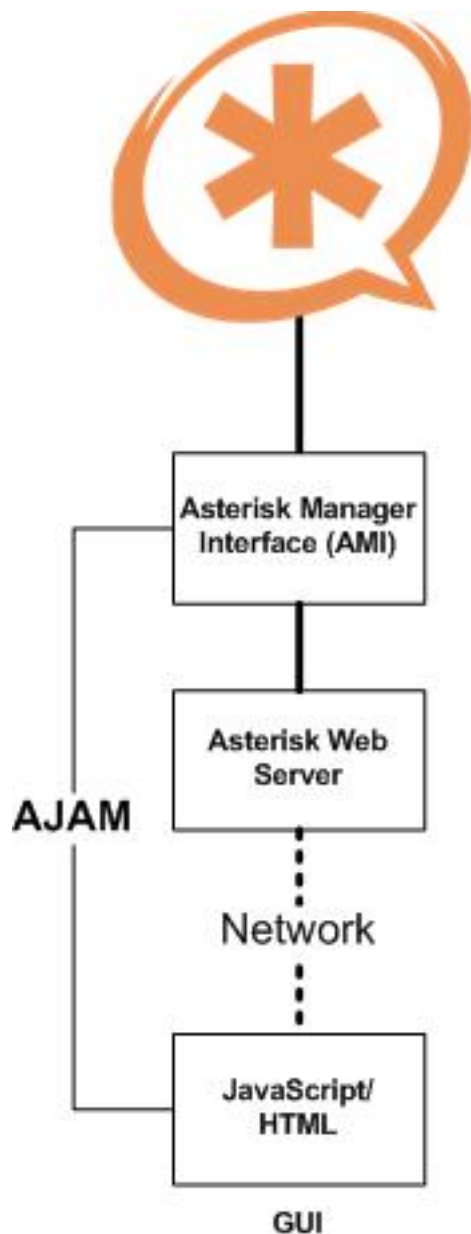
# Chapter 4. The AsteriskGUI™

*Assigned to Bill Savage*

While your system admin can set up your Asterisk system, there are certainly less technical folks that would like to change a few things without bothering your already beleaguered technical expert. As the number of Asterisk users has increased, the demand for an easier way to manage the day to day administration of an Asterisk implementation has also increased. A graphical interface (GUI) could meet that demand. Asterisk GUIs created in the past have been typically targeted at a specific audience or system configuration. The GUI which works for a system administrator could be beyond difficult for a casual user. A GUI created for a PBX implementation may not work properly for a retail business.

Digium® have developed several GUIs, as well as GUI framework, to meet the needs of users who want to use a graphical interface to manage their implementation. The framework as well as the GUIs created are referred to as the AsteriskGUI™. The AsteriskGUI utilizes a web based interface to communicate directly with Asterisk. The AsteriskGUI utilizes the configuration files with which many users are familiar. Changes made through the GUI are saved to the appropriate configuration (.conf). Likewise modifications of configuration files are reflected in the GUI. Options not displayed on a GUI configuration can be enabled using traditional methods as well as through a special GUI command interface.

The AsteriskGUI interface is displayed through a web browser. The web interface is comprised of HTML with Javascript (and a couple of associated libraries) that allow a configuration web page running on the client browser to communicate directly with Asterisk (assuming it has permission to) to directly dictate the necessary changes to the system. Changing the GUI only requires the editing of the associated HTML and Javascript, without server-side changes. The components which allow the AsteriskGUI to function are collectively referred to as AJAM, or Asynchronous Javascript Asterisk Manager. AJAM is a new technology which allows web browsers or other HTTP enabled applications and web pages to directly access the Asterisk Manager Interface (AMI) via HTTP. The Asterisk Manager Interface is an API which allows a client program to connect to Asterisk and issue commands or read events over a TCP/IP stream. The AsteriskGUI connects to Asterisk through the AMI. Asterisk's web server provides the functionality needed to serve the GUI interface. Lastly, the interface portion of AJAM is created with a combination of primarily Javascript and HTML. The following illustration gives you an idea of how all of these components work together.

**Figure 4.1. AsteriskGUI Framework**

This chapter serves as an introduction to the components which comprise the GUI and help it work with Asterisk. The enabling of the GUI and its components is needed even though they are installed as part of the Business Edition distribution.

## AsteriskGUI Setup

All of the AsteriskGUI files are installed during the Business Edition installation process. The GUI is not, however, activated by default. In order to activate the GUI you will need to edit a few configuration files. Configuring the Asterisk web server to process AJAM requests involves enabling the web server, enabling GUI access, and configuration of your distribution. Use the following procedure to setup the server and enable AJAM access, as well as enable the GUI content.

## Enable the Server

The simplified web server which comes with Asterisk Business Edition is used to serve the GUI content. The web server must be enabled before the content can be displayed. The web server can be enabled by making a few changes to the `http.conf` file. Use the following procedure to enable the AsteriskGUI web server.

1. Using a text editor, such as `vi`, open the `http.conf` for editing. This file is located in `/etc/asterisk/`.
2. Remove the comment marks from the line `“enabled=yes”`. This action will enable Asterisk's built-in micro web server.
3. Remove the comment marks from the line `“enablestatic=yes”` if you want Asterisk to deliver simple HTML pages, CSS, javascript, etc.
4. Adjust the `“binaddr”` and `“bindport”` settings as appropriate for your accessibility.
5. Adjust the `“prefix”` setting as needed. The prefix is the beginning of any URI on the server. The default is `“asterisk”`. The rest of the instructions assume the default value.
6. Save your changes and close the `http.conf` editing session.

## Enable the AMI

The Asterisk Manager Interface (AMI) must be enabled so that changes made through the GUI can be communicated to your Asterisk implementation. The AMI can be enabled by making a few changes to the `manager.conf` file. Use the following procedure to enable the AMI.



### Note

You should not enable the AMI on a public IP address. If needed, block this TCP port with iptables (or another FW software) and reach it with IPsec, SSH, or SSL vpn tunnel.

1. Using a text editor, such as `vi`, open the `manager.conf` file for editing. The `manager.conf` file located in `/etc/asterisk/`.
2. Remove the comment marks from the line `“enabled=yes”`.
3. Remove the comment marks from the line `“webenabled=yes”`.
4. Create a manager username and password. The manager user created must have a `“config”` access level for both read and write.
5. Save your changes and close the `manager.conf` editing session.

## Restart Asterisk

Once you have completed configuration of the `http.conf` and `manager.conf` files, restart Asterisk. Restarting Asterisk will restart the web server with your new configuration. Access to web-based functions will be enabled. You will be able to access these functions by entering specific URLs in a browser's location field. The URL used to access the current AsteriskGUI uses the following format: `http://<host>:8088/asterisk/static/config/cfgbasic.html` **<host>** is the IP address or hostname that is used to connect to the Asterisk server. **8088** is the port number used to connect to the Asterisk HTTP server.

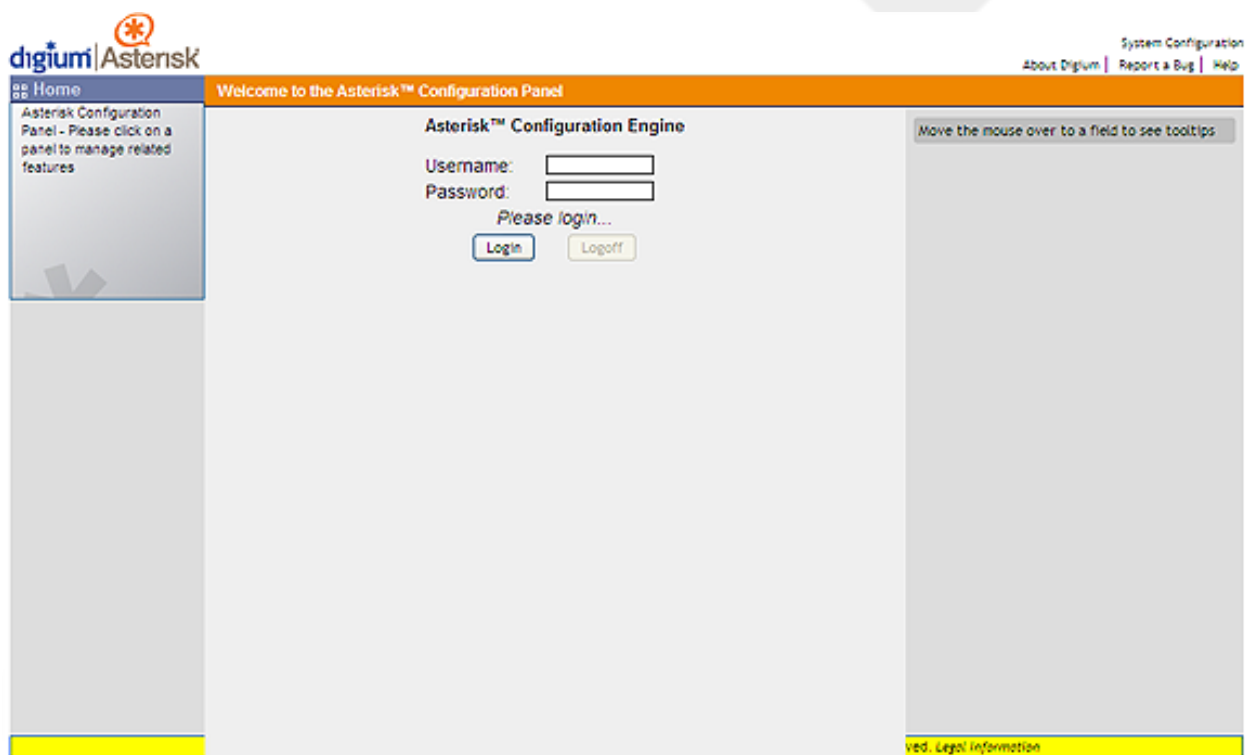
# Using the AsteriskGUI

The AsteriskGUI gives you the ability to configure and manage your Asterisk implementation. You can set up your Asterisk implementation through the GUI, or through the command line. Any Asterisk options that are not displayed in the GUI can be edited through either the command line or through a special GUI command interface. This section discusses how to log on to the AsteriskGUI, as well as an overview of the standard GUI delivered with Asterisk Business Edition.

## GUI Log On

In the address field of a browser (Firefox is recommended), enter the URL assigned to your Asterisk server. The initial GUI log on page is displayed.

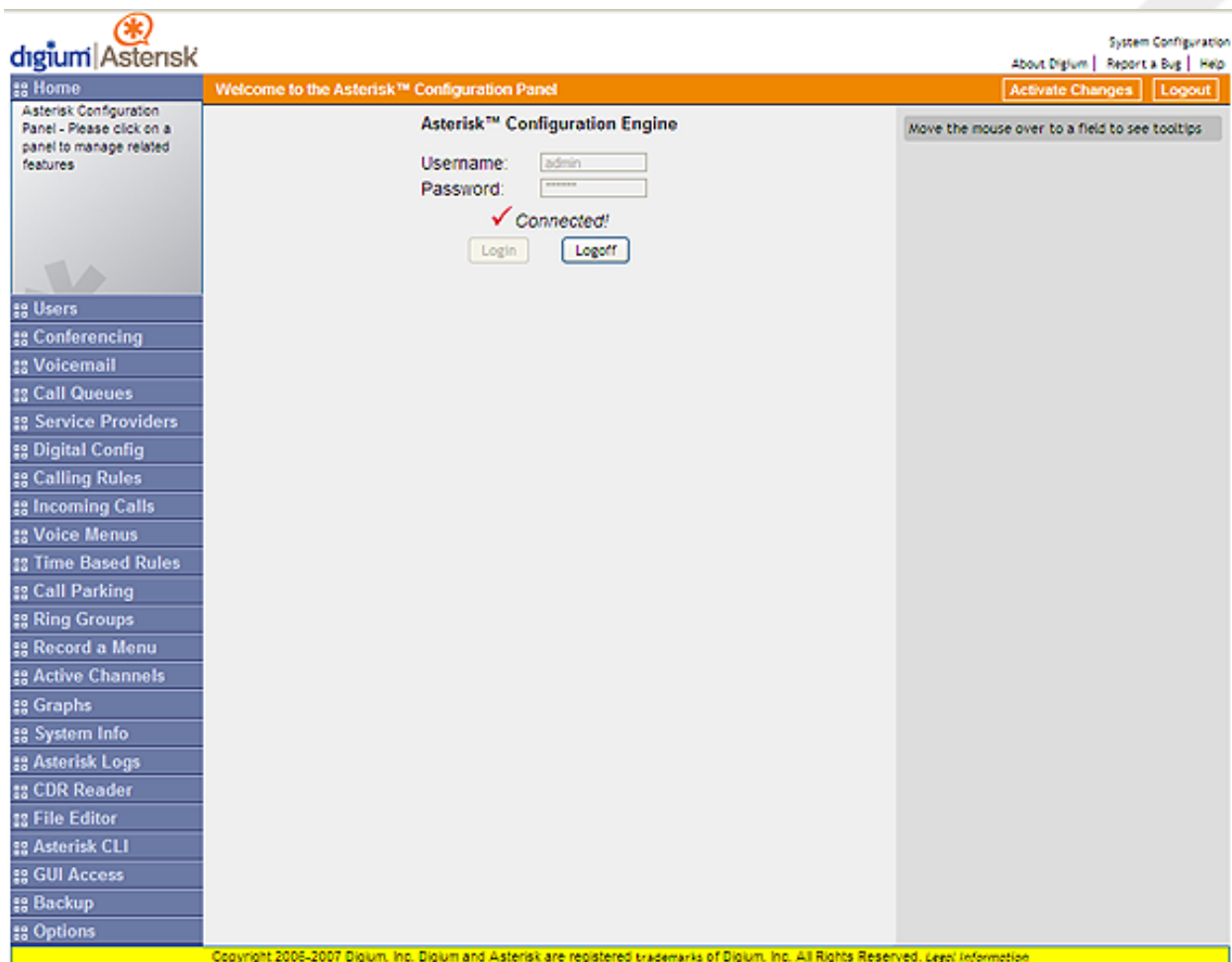
Figure 4.2. Login Page



The first time you log on you will be prompted to change your password from the default. You will then be prompted to log on with the new password. Once the log on process is complete the AsteriskGUI home page will be displayed.

## GUI Overview

Every page of the GUI has three columns. The column to the furthest left identifies all the elements for which you can program your Asterisk implementation. The elements listed begin with Home and proceed down to Options. They are listed in the order of usage frequency, with each heading down the list utilizing information specified in a previous tab. For example, the users configured in the Users tab are used to assign voice mail options on the Voicemail tab and can be included in the Call Queues.

**Figure 4.3. Home Page**

Clicking any of the tabs on the left of the page opens the heading to list a brief explanation of what's available in that section of the GUI, as well as to add the options to the center section.

The middle of the GUI contains the primary content for each page. The Home page is used for log on and log off purposes.

The far right column of the GUI contains Tooltips. This area provides explanations for any element of the GUI. To see a brief description of any tab on the left, or of some menu elements in the center section, just mouse over the item. The Tooltips section of the page will immediately populate with a definition of the heading.

Above the tooltips section you will see the Activate Changes and Logout buttons. These buttons appear in the upper right corner of every page. Click Activate Changes to activate changes you have made on a page so that you can utilize the changes. The changes will be saved immediately and made active. Click Logout on any page to exit the AsteriskGUI.

# Chapter 5. AsteriskGUI™ Section Descriptions

*Assigned to Bill Savage*

While your system admin can set up your Asterisk system, there are certainly less technical folks that would like to change a few things without bothering your already beleaguered technical expert. As the number of Asterisk users has increased, the demand for an easier way to manage the day to day administration of an Asterisk implementation has also increased. A graphical interface (GUI) could meet that demand. Asterisk GUIs created in the past have been typically targeted at a specific audience or system configuration. The GUI which works for a system administrator could be beyond difficult for a casual user. A GUI created for a PBX implementation may not work properly for a retail business.

## User Extensions

Click the User Extensions tab and you'll see the extensions you created in the initial setup process.

**Figure 5.1. User Extensions**

The screenshot displays the AsteriskGUI interface for managing user extensions. On the left, a sidebar contains navigation links for Home, Users, Conferencing, Voicemail, Call Queues, Service Providers, Configure Hardware, mISDN Config, Calling Rules, Incoming Calls, Voice Menus, Time Based Rules, Call Parking, Ring Groups, Record a Menu, Active Channels, System Info, Asterisk Logs, CDR Reader, File Editor, Asterisk CLI, Backup, and Options. The main content area is titled 'User and Phone Configuration' and includes a 'User Extensions' list on the left with entries like '6001 -- John Doe', '6123 -- Conference Bridge', and '6500 -- Check Voicemail'. The central form is for editing extension 6007, with fields for Name (John Doe), Password (5555), VM Password (1234), E-mail (jdoe@digium.com), Caller ID (256-428-6007), Analog Phone (Analog Port #5), Dial Plan (DialPlan1), and Phone Serial. Below these are 'Extension Options' with checkboxes for Voicemail, Email Only, SIP, CTI, Call Waiting, Can Reinvite, In Directory, IAX, Is Agent, 3-Way Calling, NAT, and DTMFMode. A 'Where to Buy' button is located at the bottom right. The footer contains copyright information for Digium, Inc. from 2006-2007.

The following information comprises a user extension definition:



- **Extension** - The extension assigned to the defined user.
- **Name** - The first and last name of the individual assigned to this extension. The name can also be that of a department, such as Sales or Support, for example. This is important because the Dial By Name Directory function of Asterisk uses this information to route calls.
- **Password** - The password for the user's sip/iax account.
- **VM Password** - The password used to access voicemail for the specified extension.
- **E-mail Address** - Voice mails received by this extension can be sent as audio file attachments e-mailed to a specific address.
- **Caller ID** - Identifies the Caller ID presented when the listed extension dials out.
- **Analog Phone** - A drop-down menu is available to identify the analog phone port which this extension will access. If more than one phone is connected to your Asterisk system you will need to confirm the port number listed on your hardware card.
- **Dial Plan** - This option references the Calling Rules option on the left tool bar. Based on the calling rules you've created, you can restrict the outbound dialing of this extension to local calls, emergency calls, and standard long-distance calls for North America. This option also allows blocking or allowing international (011 prefix dialed) calls.
- **Phone Serial** - This field is used to enter the serial number of a Polycom phone to enable phone provisioning. Once enabled the phone will be provisioned.

There are also several advanced extension options available. The advanced options establish the connections from the listed extension to other systems within the Asterisk server. These systems include the following:

- **Voicemail** - Builds a voice mail box for the extension that can be reached by dialing the Check Voicemail extension. The Voicemail extension can be configured. The current default is 6050.
- **In Directory** - Asterisk establishes a directory of all extensions so that inbound callers can reach someone in your office by dialing the first few digits of the person's first or last name. The company directory includes only the name of the extension if this option is checked.
- **E-mail Only** - Select this option to send voicemail messages to the specified e-mail address without storing the message in the mailbox.
- **SIP** - Identifies whether the extension sends and receives calls using the VoIP protocol SIP.
- **IAX** - Identifies whether the extension sends and receives calls using the VoIP protocol IAX.
- **CTI** - Selecting this option (Computer Telephony Integration) allows the user to connect applications to the Asterisk Management Interface.
- **Is Agent** - Call queuing is made up of a bank of agents who receive calls. An extension listed as Is Agent can be added to queues from the CallQueues option on the left toolbar.
- **Call Waiting** - If call waiting is not enabled, the extension accepts only one call before it is identified as busy.
- **3-Way Calling** - Allows the extension to receive a call and then dial out to another phone number to conference with the inbound call and the recipient of the outbound call.
- **Can Reinvite** - This option can be used to tell the Asterisk server whether or not to issue a reinvite to the client.

- **NAT** - Try this setting when Asterisk is on a public IP, communicating with devices behind a NAT device (broadband router). If you have one-way audio problems, you usually have problems with your NAT configuration or your firewall's configuration of SIP and RTP ports.
- **DTMF Mode** - Set the default DTMF mode for sending DTMF (touch tone). The default setting is rfc2833. Other options include:
  - **info** - Used to display SIP Info messages
  - **inband** - Inband audio (requires 64 kbit codec - alaw, ulaw)
  - **auto** - Use rfc2833 if offered, inband otherwise.
- **Insecure** - Insecure is a SIP parameter used to determine peer matching. The following options are valid:

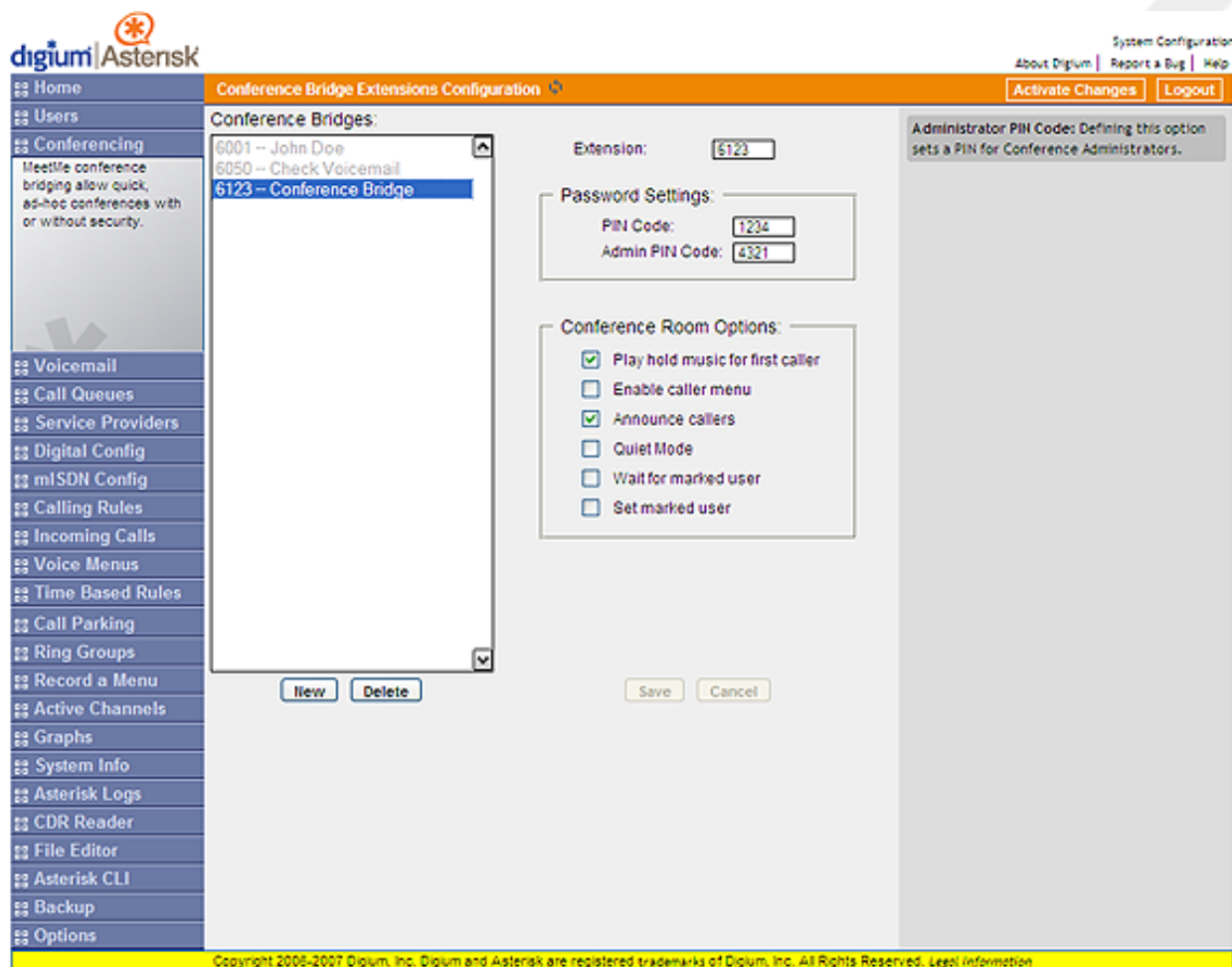
Enter Port for a peer without matching port, do not require authentication of invites. Options are: \* Port \* Invite \* port,invite

  - **Port** - Enter this value to match against only an IP address. This setting is useful if you have multiple endpoints behind a NAT device.
  - **Invite** - Enter this value to match against both the IP address and port number provided in the Contact field of the SIP header. A call will be allowed without authentication if a match is found.
  - **Invite, Port** - Specify this value if you do not want to require authentication upon an initial invite.

To add new extensions, select **New Entry** in the area under Extensions and then click New. The system defaults to four-digit extensions, beginning with the number 6000. Click **Edit Codecs** to select one or more codes for the currently selected user extension.

## Conferencing

Every company reaches the point of needing more people on a phone call than it can effectively include through three-way calling. Asterisk conference bridges allow you to include more people as well as project a professional image.

**Figure 5.2. Conferencing**

The configuration of the conference bridge and standard features is very straightforward. Select the New Entry option in the Extensions window and click New to design a conference bridge. The GUI auto-populates the extension with the next available extension in sequence, but you can always change it to any extension number you want. After establishing the extension for the bridge, you need to specify the password settings for the conference. Assign the PIN Code used by participants to enter the conference as well as the Administrator PIN Code used by the moderator of the conference to open the conference bridge.

Now that you have established the conference bridge extension and password codes, you can set your conference room options. The next three options are hospitality features which allow the caller to feel more welcome, as follows:

- **Play hold music for first caller** - Checking this option makes music play for the first caller entering a conference until another caller joins. Some people don't like sitting in a quiet room — even a virtual room — alone, and this feature prevents anyone from being in that position.
- **Enable caller menu** - This feature allows callers to access the Conference Bridge Menu by pressing the asterisk (\*) key.
- **Announce callers** - All new callers to a conference are identified when they arrive when this feature is selected.

The remaining configuration features provide some great functionality and heightened control over your conferences. If you are bringing together sales teams or vendors on a conference, it is preferable to keep them from chatting amongst themselves before the host arrives. These features allow you to handle the following requirements:

- **Quiet Mode** - You may choose this feature for a conference bridge with room override because it allows all users on the bridge to have listen-only access to the conference. Establishing two access points, with one group of people using the main extension while the other remains in quiet mode allows a controlled environment in which to deliver information while the second group listens.
- **Wait for Marked User** - This is a feature that keeps all participants in quiet mode until a special participant, using a unique extension, arrives. Only after the marked user arrives is the audio activated so that all of the participants can speak to each other.
- **Set Marked User** - This option works in conjunction with the Wait for Marked User feature. Checking Set Marked User makes the individual arriving from this extension the Marked User. If the CEO of the company, for example, doesn't want anyone chatting in the conference bridge until he or she arrives, these options are set to keep everything quiet. The main conference extension of 6003 is configured with Wait for Marked User selected. Everyone in the conference arriving from extension 6003 remains silent until the CEO arrives.

## Voicemail

Voicemail is an option available for every extension in Asterisk. The relationship between the extension and the voice mail is established in the User Extension section of the GUI. That section covers only the relationship between the extension and the voice mail but doesn't identify the parameters of the voice mail service itself. If you have not yet set up an extension for checking voicemail, you will be prompted to define a Check Voicemail extension the first time you access the **Voicemail Configuration** section.

**Figure 5.3. Voicemail**

The screenshot displays the AsteriskGUI interface for Voicemail Configuration. On the left is a sidebar with a tree view containing links to various system components. The main panel is titled 'Voicemail Configuration' and 'VoiceMail Settings'. It features a list of extensions on the left, with '6500 -- Check Voicemail' selected. To the right of this list are several configuration fields: 'Extension for checking messages' (set to 6500), 'Attach recordings to e-mail' (checked), 'Max greeting (seconds)' (set to 5), and 'Dial 0 for Operator' (checked). Below these are two sections: 'Message Options' and 'Playback Options'. 'Message Options' includes 'Attach Format' (WAV (GSM)), 'Maximum messages : (per folder)' (1), 'Max message time' (15 minutes), and 'Min message time' (1 second). 'Playback Options' includes checkboxes for 'Send messages by e-mail only', 'Say message Caller-ID', 'Say message duration', 'Play envelope', and 'Allow users to review'. At the bottom of the main panel are 'Save' and 'Cancel' buttons. On the far right, a note explains the 'Minimum message Time' setting. The footer contains copyright information for Digium, Inc.

The Voicemail Configuration page displays all the extensions to the left, including voice mail. Standard configuration information is also present, allowing you to confirm the extension used to check messages, as well as general parameters such as the following:

- **Extension for Checking Messages** - This option defines the extension which Users call in order to access their voicemail account.
- **Attach Recordings to E-Mail** - This option is used to choose whether voicemails are sent to the selected users e-mail address as attachments. Click the check box to enable this option.
- **Max Greeting (Seconds)** - With this option, you specify the maximum amount of time available to record your voicemail greeting.
- **Dial “0” for Operator** - Callers who are sent to voice mail can press “0” for the operator and be transferred either during the voice mail salutation, or after recording the message. If this option is not enabled, a caller’s pressing “0” will be ignored. There are several options that can be specified to define the voicemail message in the system.
- **Attach Format** - This option gives you the ability to choose the format in which messages are delivered by e-mail.
- **Maximum Messages per Folder** - The maximum number of messages per voice mail box is set here.
- **Maximum Message Time** - The maximum duration of a message left by a caller is set here.

- **Minimum Message Time** - The minimum duration of a message is dictated here. Any message left that's under the listed duration is discarded and isn't processed or retrievable.

There are several playback options that can be specified.

- **Send Messages by E-Mail Only** - You can choose to have e-mail be your only means of notification of voice mails left for you by this option.
- **Say Message Caller-ID** - The Say Message Caller ID option reads the caller ID before the voice mail message is played.
- **Say Message Duration** - This option identifies exactly how long the message lasted.
- **Play Envelope** - The envelope provides the date, time, and caller ID related to a voice mail.
- **Allow Users to Review** - This option provides incoming callers the option to review their message before it is saved and can be played back by the owner of the voice mail extension. Standard options are presented to you, allowing you to discard the message or re-record it if you aren't happy with it.

## Call Queues

A call queue lines up callers and allows them to wait to speak to any group of employees taking a high volume of calls. The feature allows you to speak to more people rather than send callers back to voice mail to leave a message and receive a call back when time permits.

Asterisk identifies which extensions under the Users tab are capable of belonging to a call queue by whether the Is Agent option is selected. The Is Agent indicates that the user is available to answer customer calls. If a check mark does not appear next to Is Agent, that extension won't appear in the list of agents in the configuration for this option. The following illustration shows the Queue Extension Configuration window displaying the available options.

Figure 5.4. Call Queues

The Queues section lists the existing queues. Existing queues are in black. To create a new queue, click New below the queue listing. Use the following steps to create a queue. Keep in mind the purpose of the queue and how it should operate.

1. The extension for the queue will automatically populate in the **Queue** field with the next available extension. If you want the number to be something other than the automatically chosen one, enter it in the Queue field.
2. Next, give the queue a name that will be meaningful. The queue will be referenced by this name, so be sure to make it sufficiently descriptive as well. For example, “Technical Support” for the technical support queue, “Sales”, and so on.
3. You now should choose the strategy used in your queue call logic. Using the Strategy drop-down list, choose one of the following options for routing calls:
  - **Ring All** - Rings every agent who isn't on an active call when a new call arrives. The first agent to answer the call receives it.
  - **Round Robin** - Every available agent receives a call in turn, akin to how cards are dealt in a poker game.
  - **Least Recent** - The agent who has been without a call the longest receives the next call.
  - **Fewest Calls** - The agent who has handled the fewest calls receives the next incoming call.

- **Random** - Goes by the luck of the draw; any agent can receive the next incoming call.
  - **RrMemory** - This option is Round Robin with Memory. It's similar to Round Robin, but smarter — it remembers over the course of days, weeks, or years which agent received the last call so that it can commence with the next agent in sequence when calls begin again.
4. The Agents box lists all Users that are designated as an agent that can receive calls as part of a call queue. All users listed have the Is Agent checkbox selected on their user profile. Many Users may be listed as potential agents, but some may be assigned to a sales queue and some for a service queue. This box lists all agents and allows you to choose which users you assign to the queue.

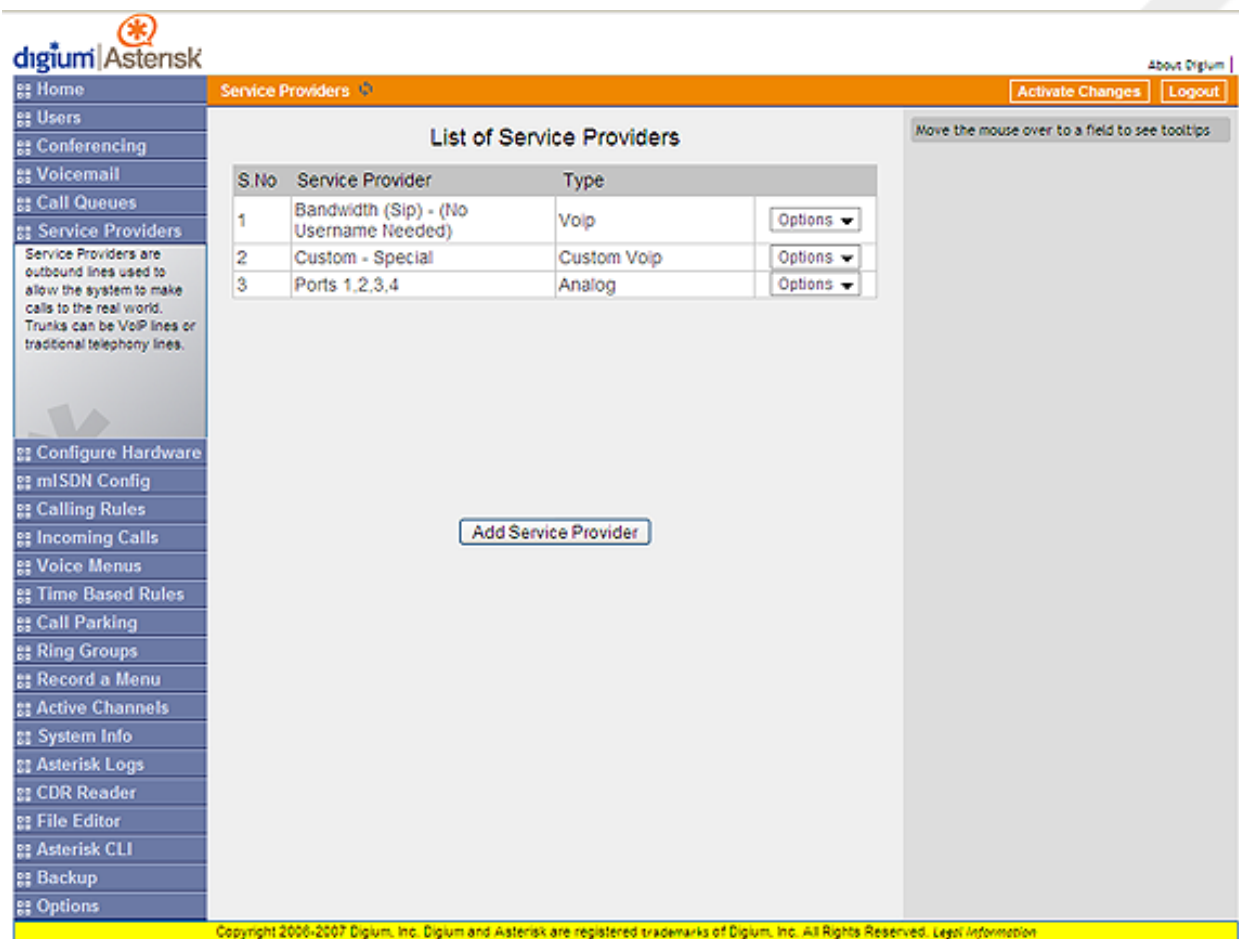
The options available in the Queue Options section control the timing and managing of the calls as well as the agents. You may not want to work with these finer points of call queuing until after your call queue has been working for a while and you have an idea of call volume and the turnover of calls by each agent. Here's a list of the available options:

- **Timeout** - The default for this option is 15, representing 15 seconds that an agent's phone will ring before the call is forwarded on to another agent.
- **Wrapup Time** - This is a buffer of time allowing your agents to finish work on one call and remain unavailable in the queue. The default on this option is 0 seconds, providing no buffer time for your agents and allowing the next call to ring through immediately after a call is complete.
- **Max Len** - This option sets the maximum number of callers allowed in the queue before they are sent to voice mail or receive a busy signal. The default is "0," which allows for an unlimited number of calls in queue before they are sent elsewhere.
- **Music on Hold** - This option gives you the ability to select the music played while a call is on hold.
- **Auto Fill** - This option is enabled by default and allows multiple calls that arrive at the same time to be immediately forwarded on to agents.
- **Auto Pause** - If an agent fails to answer a call, this option temporarily postpones sending calls to that agent.
- **Join Empty** - This option allows callers to enter a queue even if no agents are logged into it. If this option is not checked, callers cannot enter a queue until at least one agent is present.
- **Leave When Empty** - This option mirrors the Join Empty, but it represents a queue in which agents had been logged in but are now gone. At 5:00 pm, when your employees go home, you can program the queue to shut down when the agents log out. The existing callers in queue are forced to exit, and no new callers are granted access to the queue.
- **Report Hold Time** - The Report Hold Time tells the agent how long the call was holding in queue before it was sent to the agent. If the hold time was short, the agent will probably be happy to accept the call. If the hold time was 10, 15, or 20 minutes, the agent might want to brace for a frustrated customer, but at least the agent isn't overwhelmed.

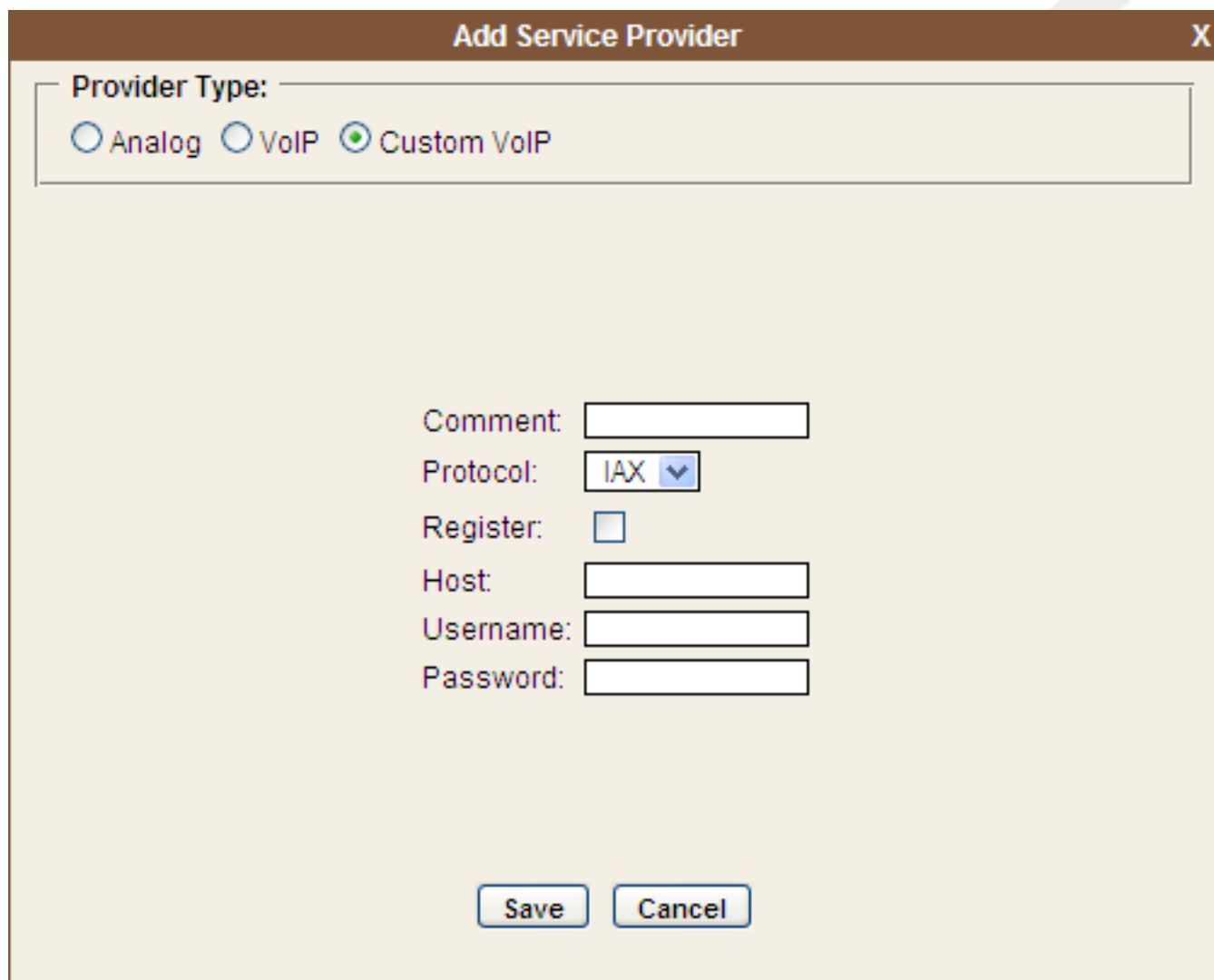
## Adding Service Providers

You must register with a service provider in order to connect to the Public Switched Telephone Network (PSTN). Access to the PSTN gives you the ability to place calls to telephone numbers no matter how they connect to the PSTN (VoIP or standard analog system). Asterisk identifies the carriers used to reach your local phone carrier, long-distance carrier, or VoIP provider as service providers. The service providers may use analog lines for your local carrier or IP connections for your VoIP carriers.



**Figure 5.5. Service Providers**

There are three service provider categories: Analog, VoIP, and Custom VoIP. If you have not yet added a service provider, a message stating that you have not added a provider will be displayed in the Service Providers window. To add a new provider, click Add Service Provider. The Add Service Provider dialog box will be displayed similar to the following illustration.

**Figure 5.6. Adding a Service Provider**

**Add Service Provider** X

**Provider Type:**

☐ Analog ☐ VoIP ☒ Custom VoIP

Comment:

Protocol:  ▼

Register: ☐

Host:

Username:

Password:

Select the Provider Type depending on your needs. If you wish to use a combination of both analog and VoIP, you will need to add a service provider definition for both.

## Analog Providers

If you need to make calls using an analog telephone, you will need to add one or more analog ports as a service provider. Use the following procedure to add an analog service provider.



### Note

Before you follow this procedure, ensure that your analog lines are connected to the FXO ports. Failure to do so will lead to improper configuration of your analog lines.

1. Click the Analog radial button in the Add Service Provider window. A list of the available analog ports will be displayed. If an analog port is displayed, but grayed out, it is already part of a service provider definition.
2. Select one or more ports as part of your service provider definition and then click Save.

3. Click Activate Changes to apply the service provider definition(s) you have added.

You may discover, after using your analog port(s), that the volume may need adjustment. You may also wish to re-calibrate the port. Click either Calibrate or Audio Levels from the Options drop-down list to make the desired adjustments.

## VoIP Providers

The AsteriskGUI comes pre-loaded with a selection of certified VoIP service providers. If you are already a VoIP provider customer, select the provider from the list and input your user name and password. If you are a new customer, click on the provider logo to establish a new account. Asterisk will confirm your account when you click the Save button.

## Custom VoIP

The Custom VoIP option allows you to create a custom VoIP definition. To create the custom VoIP provider definition you will need to complete the following Custom VoIP fields:

- **Comment** - The comment field should be used as the name of the custom VoIP definition.
- **Protocol** - Specify either a IAX or SIP protocol.
- **Register** - Click the Register checkbox if you need to register your IP address with your service provider. Registering is not required for all providers.
- **Host** - The IP address of your service provider.
- **Username** - The user name associated with your provider account.
- **Password** - The password associated with your provider account.

Click Save to retain your Custom VoIP definition, or Cancel to discard your changes.

Once you have added the service providers needed, click Activate Changes to save the configuration.

When you have added a service provider it will appear in the Service Providers list. There is an Options drop-down list associated with each Service Provider listing. The Options drop-down list allows you to edit or delete the Service Provider definition, as well as further refine the definition by choosing several advance options. Select either Codecs or Advanced to further refine the definition.

## Codecs

Codecs provide the ability for your voice to be converted to a digital signal and transmitted across the Internet. The quality of your call can be affected by the choice you make. The codecs available to you will depend on what is supported by the service provider you choose. All available codecs are allowed by default. Select a codec and transfer it to the Disallowed list if you do not want to use that codec. Select the Disallow All checkbox if you do not want to use any of the listed codecs. Click Update to retain your changes, or Cancel to discard them.

## Advanced

The following advanced options are available to further refine your service provider definition.

- **Trunkname** - Specify a trunk name if you want to refer to the service provider definition as something other than specified in Comment.
- **Insecure** - This option specifies how connections to a service provider (host) should be handled. Valid options are very|yes|no|invite|port. The default is no (authenticate all connections).

- **Port** - The register request is sent through the port specified here to the service provider (host). Defaults to 5060.
- **Caller ID** - The caller ID will be set to the value specified in this field. – **Fromdomain** - Sets default From: domain in SIP messages when acting as a SIP ua (client).
- **Fromdomain** - This allows you to set the domain in the From: field of the SIP header. It may be required by some providers for authentication.
- **Fromuser** - Sets default From: user in SIP messages when authenticating.
- **Contact** - Specifies a primary extension for call routing.
- **Can Reinvite** - The SIP protocol tries to connect endpoints directly. However, Asterisk must remain in the transmission path between the endpoints if it is required to detect DTMF.

Click Update to retain your changes, or Cancel to discard them.

## Configure Hardware

The Configure Hardware page gives you the ability to ensure that your telephony hardware is configured and working properly. Click the Configure Hardware tab and the Digital Card Configuration Page will be displayed similar to the figure below:

**Figure 5.7. Card Configuration Page**

The screenshot displays the Asterisk GUI's 'Digital Card Configuration Wizard'. The sidebar on the left contains a list of navigation links: Home, Users, Conferencing, Voicemail, Call Queues, Service Providers, Configure Hardware (selected), mISDN Config, Calling Rules, Incoming Calls, Voice Menus, Time Based Rules, Call Parking, Ring Groups, Record a Menu, Active Channels, System Info, Asterisk Logs, CDR Reader, File Editor, Asterisk CLI, Backup, and Options. The main content area is titled 'Digital Card Configuration Wizard' and features a 'Digital Hardware' section with a table of hardware configurations. Below this is a 'LoadZone' dropdown menu set to 'us'. The 'Analog Hardware' section lists 'FXS Ports' (29,30,31,32) and 'FXO Ports' (25,26,27,28). At the bottom of the main area are 'Apply Changes' and 'Cancel Changes' buttons. The footer contains copyright information for Digium, Inc. (2006-2007).

SPAN	ALARMS	Framing/Coding	channels Used/Total	Signalling	
Wildcard TE122 Card 0	OK	ESF/B8ZS	1/24	pri_net	<a href="#">Edit</a>

LoadZone :

**Analog Hardware**  
 FXS Ports : 29,30,31,32  
 FXO Ports : 25,26,27,28

[Apply Changes](#) [Cancel Changes](#)

Copyright 2006-2007 Digium, Inc. Digium and Asterisk are registered trademarks of Digium, Inc. All Rights Reserved. [Legal Information](#)

The page shows information about both digital and analog hardware installed on your system. In the example above, a Digium TE122B card is listed under **Digital Hardware**, and the ports from a TDM800B are listed under **Analog Hardware**.

ED

## Digital Hardware

The Digital Hardware section lists each T1, E1, or J1 card that is currently part of your system. The following information is listed for each card:

- **Span** - The Span column shows the type of digital card.
- **Alarms** - The Alarm column lists alarms, if any, that indicate the card performance. If the card is performing as expected, a status of OK is listed. If a card is not working properly, an alarm of RED listed.
- **Framing/Coding** - This column lists the type of framing (bandwidth division into channels) and coding (method of signal encoding).
- **Channels Used/Total** - This column list the number of channels used by the card, and the number of channels available on the card. In the example above, only 1 of 24 channels is used.
- **Signalling** - The Signalling column lists the type of signalling that the T1, E1, or J1 card is using.

The **LoadZone** drop-down list lets you specify the set of tones that should be used with your digital card.

The **Edit** button listed for each gives you the ability to edit several options associated with each card's performance. Click the Edit button and a dialog for the card selected will be displayed similar to the following:

**Figure 5.8. Card Edit Dialog**

**SPAN : Wildcard TE122 Card 0** [X]

ALARMS: OK

Framing/Coding:

Channels: undefined/24 (T1)

Signalling:

Switch Type:

Sync/Clock Source:

Line Build Out:

Channels: Use :

From : 1

Reserved : 24

The Edit Card Dialog gives you the ability to edit the following options:

- **Framing/Coding** - Select the type of channel and encoding you would like to use. The two options available are:
  - ESF/B8ZS - Extended Super Frame framing and Bipolar with 8 Zeros Substitution encoding.
  - D4/AMI -
- **Signalling** - The signalling options available are:
  - **PRI - CPE** -
  - **PRI - NET** -
  - **FXOKS** - FXO Kewl Start
  - **FXOLS** - FXO Loop Start
- **Switch Type** - The switch type options available are:
  - National ISDN 2 - This is the default switch type.
  - Nortel DMS 100 -

- AT&T 4ESS -
- Lucent 5ESS -
- Euro ISDN -
- Old National ISDN 1 -
- Q.SIG -
- **Sync/Clock Source - The following sync/clock source options are available:**
  - Master 0 -
  - Slave 1 -
- **Line Build Out - The following line build out options are available:**
  - 0db (CSU)/0-133 feet (DSX1)
  - 133-266 feet (DSX1)
  - 266-399 (DSX1)
  - 399-533 (DSX1)
  - 533-655 (DSX1)
  - -7.5db (CSU)
  - -15db (CSU)
  - -22.5db (CSU)
- **Channels - The Channels option lets you specify the number of channels you need to use.**

Click **Update** on the Card Edit Dialog when you have completed your choice of card options.

## Analog Hardware

The **Analog Hardware** section lists the FXS and FXO ports available on the analog cards installed in your system.

## mISDN Config

The mISDN Config page is used to configure a BRI (Basic Rate Interface) ISDN card, such as Digium's B410P, as well as define and configure ISDN service providers which the card will utilize. Click on the mISDN Config tab to access the page.

**Figure 5.9. mISDN Configuration**

There are two sections within the mISDN configuration page: the BRI card section, and the service provider section.

## BRI Cards

The BRI Card section lists the card(s) currently installed on your system as well as the number of ports associated with the card. A table lists each port for the card installed. You can set the mode for each port as well as timing. The ISDN mode options available are:

### TE Mode

TE (Terminal Equipment) mode sets the port to act as an ISDN endpoint connected to an ISDN device; in most cases a phone. If the port is set to PTP, only one device can be connected. More than one device can be connected to the ISDN port if PTMP is chosen. There are four options available for TE mode.

- PTP - Point to Point mode. One device will handle all calls.
- PTMP - Point to Many Point allows the connection of multiple devices to a single port.
- PTP (CAPI) - PTP mode using the CAPI driver.
- PTMP (CAPI) - PTMP Mode using the CAPI driver.



## NT Mode

NT (Network Terminator) mode should be used when a port is acting as an ISDN originator, or Telco. The port set to NT mode will act as the synchronization/timing Master, and all devices connected to it will act as Slaves.

- **PTP** - Point to Point mode. One device will handle all calls.
- **PTMP** - Point to Many Point allows the connection of multiple devices to a single port.

## Timing

Timing is set at the same that TE or NT mode is set. Timing can be set to either Master, which sets the timing for other devices, or Slave, which receives the timing from the Master. If your port is set to NT mode it should act as the timing Master. If your port is set to TE mode, it should be set to receive timing synchronization (Slave).

## mISDN Service Providers

The mISDN Service Providers section is used to define a provider and the port used by the provider. To create a provider definition click Add. A Service Provider dialog will be displayed. There are two fields which comprise a service provider definition; Trunk Name, and Ports.

- **TrunkName** - The name of the service provider associated with one or more ports.
- **Ports** - Specify the port(s) which should be associated with the trunk. Each port number is comma separated.

Click **Update** in the Service Provider dialog when you have completed the service provider definition. The service provider definition will be added to the mISDN Config page.

## Calling Rules

The Calling Rules tab on the left toolbar allows you to use basic pattern matching to differentiate outbound calls and route them accordingly.

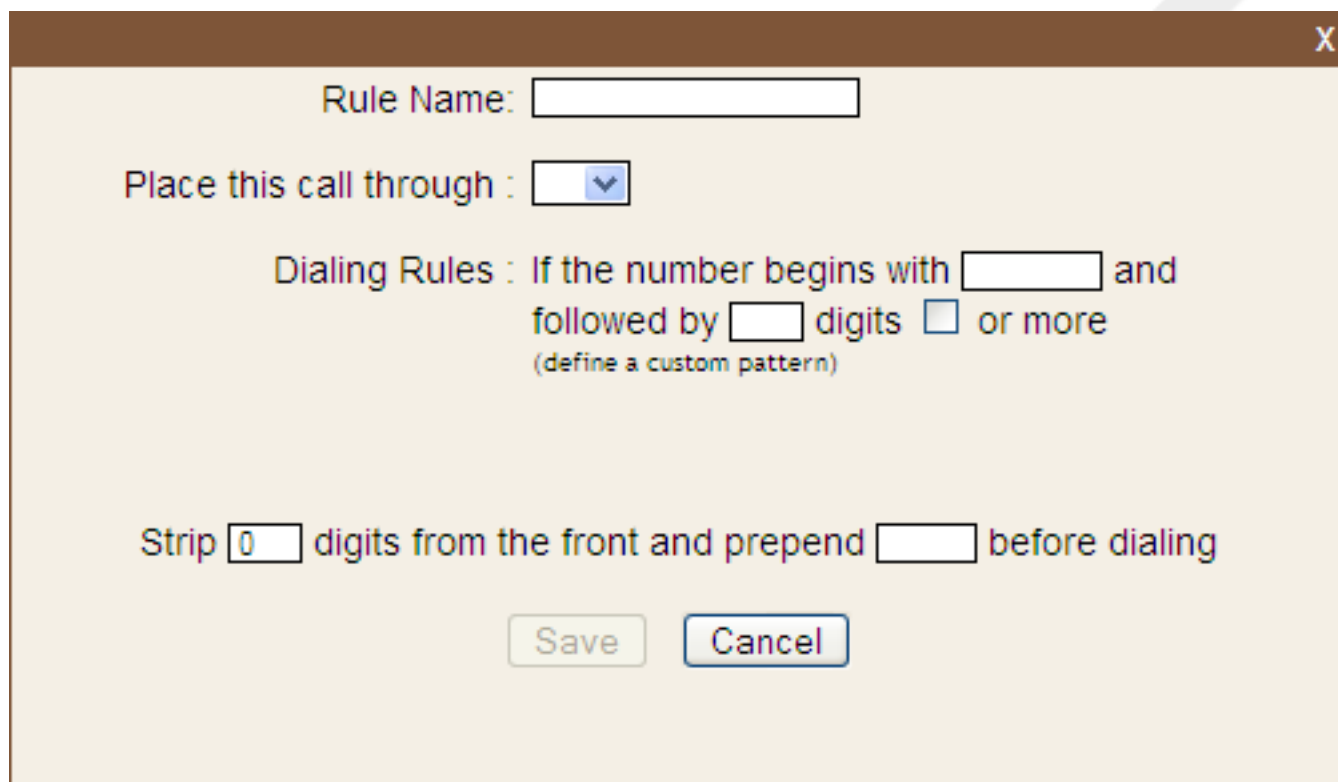
**Figure 5.10. Calling Rules**

The screenshot displays the AsteriskGUI interface for managing Calling Rules. The left sidebar contains a list of navigation options, with 'Calling Rules' selected. The main content area is titled 'Calling Rules' and includes a 'List of DiaPlans' section with a dropdown menu set to 'DiaPlan1', buttons for 'new' and 'delete', and a checkbox for 'Allow Parked Calls'. Below this is a 'List of Calling Rules in the selected DiaPlan' section containing a table with three rules. At the bottom of the main area is an 'Add a Calling Rule' button. The right sidebar shows an example rule: 'Ex: Strip 1 digits from the front and pre 256 before dialing'. The footer contains copyright information for Digium, Inc.

S.No	RuleName	Dial Pattern	Call Using	Options
1	Long Distance	Begins with 91 and followed by 11 or more digits	Span 1	Edit Delete
2	Local	Begins with 9256 and followed by 7 digits	Port 25	Edit Delete
3	911	Begins with 911 and followed by 0 or more digits	Port 25	Edit Delete

Copyright 2006-2007 Digium, Inc. Digium and Asterisk are registered trademarks of Digium, Inc. All Rights Reserved. Legal information

The Calling Rules menu shows every rule name established, a brief description of the rule name, as well as the port or span used to complete the call. Click on Add a Calling Rule to define a new calling rule. The following dialog will be displayed.

**Figure 5.11. Add a Calling Rule**

Rule Name:

Place this call through :

Dialing Rules : If the number begins with  and followed by  digits  or more  
(define a custom pattern)

Strip  digits from the front and prepend  before dialing

A calling rule is comprised of the following items:

- **Rule Name** - Choose a name that describes the type of rule you are creating.
- **Place this Call Through** - Select a service provider through which the call should be made.
- **Dialing Rules** - The Dialing Rule gives you the ability to use basic pattern matching to differentiate calls and route them accordingly. For instance, if a number begins with 9256, and is followed by 7 or more digits, that would define a call within the state of Alabama. If a call began with 9 followed by 7 digits, it would be a local call that probably didn't require a long distance charge. Instead of adding a rule for every extension or phone number you call, specify the pattern in this rule similar to the example.
- **Strip** - This option gives you the opportunity to remove specified digits from the call being dialed and replace them with the digits needed to make the call. You can also Prepend digits to the beginning.



### Note

You can call international destinations from North America without dialing an 011 prefix. Using a Calling Rule that restricts 011 calling prevents the extension from reaching Africa, Europe, Asia, Oceania, and South and Central America. This won't block all calls outside the United States however. Canada, the U.S. Virgin Islands, Guam, Saipan, and Puerto Rico, as well as a handful of Caribbean countries, are all a part of the North American Dialing plan and can be reached by dialing 1 + the area code and a seven-digit phone number. If you want to block these, it may be best to block all long-distance calls from the extension.

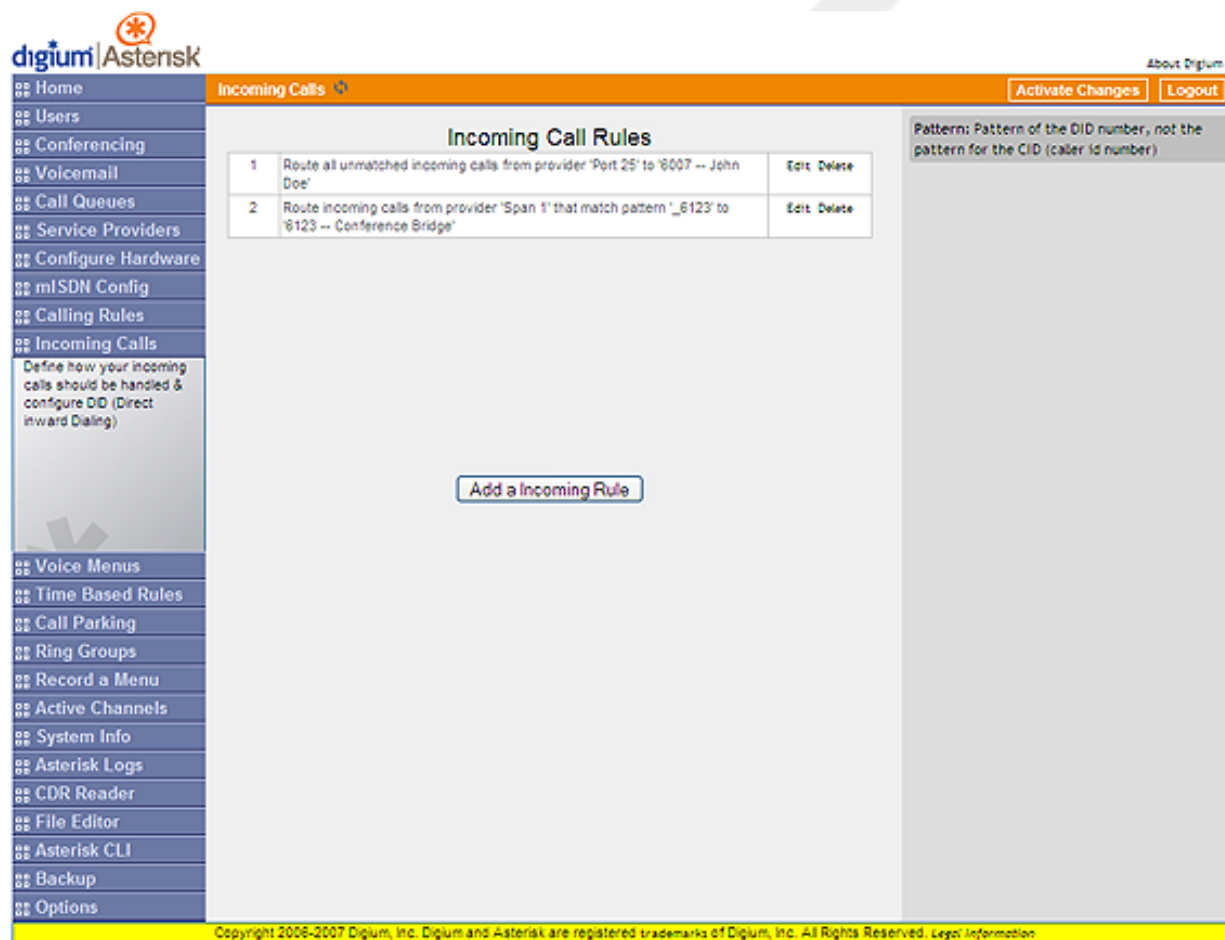
Once you have completed the calling rule definition click Save to accept the rule or Cancel to abandon your changes. Click Activate Changes in the upper right corner of the page to save and apply your changes.

The default dial plan, the collection of your calling rules, is DialPlan1. You can create more than one dial plan, especially if you want to have different dial plans for different user extensions. Click **New** at the top of the Calling Rules page and create a new dial plan name. You can then add calling rules for that dial plan definition. Select the **Allow Parked Calls** checkbox to access the call parking lot from this selected dial plan.

## Incoming Calls

The same pattern-matching logic used for processing outbound calls can also be employed for inbound calls. The two defaults define routing based on whether an incoming call matches or doesn't match a pattern you define.

**Figure 5.12. Incoming Calls**



Some example incoming rules are shown in the figure above. Click **Add a Incoming Rule** to create a new incoming rule. There are only a few options you will need to define.

- **Route** - Make a selection from the drop-down list to choose how the calls will be routed. You can select from All Unmatched Calls or Calls Which Match.
- **From Provider** - Select from the list of providers which you previously defined.
- **To Extension** - The previously defined extension which should receive the call.

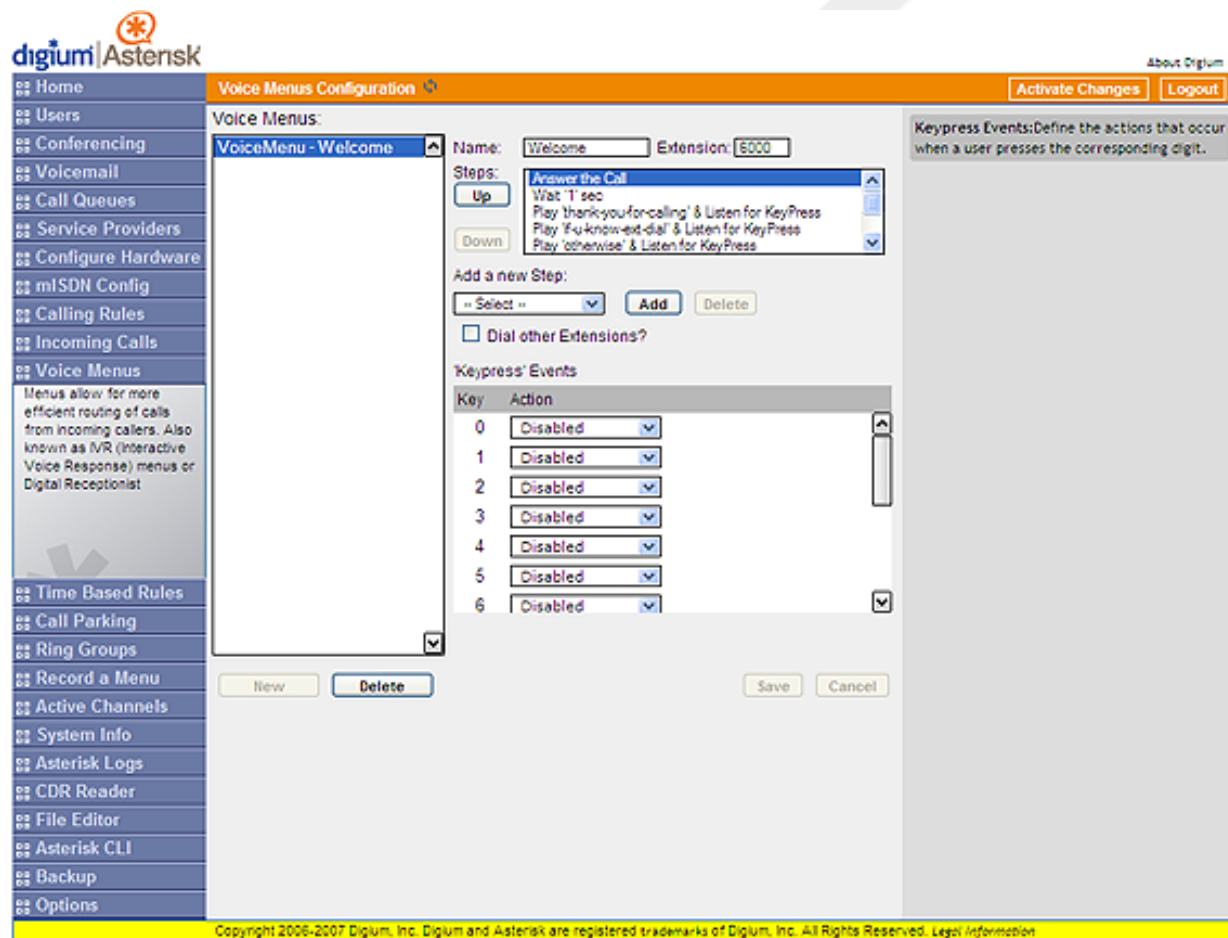
Once you have completed the definition of your incoming call rules, click Save. Click Activate Changes to save and apply your changes.

# Voice Menus

A valuable feature of Asterisk is the ability to create Interactive Voice Response (IVR) or voice menus. Voice menus are designed to allow for more efficient routing of calls from incoming callers. The menus provide a caller with specific instructions, receive responses from the caller, and process those responses into an action.

Each Business Edition system ships with a default voice menu already created. To better understand the creation and operation of these menus, we will examine the default one.

**Figure 5.13. Voice Menus**



Voice menus are constructed depending on your needs. Just like your business you need to create the solution best suited to your customers. The best way to understand how a voice menu is constructed is to examine the default “Welcome” menu provided with Asterisk Business Edition. Click **Voice Menus - Welcome** in the Voice Menus list. The options for the welcome menu are displayed similar to the example shown in the above illustration. The Welcome menu consists of the following steps:

```
Answer the Call
Wait '1' Sec
Play 'thank-you-for-calling & Listen for KeyPress
Play 'if-u-know-ext-dial' & Listen for KeyPress
Play 'otherwise' & Listen for KeyPress
```

```
Play 'to-reach-operator' & Listen for KeyPress  
Play 'pls-hold-while-try' & Listen for KeyPress  
WaitExten '6' Sec
```

In the example, when a caller dials your company number ending in 6000, the call is answered, and after a pause of one second the caller is greeted in the following manner: “Thank you for calling. If you know your party’s extension, please dial it now. Otherwise to reach an operator please dial 0.” If the caller tries an extension, the menu will respond with “Please wait while I try that extension.” If no action is taken by the caller, the menu will repeat after 6 seconds. This is an example of a basic voice menu.

## Voice Menu Options

In the example, each action is a step chosen from the list of available menu options. The available menu options are as follows:

- **Answer** - This step is automatically added when creating a new menu. This step answers the incoming call.
- **Authenticate** - The Authenticate step is used to restrict access one or more areas of your system. This is useful when one wants users to have to enter a PIN code in order to proceed to a particular part of the current menu, to a different menu, or to ring an extension.
- **Background** - This step is used to play an audio file in the background while waiting for the caller to enter an extension or number. Playback is terminated once the user begins to enter an extension. To select a file to play, click and hold in the field next to the Background choice to scroll through a list of pre-recorded sound files. In the example above, “Play ‘otherwise’ & Listen for KeyPress” is an example of using the Background option.
- **Busytone** - The Busytone option should be selected if there is a step in the process in which you want to play a busy signal to the caller. For instance if the call is over.
- **Congestion** - The Congestion option is similar to the Busytone option. A congestion tone will be played to the caller, should be selected if there is a step in the process in which you want to play a busy signal to the caller. For instance if the call is over.
- **Digit Timeout** - The Digit Timeout option is used to set the maximum amount of time allowed between key presses. If a full extension is not entered in the specified time, the entry will be considered invalid. A field for entering the number of seconds before timeout appears next to the option.
- **DISA** - The DISA option allows callers from outside the system to get access an internal dial tone and place calls from within your internal system. A passcode is required.



### Note

Use caution when choosing this option. This option can pose a security risk.

- **Response Timeout** - If a caller does not enter a response with the time specified in this field, the call will terminate. This step could be put at the end of a series of menu choices.
- **Playback** - The Playback option is similar to the Background option because it will play a sound file you select. However, this option does not listen for a KeyPress event, and will move on to the next step in your list.
- **Wait** - The Wait option pauses the execution of steps in the voice menu list for the number of seconds you specify.
- **WaitExten** - The WaitExten option is specified to give a caller a specified amount of time to enter an extension.

- **Goto Menu** - The Goto Menu option sends a caller to one of the voice menus that you specify.
- **Goto Directory** - The Goto Directory option sends a caller to the system phone directory. This gives the user the chance to select a user name from the directory if the extension is unknown.
- **Goto Extension** - The Goto Extension option sends a caller to a specified extension. Select the extension from the available list.
- **Dial RingGroup** - This option will dial a specified RingGroup. For example, if your menu says “Press 1 for Technical Support”, the Technical Support ring group will be dialed.
- **Hangup** - The Hangup option terminates the call.

## Creating a Voice Menu

Use the following procedure as a guide to creating a voice menu.

1. On the Voice Menu page, click New to create a new voice menu.
2. Specify a Name and an Extension. The extension will be the direct dial to the voice menu.
3. Specify the Steps of your voice menu using the welcome menu example and step descriptions as guides.
4. Select the Dial Other Extensions checkbox if you want to give a user the ability to break out of the menu selections and dial an extension within your system.



### Warning

The Dial Other Extensions option is important. This option allows an inbound caller to break out of the listed Keypress Events and dial another extension. A malicious person may be able to hack through your Asterisk implementation to find an outside dial tone and use it for fraud. Any extensions that are known to the public should be completely handled by the Keypress Events; callers should not be allowed to dial other extensions. Sticking to this policy protects your Asterisk system from being compromised.

5. Specify the Keypress Event actions for digits 0-9 as well as \*, #, t, and i. The options available for a Keypress Event are:
  - **Disabled** - The associated key is not enabled.
  - **Goto Menu** - Pressing a key with this option will send the caller to a specified menu.
  - **Goto Extension** - Pressing a key with this option will send the caller to a specified extension.
  - **Custom** - You can define your own keypress event.
  - **Hangup** - Pressing a key with this option will terminate the call.
  - **Play Invalid** - Pressing a key with this option will tell the caller that they have made an invalid entry.

Both the t key and i key should be used for specific actions. The action associated with the t key should be the desired action if a user response has timed-out. The action associated with the i key should be the desired action if a user makes an invalid entry.

6. Once you have constructed your voice menu, click Save. You can then click Activate Changes to add the voice menu to your current configuration. If you are happy with the voice menu, click Save Configuration on the Home page.

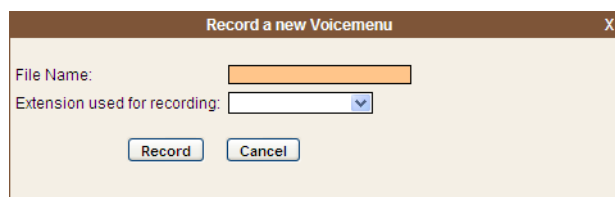


## Record A Menu

In the event that you want to record custom menu prompts for Asterisk, which can be used in a voice menu, the Record a Menu tab may be used. A list of previously recorded menus is displayed. Here, the user may modify several options:

- **Record Again** - Clicking this button allows the user to make another attempt at recording and replacing an existing custom sound file.
- **Play** - Clicking this button brings up a dialog entry box to allow the input of an extension that Asterisk will dial and play the prompt over.
- **Delete** - Clicking this button will delete the selected prompt.

**Figure 5.14. Record a New Voice Menu**



There are three options under "Record a new voice menu":

- **File Name** - This text entry box specifies the saved name of the file that is to be recorded.
- **Extension Used for Recording** - This drop-down select box allows the user to choose which extension Asterisk will dial to wait for the user to speak the prompt.
- **Record** - Clicking this button causes Asterisk to launch the call that will record a file.

## Time Based Rules

Time based rules should be created when you want to treat incoming calls received at specific times differently than typical calls. The most obvious use for this conditional rule would be routing calls to voice mail after or before business hours. You could also create time based rules which will direct incoming calls to specialized messages for holidays. Click the **Time Based Rules** tab to access your list of time based rules. If you have not created a time based rule, click **New Time Rule** at the top of the page to create a new rule.



**Figure 5.15. New Time Based Rule**

**Add new Time Rule** X

Rule Name :  (Ex: July4)

**Time & Date Conditions**

Start Time:  :  End Time:  :

Start Day:  End Day:

Start Date:  End Date:

Start Month:  End Month:

**Destination**

if time matches:

if time did not match:

A time based rule uses the `GotoIfTime()` application within Asterisk to match time and date conditions specified within your rule to the current time. If the current time matches the rule, calls can be routed to an IVR (voice mail) or a department or individual's extension. Use the following field descriptions when creating a time based rule.

- **Rule Name** - Choose a name that describes the type of rule you are creating. For example, "Closed" could be used as the name for a rule which routes calls received before or after business hours.
- **Start/End Time** - Specify the time of day, in 24-hour format, in which this rule should be used.
- **Start/End Day** - Use this condition to specify a day of the week.
- **Start/End Date** - Specify the days of the month in which this rule should be active. For example, select 24 and 25 if you want to specify Christmas holidays. Do not select a day of the month if the rule should be active all month.
- **Start/End Month** - Use this condition to specify the month(s) during which this rule should be active.
- **If Time Matches** - Specify the voice menu or extension this call should be routed to if the time conditions specified are met.
- **If Time Did Not Match** - Specify the voice menu or extension this call should be routed to if the time conditions specified do not match.

Once you have completed the rule definition click Save to accept the rule or Cancel to abandon your changes. Click Activate Changes in the upper right corner of the page to save and apply your changes. Once your rule is saved it

will appear in the list of time based rules. Click Edit next to a rule in the list to edit a rule, or Delete to delete the rule from the list.

## Call Parking

Call parking is an Asterisk feature which allows a user to place a call on hold so that it can be taken off hold from another extension. The Call Parking page gives you the ability to define the call parking options which will enable use of this feature.

**Figure 5.16. Call Parking**

The screenshot shows the Asterisk GUI interface for configuring call parking. On the left is a sidebar menu with options like Home, Users, Conferencing, Voicemail, Call Queues, Service Providers, Configure Hardware, miSDN Config, Calling Rules, Incoming Calls, Voice Menus, Time Based Rules, Call Parking (selected), Ring Groups, Record a Menu, Active Channels, System Info, Asterisk Logs, CDR Reader, File Editor, Asterisk CLI, Backup, and Options. The main content area is titled 'Call Parking Preferences' and contains three input fields: 'Extension to Dial for Parking Calls' with the value '700', 'What extensions to park calls on' with the value '701-720' (with an example '(Ex: '701-720')'), and 'Number of seconds a call can be parked for' with the value '45'. Below these fields are 'Save' and 'Cancel' buttons. On the right side of the main area, there is a section titled 'Add a new step' with a description 'Add additional steps performed during the menu.' and a plus icon. At the top right of the main area are 'Activate Changes' and 'Logout' buttons. The footer of the page contains a copyright notice: 'Copyright 2006-2007 Digium, Inc. Digium and Asterisk are registered trademarks of Digium, Inc. All Rights Reserved. Legal information'.

The following options must be configured to enable call parking.

- **Extension to Dial for Parking Calls** - Specify the extension to call when transferring a call to hold or the “parking lot”.
- **What Extensions to Park Calls On** - The extensions specified here will be the “parking lot” designations for the calls you place on hold. The call on hold will be retrieved by dialing one of these extensions.
- **Number of Seconds a Call Can Be Parked** - The number of seconds a call can be placed on hold. After the time has elapsed the call will ring the originating extension.

## Parking A Call

You can park a call using either an analog or VoIP phone. To use an analog phone, hit the flash button, or quickly click the hook switch, wait for a dial tone, then dial the extension (700). With a VoIP phone, initiate the transfer, dial the call parking extension (i.e. 700), then complete the transfer (such as by click send). The method using a VoIP phone will vary depending on the phone.

At this point, the system will prompt you with a number. The number it prompts you with is the number from the pool specified. This is the number that can be entered to retrieve the call. To retrieve the call, pickup a phone, and dial the parking number specified. The amount of time that the call remains parked is determined by the number of seconds specified. If the call is not retrieved in this time, the call will be redirected to the user that originally parked the call.



### Note

In order to properly park a call, you must use attended transfer functions. Using a blind transfer function will not provide the parking number to the person parking the call. This makes recovery of the call impossible, except for the fall through timeout.

## Ring Groups

Ring groups allow a group of phones, or devices, to ring simultaneously or in sequence (ring order). This provides the opportunity for multiple people to answer a call (ring all) or one person can answer a call from any phone. Asterisk Business Edition does not come with a default ring group. To create a new ring group click New Ring Group at the top of the Ring Groups page.

**Figure 5.17. Ring Groups**

**Add Ring Group** X

Name:

Strategy: Ring all ▾

←

→

»»

SIP/6000 -- New User

SIP/6001 -- New User

IAX2/6001 -- New User

Zap/1

Zap/2

Zap/3

Zap/4

**Ring Group Members** **Available Channels**

Extension for this ring group (optional) :

Ring (each/all) for these many seconds :

If not answered

☐ Goto Voicemail of this user

☐ Goto an IVR menu

☒ HangUp

**Note**

You will not be able to define a ring group without user extensions or trunks defined.

## Creating A Ring Group

To create a ring group, use the following procedure.

1. Define the Name of the group. The name can be any mnemonic such as Sales or Technical Support.
2. Choose a ring group strategy from the Strategy drop-down list. You can choose either Ring All which will ring all phones in the defined group simultaneously, or Ring Order which will ring phones in sequence determined by the order of the users or trunks in the group.

3. Choose the members of the ring group from the Available Channels list. Click on a user extension or trunk, and then click the arrow pointed at the Ring Group Members list to transfer. Select a user extension or trunk in the Ring Group Members list to and then click the arrow pointing toward Available Channels to transfer the selected item back to the list. Click the double arrow symbol to transfer all group members back to the Available Channels list.
4. Specify an extension to associate with the ring group. This is the extension that can be dialed to ring all members of the group simultaneously or in order of listing.
5. Specify the number of seconds that each phone (or all phones) should ring before either ringing the next phone in order.
6. Lastly, determine what action you want the system to take if no one answers the call. You can either direct the call to the voicemail of the first user, go to an IVR menu, or end the call.

## Asterisk Management Options

There are several administrative tabs which are used to manage your Asterisk server and the GUI interface. You may use them often or not at all, depending on how much interaction you have with your Asterisk implementation. The management selections are as follows:

- **Active Channels** - This tab provides you with a remote view of the active calls and devices. It displays a snapshot of the activity of the server and can be refreshed to view the progression of calls.
- **System Info** - The general system information of your Asterisk implementation is displayed from this tab, as well as tabs for your ifconfig, disk partitioning resources, and IP logs.
- **Backup** - This is a housekeeping tab which allows you to back up your system configuration. To create a backup, click Take a Backup and then specify a file name (i.e. the backup date).
- **Asterisk Logs** - This tab provides the text of all Asterisk log messages for the current day. Log messages from another can be displayed by specifying the date.
- **CDR Reader** - This tab displays the call detail records for calls made through the system.
- **File Editor** - The File Editor page lets you edit any Asterisk configuration file within the GUI, as well as create a new configuration file.
- **Asterisk CLI** - The Asterisk CLI is a command line interface which can be used for issuing any Asterisk command or series of commands. The results of the commands are displayed in the pane above the command line field. Enter Help in the command line field for a list of commands.
- **Options** - The options tab provides several options which allow you to change the password for your AsteriskGUI logon, modify local extension and agent settings, as well as utilize the Setup Wizard. The Advanced/Basic tab allows you to enable or disable advanced options. The basic options are displayed by default. Please refer to the Advanced Options section for a description of the advanced options.

## Advanced Options

There are several advanced options available from the Options page which give advanced users with a background in Asterisk the ability to refine your Asterisk implementation. Clicking the Options tab will display some advanced administration settings.

**Figure 5.18. Administrative Options**

The administrative options are global options which will apply to local extensions, agents, or new users. Choosing these global settings will help save time when creating new user, agent, or extension definitions.

- **Local Extension Settings** - Global settings for local extensions.
  - **Local Extensions are** - You can set all local extensions to be between two to five digits, or a variation.
  - **First Extension Number** - Set the first extension which all other extensions will be based upon. This extension is typically your primary extension.
  - **Operator Extension** - Designate the extension for the system operator from the list of extensions you have created.
  - **Allow Multiple Extensions** - Select this checkbox if you want to associate multiple extensions with the same analog phone.
  - **Allow AlphaNumeric Extensions** - Select this checkbox if you are a SIP/IAX user and wish extensions to be alphanumeric.
- **Agent Login Settings** - The following settings determine how agents will log on, log out, and defines the callback extension for an agent.

- **Agent Login Extension** - This field specifies the extension which all Agents can call to log in to their assigned queues.
- **Agent Callback Login Extension** - This field specifies the extension which all agents can log in to their assigned extensions without remaining on the line. When a call comes in for an agent, the agent's extension will be dialed.
- **Agent Logout** - There are two different methods an agent can use to log out dependent on their method of logging in. If an agent logged in to the queues using the extension specified in **Agent Login**, they can just hang up their phone. If an agent logged in via the **Agent Callback Login** extension, they should dial the same extension used to login, specify their extension and password when prompted, and hit # when asked for their callback extension. This will successfully log the agent out of all queues.
- **Default Settings for New User** - This section gives you the ability to set the default settings for all user extensions. Please refer to the User Extensions setting for information on each setting.

Click **Show Advanced Options** next to the Admin Settings label to access the advanced options pages. A small drop down box will be displayed in the upper right corner of the page. Select one of the following options to access that page.



### Note

Any changes made on the advanced options pages must be activated by clicking Activate Changes at the top of the GUI.

- **Music On Hold** - Access this page to specify Music On Hold classes.
- **VM Email Settings** - Asterisk can be configured to send voicemail files to a user extension via e-mail. The VM Email Settings page give you the ability to modify the e-mail template.
- **Global SIP Settings** - The SIP configuration settings can be enabled or disabled from this page.
- **Global IAX Settings** - The IAX configuration settings can be enabled or disabled from this page.
- **Change Password** - Access this page to change the system admin password.

## Part III. Reference

DRAFT



# Appendix A. Application Reference

Applications are the core functionality of the dialplan. Generally these all will operate on the channel, whereas functions, described in Appendix F, merely return values that can be used by applications. There are a few applications that still simply return values, but these will probably be deprecated in a future version and replaced with dialplan functions.

There are a few things to keep in mind about applications. First, when they exit, they will either terminate normally or abnormally. Abnormal termination almost always occurs when an application detects that the channel has hung up (or if it doesn't, the dialplan will detect that shortly thereafter). An application may also exit abnormally when it wishes to indicate to the dialplan that some condition has not been satisfied and that it should force a hangup. In all other cases, an application will exit normally, which indicates that processing should continue at the next priority in the dialplan.

In many cases, if you wish to override the application's wish to cause a hangup, you may wrap the application in a `TryExec()`.

In many places throughout this reference, you will see what's described as a *label*. This is shorthand for describing a location in the dialplan, whether it is simply a *priority*, an *extension* and a *priority*, or a *context*, an *extension*, and a *priority*. Note that if a *text label* is defined for a particular priority, the *priority* may be replaced with that *text label* in any of those cases. See the `GotoIf()` application for more information and an example.



## Note

You will find many of the examples in this appendix to contain numbered priorities, which is not the preferred method of writing dialplans. We prefer the use of the 'n' priority for all priority numbers except 1 (which is required), but we have decided to utilize them in order to make some of the examples more clear.

## AddQueueMember()

`AddQueueMember()` — Dynamically adds queue members to the specified call queue

## Synopsis

```
AddQueueMember(queuename [, interface [, penalty [, option [, membername ] ] ] ] )
```

Dynamically adds the specified *interface* to an existing queue named *queuename*, as specified in `queues.conf`. If specified, *penalty* sets the penalty for queues to use this member. Members with a lower penalty are called before members with a higher penalty.

The `AddQueueMember()` application sets a channel variable named `AQMSTATUS` upon completion. The `AQMSTATUS` variable will be set to one of the following values:

```
ADDED
MEMBERALREADY
NOSUCHQUEUE
```

Calling `AddQueueMember()` without an *interface* argument will use the interface that the caller is currently using.

If the *option* argument is set to `j`, Asterisk cannot add the *interface* to the specified queue, and there exists an `n+101` priority (where `n` is the number of the current priority), the call will jump to that priority.

The *membername* argument may be set to the name of the queue member. Consequently, this name will show up in the entries of the `queue_log` as well as Asterisk Manager Interface events, making it easier to identify the agent for reporting purposes:

```
; add SIP/3000 to the techsupport queue, with a penalty of 1
exten => 123,1,AddQueueMember(techsupport,SIP/3000,1)
```

## See Also

Queue(), RemoveQueueMember(), PauseQueueMember(), UnpauseQueueMember(), AgentLogin(), `queues.conf`

# ADSIProg()

ADSIProg() — Loads an ADSI script into an ADSI-capable phone

## Synopsis

```
ADSIProg(script)
```

Programs an Analog Display Services Interface (ADSI) phone with the given *script*. If none is specified, the default script, *asterisk.adsi*, is used. The path for the *script* is relative to the Asterisk configuration directory (usually `/etc/asterisk/`). You may also provide the full path to the script.

To get the CPE ID and other information from your ADSI-capable phone, use the `GetCPEID( )` application:

```
; program the ADSI phone with the telcordia-1.adsi script
exten => 123,1,ADSIProg(telcordia-1.adsi)
```

## See Also

GetCPEID(), `adsi.conf`

# AgentCallbackLogin()

AgentCallbackLogin() — Enables agent login with callback

## Synopsis

```
AgentCallbackLogin([AgentNumber][,[options][,[exten]@context]])
```

Allows a call agent identified by *AgentNumber* to log in to the call queue system, to be called back when a call comes in for that agent.

When a call comes in for the agent, Asterisk calls the specified *exten* (with an optional *context*).

The *options* argument may contain the letter *s*, which causes the login to be silent:

```
; silently log in as agent number 42, and have Asterisk
; call extension 123 in the internal context
; when a call comes in for this agent
```

```
exten => 123,1,AgentCallbackLogin(42,s,123@internal)
```



### Note

This application is deprecated, and the functionality has been replaced with AEL dialplan logic located in the `doc/queues-with-callback-members.txt` file within the Asterisk source.

## See Also

Queue(), AgentLogin(), AddQueueMember(), RemoveQueueMember(), PauseQueueMember(), UnpauseQueueMember(), AGENT, `agents.conf`, `queues.conf`

## AgentLogin()

AgentLogin() — Allows a call agent to log in to the system

## Synopsis

```
AgentLogin([AgentNumber][,options])
```

Logs the current caller in to the call queue system as a call agent (optionally identified by *AgentNumber*). While logged in, the agent can receive calls and will hear a beep on the line when a new call comes in. The agent can hang up the current call by pressing the asterisk (\*) key. If *AgentNumber* is not specified, the caller will be prompted to enter her agent number. Agents are defined in `agents.conf`.

The *options* argument may contain the letter *s*, which causes the login to be silent:

```
; silently log in the caller as agent number 42, as defined in agents.conf
exten => 123,1,AgentLogin(42,s)
```

## See Also

Queue(), AddQueueMember(), RemoveQueueMember(), PauseQueueMember(), UnpauseQueueMember(), AGENT, `agents.conf`, `queues.conf`

## AgentMonitorOutgoing()

AgentMonitorOutgoing() — Records an agent's outgoing calls

## Synopsis

```
AgentMonitorOutgoing([options])
```

Records all outbound calls made by a call agent.

This application tries to figure out the ID of the agent who is placing an outgoing call based on a comparison of the Caller ID of the current interface and the global variable set by the `AgentCallbackLogin()` application. As such, it should be used only in conjunction with (and after!) the `AgentCallbackLogin()` application. It uses the monitoring functions in the `chan_agent` module instead of the `Monitor()` application to record the calls. This means that call recording must be configured correctly in the `agents.conf` file.

By default, recorded calls are saved to the `/var/spool/asterisk/monitor/` directory. This may be overridden by changing the `savecallsin` parameter in `agents.conf`.

If the Caller ID and/or agent ID are not found, this application will go to priority `n+1`, if it exists (where `n` is the current priority).

Returns 0 unless overridden by one of the options.

The `options` argument may include one or more of the following:

- d Make this application return -1 if there is an error condition and there is no extension `n+101`.
- c Change the Call Detail Record so that the source of the call is recorded as `Agent/agent_id`.
- n Don't generate warnings when there is no Caller ID or if the agent ID is not known. This option is useful if you want to have a shared context for agent and non-agent calls.

```
; record outbound calls for this agent, and change the CDR to reflect
; that the call is being made by an agent
exten => 123,1,AgentMonitorOutgoing(c)
```

## See Also

`AgentCallbackLogin()`, `agents.conf`

## AGI()

`AGI()` — Executes an AGI-compliant application

## Synopsis

```
[E]AGI(program[ ,arguments])
```

Executes an Asterisk Gateway Interface-compliant *program* on the current channel. AGI programs allow external programs (written in almost any language) to control the telephony channel by playing audio, reading DTMF digits, and so on. Asterisk communicates with the AGI program on `STDIN` and `STDOUT`. The specified *arguments* are passed to the AGI program.

The *program* must be set as executable in the underlying filesystem. The program path is relative to the Asterisk AGI directory, which by default is `/var/lib/asterisk/agi-bin/`.

If you want to run an AGI when no channel exists (such as in an `h` extension), use the `DeadAGI()` application instead. You may want to use the `FastAGI()` application if you want to do AGI processing across the network.

If you want access to the inbound audio stream from within your AGI program, use `EAGI()` instead of `AGI()`. Inbound audio can then be read in on file descriptor 3.

If the channel hangs up prematurely, the process initiated by the AGI command will be sent a HUP signal to tell it that the channel has hung up. If your program does not catch this signal, it will be terminated. You can override this behavior by setting the channel variable `AGISIGHUP` to 0:

```
; call the demo AGI program
exten => 123,1,AGI(agi-test)
exten => 123,2,EAGI(eagi-test)
```

## See Also

DeadAGI()

## AMD()

AMD() — Answering machine detection

## Synopsis

```
AMD([initialSilence[,greeting[,afterGreetingSilence[,totalAnalysisTime
[,minimumWordLength[,betweenWordsSilence[,maximumNumberOfWords
[,silenceThreshold]]]]]]]])
```

This application attempts to detect an answering machine, based on the timing patterns. This application is usually used by outbound calls originated from either call files or from the Asterisk manager Interface. This application sets AMDSTATUS variable is set to one of the following, to show what type of call was detected:

- MACHINE    The called party is believed to be an answering machine.
- HUMAN      The called party is believed to be a human being, and not an answering machine.
- NOTSURE    The application was unable to tell whether the called party was a human or an answering machine.
- HANGUP     A hangup occurred during the detection.

The AMD() application also sets a channel variable named AMDCAUSE with the cause that lead to the conclusion stated in the AMDSTATUS variable. The AMDCAUSE variable will be set to one of the following values:

```
TOOLONG-total_time
INITIALSILENCE-silence_duration-initial_silence
HUMAN-silence_duration-after_greeting_silence
MAXWORDS-word_count-maximum_number_of_words
LONGGREETING-voice_duration-greeting
```

The parameters to this application all help tune it so that it can more effectively tell the difference between a human and an answering machine. If the parameters are not passed to this application, Asterisk will read the default values as configured in `amd.conf`. The parameters are:

- |                             |  |
|-----------------------------|--|
| <i>initialSilence</i>       | The maximum silence duration before the greeting. If exceeded, then the AMDSTATUS variable will be set to MACHINE.                   |
| <i>greeting</i>             | The maximum length of the greeting. If exceeded, then the AMDSTATUS variable will set to MACHINE.                                    |
| <i>afterGreetingSilence</i> | The maximum silence after detecting a greeting. If exceeded, then the AMDSTATUS variable will be set to MACHINE.                     |
| <i>totalAnalysisTime</i>    | The maximum time allowed for the algorithm to decide whether the called party is a human or an answering machine.                    |
| <i>minimumWordLength</i>    | If the duration of the voice activity is shorter than <code>minimumWordLength</code> , it will not be considered to be human speech. |

<i>betweenWordsSilence</i>	The minimum duration of silence after a word to consider the audio that follows as a new word.
<i>maximumNumberOfWords</i>	The maximum number of words detected in the greeting. If exceeded, then the AMD-STATUS variable will set to MACHINE.
<i>silenceThreshold</i>	How sensitive the algorithm should be when detecting silence

```

; Use answering machine detection.  If the called party
; is human, connect them to Bob.  Otherwise, play a
; message and hang up
exten => 123,1,Answer()
exten => 123,n,AMD()
exten => 123,n,GotoIf($["${AMDSTATUS}" = "HUMAN"]?human:machine)
exten => 123,n(machine),WaitForSilence(2000)
exten => 123,n,Playback(asterisk-friend)
exten => 123,n,Hangup()
exten => 123,n(human),Verbose(3, We've got a human on the line!)
exten => 123,n,Playback(transfer)
exten => 123,n,Dial(SIP/bob)
exten => 123,n,Playback(im-sorry)
exten => 123,n,Hangup()

```

## See Also

WaitForSilence()

## Answer()

Answer() — Answers a channel, if it is ringing

## Synopsis

```
Answer([delay])
```

Causes Asterisk to answer the channel if it is currently ringing. If the current channel is not ringing, this application does nothing.

If a *delay* is specified, Asterisk will answer the call and then wait *delay* milliseconds before going on to the next priority in the dialplan.

It is often a good idea to use Answer() on the channel before calling any other applications, unless you have a very good reason not to. There are several key applications that require that the channel be answered before they are called, and may not work correctly otherwise:

```

exten => 123,1,Answer(750)
exten => 123,n,Playback(tt-weasels)

```

## See Also

Hangup()

# AppendCDRUserField()

AppendCDRUserField() — Appends a value to the user field of the Call Detail Record

## Synopsis

```
AppendCDRUserField(value)
```

Appends *value* to the user field of the Call Detail Record (CDR). The user field is often used to store arbitrary data about the call, which may not be appropriate for any of the other fields:

```
; set the user field to 'abcde'
exten => 123,1,SetCDRUserField(abcde)
; now append 'xyz'
exten => 123,1,AppendCDRUserField(xyz)
```



### Note

This application has been deprecated in favor of the CDR function.

```
exten => 123,1,Set(CDR(userfield)=${CDR(userfield)}12345)
```

## See Also

SetCDRUserField(), ForkCDR(), NoCDR(), ResetCDR(), the CDR

# Authenticate()

Authenticate() — Requires that the caller enter a correct password before continuing

## Synopsis

```
Authenticate(password[,options[,maxdigits]])
```

Requires a caller to enter a given *password* in order to continue execution of the next priority in the dialplan. `Authenticate()` gives the caller three chances to enter the password correctly. If the password is not correctly entered after three tries, the channel is hung up.

If *password* begins with the `/` character, it is interpreted as a file that contains a list of valid passwords (one per line). Passwords may also be stored in the Asterisk database (AstDB); see the `d` option below.

The *maxdigits* parameter sets the maximum number of digits that may be entered by the caller. If not set, the application will accept an unlimited number of digits and will wait for the caller to press the `#` key after entering his authentication code.

A set of *options* may be provided, consisting of one or more of the letters in the following list:

- a Sets the CDR field named `accountcode` and the channel variable `ACCOUNTCODE` to the password that is entered
- d Interprets the path as the database key from the Asterisk database in which to find the password, not a literal file. When using a database key, the value associated with the key can be anything.

- j Supports jumping to priority `n+101` if authentication fails
- m Interprets the given path as a file that contains a list of account codes and password hashes delimited with `:` (colon character), listed one per line in the file. When one of the passwords is matched, the channel will have its account code set to the corresponding account code in the file.
- r Removes the database key upon successful entry (valid with `d` only).

```
; force the caller to enter the password before continuing,
; and set the CDR field named 'accountcode' to the entered password
exten => 123,1,Answer()
exten => 123,n,Authenticate(1234,a)
exten => 123,n,Playback(pin-number-accepted)
exten => 123,n,SayDigits(${ACCOUNTCODE})
```

## See Also

VMAuthenticate(), DISA()

## Background()

Background() — Plays a file while accepting touch-tone (DTMF) digits

## Synopsis

```
Background(filename1[&filename2...][, options[, language]])
```

Plays the specified audio file(s) while waiting for the user to begin entering DTMF digits. Once the user begins to enter DTMF digits, the playback is terminated. Asterisk tries to find a matching extension in the destination *context* (or the current context if none is specified), and execution of the dialplan will continue at the matching extension as soon as an unambiguous match is found.

The *filename* should be specified without a file extension, as Asterisk will automatically find the file format with the lowest translation cost.

Valid *options* include one of the following:

- s Causes the playback of the message to be skipped if the channel is not in the “up” state (i.e., hasn’t yet been answered). If `s` is specified, the application will return immediately should the channel not be off-hook.
- n Does not answer the channel before playing the specified file. Without this option, the channel will automatically be answered before the sound is played. Not all channels support playing messages before being answered.
- m Only break if a digit hit matches a one-digit extension in the destination context.

The *language* argument may be used to specify a language to use for playing the prompt, if it differs from the current language of the channel.

```
exten => 123,1,Answer()
exten => 123,2,Background('exter-ext-of-person');
```

## See Also

ControlPlayback(), WaitExten(), BackgroundDetect(), TIMEOUT



## BackgroundDetect()

BackgroundDetect() — Plays a file in the background and detects talking

### Synopsis

```
BackgroundDetect(filename[,sil[,min[,max]]])
```

Similar to Background(), but attempts to detect talking.

During the playback of the file, audio is monitored in the receive direction. If a period of non-silence that is greater than *min* milliseconds yet less than *max* milliseconds and is followed by silence for at least *sil* milliseconds occurs, the audio playback is aborted and processing jumps to the `talk` extension, if available.

If unspecified, *sil*, *min*, and *max* default to 1,000 ms, 100 ms, and infinity, respectively.

```
exten => 123,1,BackgroundDetect(tt-monkeys)
exten => 123,2,Playback(im-sorry)
exten => talk,1,Playback(yes-dear)
```

### See Also

Playback(), Background()

## Busy()

Busy() — Indicates a busy condition to the channel

### Synopsis

```
Busy([timeout])
```

Requests that the channel indicate the busy condition and then waits for the user to hang up or for the optional *timeout* (in seconds) to expire.

This application signals a busy condition only to the bridged channel. Each particular channel type has its own way of communicating the busy condition to the caller. You can use `Playtones(busy)` to play a busy tone to the caller.

```
exten => 123,1,Playback(im-sorry)
exten => 123,2,Playtones(busy)
exten => 123,3,Busy()
```

### See Also

Congestion(), Progress(), Playtones(), Hangup()

## ChangeMonitor()

ChangeMonitor() — Changes the monitoring filename of a channel

## Synopsis

```
ChangeMonitor(filename_base)
```

Changes the name of the recorded file created by monitoring a channel with the `Monitor()` application. This application has no effect if the channel is not monitored. The argument *filename\_base* is the new filename base to use for monitoring the channel.

```
; start recording this channel with a basename of 'sample'
exten => 123,1,Monitor(sample)
; change the filename base to 'example'
exten => 123,2,ChangeMonitor(example)
```

## See Also

`Monitor()`, `StopMonitor()`, `MixMonitor()`

## ChanIsAvail()

`ChanIsAvail()` — Finds out if a specified channel is currently available

## Synopsis

```
ChanIsAvail(technology1/resource1 [&technology2/resource2...] [, option])
```

Checks to see if any of the requested channels are available. This application also sets the following channel variables:

AVAILCHAN	The name of the available channel, including the call session number used to perform the test
AVAILORIGCHAN	The canonical channel name that was used to create the channel—that is, the channel name without any session number
AVAILSTATUS	The status code for the channel

If the option *s* (which stands for “state”) is specified, Asterisk will consider the channel unavailable whenever it is in use, even if it can take another call.

If the *j* option is specified, and none of the requested channels are available, the new priority will be *n*+101 (where *n* is the current priority), if that priority exists.

```
; check both Zap/1 and Zap/2 to see if they're available
exten => 123,1,ChanIsAvail(Zap/1&Zap/2)
; print the available channel name to the Asterisk CLI
exten => 123,2,Verbose(0,${AVAILORIGCHAN})
```



### Warning

This application does not work correctly on MGCP channels.

## ChannelRedirect()

`ChannelRedirect()` — Redirects a channel to a new location in the dialplan

## Synopsis

```
ChannelRedirect(channel,[[context,]extension,]priority)
```

This application redirects the specified *channel* to a new *priority* in the dialplan. If *extension* is not specified, the current extension is assumed. If *context* is not specified, the current context will be assumed:

```
; Transfer SIP/Bob to hold music when extension 123 is dialed
exten => 123,1,ChannelRedirect(SIP/Bob,124,1)

exten => 124,1,Answer()
exten => 124,2,MusicOnHold()
```

## See Also

Transfer()

## ChanSpy()

ChanSpy() — Listens to the audio on a channel, and optionally whisper to the calling channel

## Synopsis

```
ChanSpy([chanprefix[,options]])
```

This application is used to listen to the audio going to and from an Asterisk channel. If the *chanprefix* parameter is specified, only channels beginning with this value will be spied upon.

While a channel is being spied upon, the following actions may be performed:

- Dialing # cycles the volume level.
- Dialing \* will cause the application to spy on the next available channel.
- Dialing a series of digits followed by # builds a channel name (which will be appended to *chanprefix*). For example, placing ChanSpy(Zap) and then dialing the digits 42# while spying will begin spying on the channel Zap/42.

The *options* parameter may contain zero or more of the following options:

b	Only spy on channels that are involved in a bridged call.
g( <i>group</i> )	Only spy on channels that contain a channel variable named SPYGROUP, which should contain <i>group</i> in an optional colon-delimited list.
q	Quiet mode. Tells the application not to beep or read the selected channel's name when spying begins.
r( <i>basename</i> )	Records the channel audio to the monitor spool directory (usually /var/spool/asterisk/monitor). An optional <i>basename</i> set the base filename of the recordings, which defaults to chanspy.

<code>v([value])</code>	Adjusts the volume of the audio being listened to. The <i>value</i> must be in the range of 4 to -4. A negative <i>value</i> will make the volume quieter, while a positive value will make it louder.
<code>w</code>	Whisper mode. This allows the spying channel to talk to the spied-upon channel, without any other bridged channel being able to hear the audio.
<code>W</code>	Private whisper mode. This enables the spying channel to speak to the spied-upon channel without being able to hear the audio from the spied-upon channel.

```
; Spy on the Zap channels in whisper mode
exten => 123,1,ChannelSpy(Zap,w)
```

## See Also

ExtenSpy()

## Congestion()

Congestion() — Indicates congestion on the channel

## Synopsis

```
Congestion([timeout])
```

Requests that the channel indicate congestion and then waits for the user to hang up or for the optional *timeout* (in seconds) to expire.

This application signals congestion only to the far end; it doesn't actually play a congestion tone to the user. Use `Playtones(congestion)` to play a congestion tone to the caller.



### Warning

If you use this command without a timeout, you run the risk of having a channel get stuck in this state. This is not really needed when you want to indicate congestion to a user. Just use `Playtones(congestion)` so they hear the fast-busy, and then `Hangup()`.

Always exits abnormally:

```
; if the Caller ID number is 555-1234, always play congestion
exten => 123,1,GotoIf($[${CALLERID(num)} = 5551234]?5:2)
exten => 123,2,Playtones(congestion)
exten => 123,3,Congestion(3)
exten => 123,4,Hangup()
exten => 123,5,Dial(Zap/1)
```

## See Also

Busy(), Progress(), Playtones(), Hangup()

## ContinueWhile()

ContinueWhile() — Restart a `While()` loop

## Synopsis

```
ContinueWhile()
```

Return to the top of a While loop and re-evaluate the conditional.

## See Also

While(), ExitWhile()

## ControlPlayback()

ControlPlayback() — Plays a file, with the ability to fast forward and rewind the file

## Synopsis

```
ControlPlayback(file[, skipms[, ff[, rew[, stop[, pause[, restart[, options]]]]])
```

Plays back a given *file* (without the file extension), while allowing the caller to move forward and backward through the file by pressing *ff* and *rew* keys. By default, you can use *\** and *#* to rewind and fast-forward the playback of the file, respectively.

The *skipms* option specifies how far forward or backward to jump in the file with each press of *ff* or *rew*.

If *stop* is specified, the application will stop playback when *stop* is pressed.

A *pause* argument may also be specified, which when pressed will pause playback of the file. Pressing *pause* again will continue the playback of the file.

If the *restart* parameter is specified, the specified key may be used to restart the playback of the *file*.

If the *options* parameter is set to *j*, and the *file* does not exist, the application jumps to priority *n+101*, if present (where *n* is the current priority number).

The ControlPlayback() application sets a channel variable named CPLAYBACKSTATUS upon completion. The CPLAYBACKSTATUS variable will be set to one of the following values:

```
SUCCESS
USERSTOPPED
ERROR
```

```
; allow the caller to control the playback of this file
exten => 123,1,ControlPlayback(tt-monkeys|3000|#|*|5|0)
```

## See Also

Playback(), Background(), Dictate(),

## DateTime()

DateTime() — Says the date and/or time in the user-specified format

## Synopsis

```
DateTime([unixtime[,timezone[,format]]])
```

If the *unixtime* parameter is specified, this application says that date and time. Otherwise, it says the current date and time. If a *timezone* is specified, the date and time is calculated according to that time zone. Otherwise, the time zone setting of the Asterisk server is used. If the *format* parameter is specified, the date and time will be said according to that format. (See the sample *voicemail.conf* file for more information on the date and time format.)

```
; say the current date and time in several time zones
exten => 123,1,DateTime(,America/New_York)
exten => 123,2,DateTime(,America/Chicago)
exten => 123,3,DateTime(,America/Denver)
exten => 123,4,DateTime(,America/Los_Angeles)
```

## DBdel()

DBdel() — Deletes a key from the AstDB

## Synopsis

```
DBdel(family/key)
```

Deletes the key specified by *key* from the key family named *family* in the AstDB.

```
exten => 123,1,DBput(test/name=John) ; add name to AstDB
exten => 123,2,DBget(NAME=test/name) ; retrieve name from AstDB
exten => 123,3,DBdel(test/name)      ; delete from AstDB
```



### Note

This application is deprecated and the functionality has been replaced with the `DB_DELETE ( )` function.

## See Also

`DB_DELETE ( )`, `DBdeltree()`, `DB`

## DBdeltree()

DBdeltree() — Deletes a family or key tree from the AstDB

## Synopsis

```
DBdeltree(family[/keytree])
```

Deletes the specified *family* or *keytree* from the AstDB.

```
; create a couple of entries in the AstDB
exten => 123,1,DBput(test/blue)
```

```
exten => 123,2,DBput(test/green)
; now delete the key family named test
exten => 123,3,DBdeltree(test)
```

## See Also

DB\_DELETE( ), DBdel(), DB

# DeadAGI()

DeadAGI() — Executes an AGI-compliant script on a dead (hung-up) channel

## Synopsis

```
DeadAGI(program,args)
```

Executes an AGI-compliant *program* on a dead (hung-up) channel. AGI allows Asterisk to launch external programs written in almost any language to control a telephony channel, play audio, read DTMF digits, and so on by communicating with the AGI protocol on STDIN and STDOUT. The arguments specified by *args* will be passed to the program.

This application has been written specifically for dead channels, as the normal AGI interface doesn't work correctly if the channel has been hung up.

Use the `show agi` command on the command-line interface to list all of the available AGI commands.

```
exten => h,1,DeadAGI(agi-test)
```

## See Also

AGI()

# Dial()

Dial() — Attempts to connect channels

## Synopsis

```
Dial(tech/username:password@hostname/extension[&tech2/peer2...]  
[,ring-timeout[,flags[,URL]]])
```

Allows you to connect together all of the various channel types.<sup>1</sup> `Dial()` is the most important application in Asterisk; you'll want to read through this section a few times.

<sup>1</sup>The fact that Asterisk will happily connect IAX, SIP, H.323, Skinny, PRI, FX(O/S), and anything else is amazing, but possibly the most amazing of all is the Local channel. By allowing a single `Dial()` command to connect to multiple Local channels, one `Dial()` event can trigger a multitude of completely independent and unique actions in other parts of the dialplan. The power of this concept is truly revolutionary and has to be experienced to be believed.

Any valid channel type (such as SIP, IAX2, H.323, MGCP, Local, or Zap) is acceptable to `Dial()`, but the parameters that need to be passed to each channel will depend on the information the channel type needs to do its job. For example, a SIP channel will need a network address and user to connect to, whereas a Zap channel is going to want some sort of phone number.

When you specify a channel type that is network-based, you can pass the destination host (name or IP address), username, password, and remote extension as part of the options to `Dial()`, or you can refer to the name of a channel entry in the appropriate `.conf` file; all the required information will then need to be obtained from that file. The username and password can be replaced with the name contained within square brackets (`[]`) of the channel configuration file. The hostname is optional.

This is a valid `Dial` statement:

```
exten => s,1,Dial(SIP/sake:arigato@thathostoverthere.tld)
```

This is effectively identical:

```
exten => s,1,Dial(SIP/some_SIP_friend)
```

but will work only if there is a channel defined in `sip.conf` as `[some_SIP_friend]`, whose channel definition contains `fromuser=sake,password=arigato, and host=thathostoverthere.tld`.

An extension number is often attached after the address information, like this:

```
exten => s,1,Dial(IAX2/user:pass@otherend.com/500)
```

This asks the far end to connect the call to extension 500 in the context in which the channel arrived. The extension is not required by `Dial()`, as the information in the remote end's channel configuration file may be used, or the remote server will pass the call to the `s` extension in the context in which the call came in. Ultimately, the far end controls what happens to the call; you can only request a specific treatment.

If no *ring-timeout* is specified, the channel will ring indefinitely. This is not always a bad thing, so don't feel you need to set it; just be aware that "indefinitely" could mean a very long time. *ring-timeout* is specified in seconds. The ring timeout always follows the addressing information, like this:

```
exten => s,1,Dial(IAX2/user:pass@otherend.com/500,ring-timeout)
```

Much of the power of the `Dial()` application is in the flags. These are assigned following the addressing and timeout information, like this:

```
exten => s,1,Dial(IAX2/user:pass@otherend.com/500,60,flags)
```



## Tip

If you don't have a timeout specified, and you want to assign flags, you must still assign a spot for the timeout. You do this by adding an extra comma in the spot where the timeout would normally go, like this:

```
exten => s,1,Dial(IAX2/user:pass@otherend.com/500,,flags)
```

The valid flags that may be used with the `Dial()` application are:

- |               |  |
|---------------|--|
| A( <i>x</i> ) | Plays an announcement to the called party; <i>x</i> is the filename of the sound file to play as the announcement.   |
| C             | Resets the Call Detail Record for the call. Since the CDR time is set to when you <code>Answer()</code> the call, you may wish to reset the CDR so the end user is not billed for the time prior to the <code>Dial()</code> application being invoked. |



d	Allows the user to dial a one-digit extension while waiting for a call to be answered. The call will then exit to that extension (either in the current context, if it exists, or in the context specified by the EXITCONTEXT variable).										
D([ <i>called</i> ][: <i>calling</i> ])	Sends DTMF digits after the call has been answered, but before the call is bridged. The <i>called</i> parameter is passed to the called party, and the <i>calling</i> parameter is passed to the calling party. Either parameter may be used individually.										
f	Forces the Caller ID of the <i>calling</i> party to be set as the extension associated with the channel using a dialplan hint. This is often used when a provider doesn't allow the Caller ID to be set to anything other than a number that is assigned to you. For example, if you had a PRI, you would use the f flag to override any Caller ID set locally on a SIP phone.										
g	Execution of the dialplan goes on in the current context if the destination channel hangs up.										
G( <i>context</i> ^ <i>extension</i> ^ <i>priority</i> )	When the call is answered, the calling party is transferred to the specified priority and the called party to the specified priority+1. You cannot use any additional action post-answer options in conjunction with this option.										
h	Allows the called user to hang up the channel by pressing the * key.										
H	Allows the calling user to hang up the channel by pressing the * key.										
i	Causes Asterisk to ignore any forwarding requests it may receive on this dial attempt.										
j	Causes Asterisk to jump to priority n+101 if all the requested channels were busy (where n is the current priority).										
L( <i>x</i> [: <i>y</i> ] [: <i>z</i> ] )	Limits the call to <i>x</i> milliseconds, warning when <i>y</i> milliseconds are left and repeating every <i>z</i> milliseconds until the limit is reached. The <i>x</i> parameter is required; the <i>y</i> and <i>z</i> parameters are optional. The following special variables may also be set to provide additional control: <table> <tr> <td>LIMIT_PLAYAUDIO_CALLER=yes</td><td>Specifies whether to play sounds to the caller. Defaults to yes.</td></tr> <tr> <td>LIMIT_PLAYAUDIO_CALLEE=yes</td><td>Specifies whether to play sounds to the callee.</td></tr> <tr> <td>LIMIT_TIMEOUT_FILE=<i>filename</i></td><td>Specifies which file to play when time is up.</td></tr> <tr> <td>LIMIT_CONNECT_FILE=<i>filename</i></td><td>Specifies which file to play when call begins.</td></tr> <tr> <td>LIMIT_WARNING_FILE=<i>filename</i></td><td>Specifies the file to play if the argument <i>y</i> is defined. Defaults to saying the time remaining.</td></tr> </table>	LIMIT_PLAYAUDIO_CALLER=yes	Specifies whether to play sounds to the caller. Defaults to yes.	LIMIT_PLAYAUDIO_CALLEE=yes	Specifies whether to play sounds to the callee.	LIMIT_TIMEOUT_FILE= <i>filename</i>	Specifies which file to play when time is up.	LIMIT_CONNECT_FILE= <i>filename</i>	Specifies which file to play when call begins.	LIMIT_WARNING_FILE= <i>filename</i>	Specifies the file to play if the argument <i>y</i> is defined. Defaults to saying the time remaining.
LIMIT_PLAYAUDIO_CALLER=yes	Specifies whether to play sounds to the caller. Defaults to yes.										
LIMIT_PLAYAUDIO_CALLEE=yes	Specifies whether to play sounds to the callee.										
LIMIT_TIMEOUT_FILE= <i>filename</i>	Specifies which file to play when time is up.										
LIMIT_CONNECT_FILE= <i>filename</i>	Specifies which file to play when call begins.										
LIMIT_WARNING_FILE= <i>filename</i>	Specifies the file to play if the argument <i>y</i> is defined. Defaults to saying the time remaining.										
m[ <i>class</i> ]	Provides music to the calling party until the call is answered. You may also optionally indicate the music-on-hold <i>class</i> , as defined in musiconhold.conf.										

`M( x [ ^arg ] )`

Executes the macro `x` upon the connection of a call, optionally passing arguments delimited by `^`. The macro can also set the `MACRO_RESULT` channel variable to one of the following values, to determine what should happen after the macro has finished:

ABORT	Hangs up both legs of the call.
CONGESTION	Acts as if the line encountered congestion.
BUSY	Acts as if the line was busy. If the <i>j</i> option is specified, it sends the call to priority <code>n+101</code> , where <i>n</i> is the current priority.
CONTINUE	Hangs up the called party and continues on in the dialplan.
<code>GOTO:&lt;context&gt;^&lt;extension&gt;</code>	Transfers the call to the specified destination.



### Warning

You cannot use any additional action post-answer options in conjunction with this option. Also, PBX services are not run on the called channel, so you will not be able to set timeouts via the `TIMEOUT` function in this macro.

`n`

This option is a modifier for the screen/privacy mode. It specifies that no introductions are to be saved in the *priv-callerintros* directory.

`N`

This option is a modifier for the screen/privacy mode. It tells Asterisk not to screen the call if Caller ID is present.

`o`

Uses the Caller ID received on the inbound leg of the call for the Caller ID on the outbound leg of the call. This is useful if you are accepting a call and then forwarding it to another destination, but you wish to pass the Caller ID from the inbound leg of the call instead of overwriting it with the local Caller ID settings. This was the default behavior on Asterisk versions prior to 1.0.

`O[x]`

This option turns on Operator Services mode on a Zaptel channel. If this option is used on a non-Zaptel interface, it will be ignored. When the destination answers (presumably an operator services station), the originator no longer has control of her line. She may hang up, but the switch will not release her line until the destination party (the operator) hangs up. Specified without an arg, or with 1 as an arg, the originator hanging up will cause the phone to ring back immediately. With a 2 specified as the argument, when the “operator” flashes the trunk, it will ring the caller’s phone.

`p`

This option enables screening mode. This is basically Privacy mode without memory.

`P[(x)]`

Sets the privacy mode, optionally specifying `x` as the family/key value in the local AstDB database. This option is useful for accepting calls based on a blacklist (explicitly denying calls from listed numbers) or whitelist (explicitly accepting calls from listed numbers). See also `LookupBlacklist( )`.

r	Indicates ringing to the calling party, without passing any audio until the call is answered. This flag is not normally required to indicate ringing, as Asterisk will signal ringing if a channel is actually being called.
S(x)	Hangs up the call x seconds after the <i>called</i> party has answered the call.
t	Permits the called party to transfer a call by pressing the # key. Please note that if this option is used, reinvites are disabled, as Asterisk needs to monitor the call to detect when the called party presses the # key.
T	Permits the caller to transfer a connected call by pressing the # key. Again, note that if this option is used, reinvites are disabled, as Asterisk needs to monitor the call to detect when the caller presses the # key.
w	Permits the called user to start and stop recording the call audio to disk by pressing the automon sequence (as configured in <code>features.conf</code> ). If the variable <code>TOUCH_MONITOR</code> is set, its value will be passed as the arguments to the <code>Monitor()</code> application when recording is started. If it is not set, the default values of <code>WAV   m</code> are passed to <code>Monitor()</code> .
W	Permits the calling user to record the call audio to disk by pressing the automon sequence (as configured in <code>features.conf</code> ).
k	Permits the called party to park the call by sending the DTMF sequence defined for call parking in <code>features.conf</code> .
K	Permits the calling party to park the call by sending the DTMF sequence defined for call parking in <code>features.conf</code> .

If the *URL* argument is included, that URL will be sent to the channel (if supported).



## Note

If the channel variable named `OUTBOUND_GROUP` is set before `Dial()` is called, all peer channels created by this application will be put in to that call group. In the following example, all peer channels created by the `Dial()` application will be part of the `test` call group:

```
; using OUTBOUND_GROUP
exten => 123,1,Set(OUTBOUND_GROUP=test)
exten => 123,n,Dial(IAX2/anotherbox/12345)
```

If the `OUTBOUND_GROUP_ONCE` variable is set, all peer channels created by this application will be put in to that group. Unlike `OUTBOUND_GROUP`, however, the variable will be unset after use.

The `Dial()` application sets the following variables upon exiting:

DIALEDTIME	The total time elapsed from execution of <code>Dial()</code> until completion.						
ANSWEREDTIME	The total time elapsed during the call.						
DIALSTATUS	The status of the call, set as one of the following values: <table> <tr> <td>CHANUNAVAIL</td><td>The channel is unavailable.</td></tr> <tr> <td>CONGESTION</td><td>The channel returned a congestion signal, usually indicating that it was unable to complete the connection.</td></tr> <tr> <td>NOANSWER</td><td>The channel did not answer in the time indicated by the ring-timeout option.</td></tr> </table>	CHANUNAVAIL	The channel is unavailable.	CONGESTION	The channel returned a congestion signal, usually indicating that it was unable to complete the connection.	NOANSWER	The channel did not answer in the time indicated by the ring-timeout option.
CHANUNAVAIL	The channel is unavailable.						
CONGESTION	The channel returned a congestion signal, usually indicating that it was unable to complete the connection.						
NOANSWER	The channel did not answer in the time indicated by the ring-timeout option.						

BUSY	The dialed channel is currently busy.
ANSWER	The channel answered the call.
CANCEL	The call was cancelled.
DONTCALL	The call was set to DONTCALL by the screening or privacy options.
TORTURE	The call was set to TORTURE by the screening or privacy options.
INVALIDARGS	Invalid arguments were passed to the Dial ( ) application.

```

; dial a seven-digit number on Zap channel 4
exten => 123,1,Dial(Zap/4/2317154)

; dial the same number, but this time only have it ring for 10 seconds
; before continuing on with the dialplan
exten => 124,1,Dial(Zap/4/2317154,10)
exten => 124,2,Playback(im-sorry)
exten => 124,3,Hangup()

; dial the same number, but this time with no timeout, and using the
; t, T, and m flags
exten => 125,1,Dial(Zap/4/2317154,,tTm)

; dial extension 500 at a remote host (over the IAX protocol), using
; the specified username and password
exten => 126,1,Dial(IAX2/username:password@remotehost/500)

; dial a number, but limit the call to 5 minutes (300,000 milliseconds)
; start warning the caller 4 minutes (240,000 milliseconds) in to the call,
; and repeat the warning every 30 seconds (30,000 milliseconds)
exten => 127,1,Dial(Zap/4/2317154,,L[300000:240000:30000])

```

## See Also

RetryDial()

## Dictate()

Dictate() — Virtual dictation machine

## Synopsis

```
Dictate([base_dir[,filename]])
```

This application allows the recording and playback of files, similar to a traditional dictation machine. The *base\_dir* parameter specifies the directory in which Asterisk will write the recorded files. If not specified, it defaults to the *dictate* subdirectory of the Asterisk spool directory (as defined in *asterisk.conf*).

If the *filename* parameter is specified, it will be used when the file is written. If not specified, Asterisk will prompt the caller for a numeric filename for the file.



## Note

Asterisk writes the files in raw, headerless, signed-linear format. If you'd like to convert the file to another format, you can use an outside utility such as `sox`, or use the `file convert` command from the Asterisk command-line interface.

The `Dictate()` application has two main modes: recording mode and playback mode. The caller can press the 1 key to switch between these modes. In both modes, the 0 key can be used to get help. The \* key is used to pause or unpause the recording or playback. The # key allows the caller to choose a new filename.

In recording mode, the 8 key can be used to erase the entire recording and start over.

In playback mode, the 7 key rewinds the recording a few frames, and the 8 key forwards the recording a few frames. The 2 key is used to toggle the playback speed (either 1x, 2x, 3x, or 4x).

```
; begin dictating, and save the files in the /tmp/dictate directory
exten => 123,1,Dictate(/tmp/dictate)
```

## See Also

`Playback()`, `Background()`, `ControlPlayback()`,

## Directory()

`Directory()` — Provides a dialable directory of extensions

## Synopsis

```
Directory(vm-context[,dial-context[,options]])
```

Presents users with a directory of extensions from which they may select by name. The list of names and extensions is discovered from `voicemail.conf`. The *vm-context* argument is required; it specifies the context of `voicemail.conf` to use.

The *dial-context* argument is the context to use for dialing the users, and it defaults to *vm-context* if unspecified. If the *options* argument is set to `f`, Asterisk will find a directory match based on the first name in `voicemail.conf` instead of the last name. If the `e` option is specified, Asterisk will read the extension of the directory match as well as the person's name.

If the user enters 0 (zero) and there exists an extension `o` (the lowercase letter o) in the current context, the call control will go to that extension. Entering \* will exit similarly, but to the `a` extension, much like `Voicemail()`'s behavior.

```
exten => *,1,Directory(default,incoming)
exten => #,1,Directory(default,incoming,f)
exten => 9,1,Directory(default,incoming,fe)
```

## DISA()

`DISA()` — Direct Inward System Access: allows inbound callers to make outbound calls

## Synopsis

```
DISA(password[,context[,callerid[,mailbox[@vmcontext]]]])
DISA(password-file[,callerid[,mailbox[@vmcontext]]])
```

Allows outside callers to obtain an “internal” system dial tone and to place calls from it as if they were placing calls from within the switch. The user is given a dial tone, after which she should enter her passcode, followed by #. If the passcode is correct, the user is then given a system dial tone on which a call may be placed.



### Warning

Obviously, this type of access has serious security implications, and *extreme* care must be taken not to compromise the security of your phone system.

The *password* argument is a numeric passcode that the user must enter to be able to make outbound calls. Using this syntax, all callers to this extension will use the same password. To allow users to use `DISA()` without a password, use the string `no-password` instead of the password.

The *context* argument specifies the context in which the user will be dialing. If no context is specified, the `DISA()` application defaults the context to `disa`.

The *callerid* argument specifies a new Caller ID string that will be used on the outbound call.

The *mailbox* argument is the mailbox number (and optional voicemail context, *vmcontext*) of a voicemail box. The caller will hear a stuttered dial tone if there are any new messages in the specified voicemail box.

Additionally, you may use an alternate syntax and pass the name of a global password file instead of the *password* and *context* arguments. On each line, the file may contain either a passcode, or a passcode and context, separated by a pipe character (`|`). If a context is not specified, the application defaults to the context named `disa`.

If the user login is successful, the application parses the dialed number in the specified *context*:

```
; allow outside callers to call 1-800 numbers, as long
; as they know the passcode. Set their Caller IDs to make
; it appear that they are dialing from within the company
[incoming]
exten => 123,1,DISA(4569,disa,"Company ABC" <(234) 123-4567>)
```

```
[disa]
exten => _1800NXXXXXX,1,Dial(Zap/4/${EXTEN})
```

## See Also

`Authenticate()`, `VMAuthenticate()`

## DumpChan()

`DumpChan()` — Dumps information about the calling channel to the console

## Synopsis

```
DumpChan([min_verbose_level])
```

Displays information about the calling channel, as well as a listing of all channel variables. If *min\_verbose\_level* is specified, output is displayed only when the verbosity level is currently set to that number or greater.



### Warning

If you have many channel variables set, `DumpChan ( )` will show only the first 1,024 characters of your channel variable listing.

```
exten => s,1,Answer()  
exten => s,2,DumpChan( )  
exten => s,3,Background(enter-ext-of-person)
```

## See Also

`NoOp()`, `Verbose()`

## EAGI()

`EAGI()`

## Synopsis

See `AGI ( )`.

## Echo()

`Echo()` — Echoes inbound audio back to the caller

## Synopsis

```
Echo ( )
```

Echoes audio read from the channel back to the channel. This application is often used to test the latency and voice quality of a VoIP link. The caller may press the # key to exit.

```
exten => 123,1,Echo( )  
exten => 123,2,Playback(vm-goodbye)
```

## See Also

`Milliwatt()`

## EndWhile()

`EndWhile()` — Ends a `while` loop

## Synopsis

```
EndWhile()
```

Returns to the previously called `While()` application. See `While()` for a complete description of how to use a `while` loop.

```
exten => 123,1,Set(COUNT=1)
exten => 123,2,While($[ ${COUNT} < 5 ])
exten => 123,3,SayNumber(${COUNT})
exten => 123,4,Set(COUNT=${COUNT} + 1)
exten => 123,5,EndWhile()
```

## See Also

`While()`, `ExitWhile()`, `GotoIf()`

## Exec()

`Exec()` — Executes an Asterisk application dynamically

## Synopsis

```
Exec(appname(arguments))
```

Allows an arbitrary application to be invoked even when not hard-coded in to the dialplan. Exits exactly the same as the underlying *application*, or abnormally, if the underlying *application* cannot be found. The *arguments* are passed to the called *application*.

This application allows you to dynamically call applications by pulling them from a database or other external source.

```
exten => 123,1,Set(MYAPP=SayDigits(12345))
exten => 123,2,Exec(${MYAPP})
```

## See Also

The `EVAL`, `TryExec()`, `ExecIf()`

## ExecIf()

`ExecIf()` — Conditionally executes an Asterisk application

## Synopsis

```
ExecIf(expression,application,arguments)
```

If *expression* is true, executes the given *application* with *arguments* as its arguments, and returns the result. For more information on Asterisk expressions, see the `channelvariables.txt` file in the *doc/* subdirectory of the Asterisk source.

If *expression* is false, execution continues at the next priority.



```
exten => 123,1,ExecIf($[ ${CALLERIDNUM} = 101 ],SayDigits,12345)
exten => 123,2,SayDigits(6789)
```

## See Also

The EVAL, Exec(), TryExec()

## ExitWhile()

ExitWhile() — Exit from a While ( ) loop, whether or not the conditional has been satisfied

## Synopsis

```
ExitWhile()
```

Will cause a While() loop to exit whether or not the conditional expression has been satisfied.

```
exten => 123,1,Set(COUNT=1)
exten => 123,n,While($[${COUNT} < 5])
exten => 123,n,GotoIf($[${COUNT} != 3]?continue)
exten => 123,n,ExitWhile()
exten => 123,n(continue),NoOp()
exten => 123,n,SayNumber(${COUNT})
exten => 123,n,Set(COUNT=${COUNT} + 1)
exten => 123,n,EndWhile()
```

## See Also

While(), ContinueWhile(), EndWhile()

## ExtenSpy()

ExtenSpy() — Listen to the audio on an extension, and optionally whisper to the calling channel

## Synopsis

```
ExtenSpy([exten@context[,options]])
```

This application is used to listen to the audio going to and from an Asterisk channel. Only channels created by outgoing calls for the specified extension will be selected for spying.

While a channel is being spied upon, the following actions may be performed:

- Dialing # cycles the volume level
- Dialing \* will cause the application to spy on the next available channel

The *options* parameter may contain zero or more of the following options:

b                      Only spy on channels that are involved in a bridged call.

<code>g(<i>group</i>)</code>	Only spy on channels that contain a channel variable named <code>SPYGROUP</code> , which should contain <i>group</i> in an optional colon-delimited list.
<code>q</code>	Quiet mode. Tells the application not to beep or read the selected channel's name when spying begins.
<code>r([<i>basename</i>])</code>	Records the channel audio to the monitor spool directory (usually <code>/var/spool/asterisk/monitor</code> ). An optional <i>basename</i> set the base filename of the recordings, which defaults to <code>chanspy</code> .
<code>v([<i>value</i>])</code>	Adjusts the volume of the audio being listened to. The <i>value</i> must be in the range of 4 to -4. A negative <i>value</i> will make the volume quieter, while a positive value will make it louder.
<code>w</code>	Whisper mode. This allows the spying channel to talk to the spied-upon channel, without any other bridged channel being able to hear the audio.
<code>W</code>	Private whisper mode. This enables the spying channel to speak to the spied-upon channel without being able to hear the audio from the spied-upon channel.

```
; Spy on channels created by extension 125 in the lab context
exten => 123,1,ExtenSpy(125@lab,w)
```

## See Also

ChanSpy()

## ExternalIVR()

ExternalIVR() — Interfaces with an external IVR application

## Synopsis

```
ExternalIVR(command[,arg1[,arg2...]])
```

Forks a process to run the specified ExternalIVR-compliant *command*, and starts a generator on the channel. The generator's play list is controlled by the external application, which can add and clear entries via simple commands issued over `STDOUT`. The external application will receive notifications of all DTMF events received on the channel, and notification if the channel is hung up. The application will not be forcibly terminated when the channel is hung up.

See `doc/externalivr.txt` in the Asterisk source code for the specification of the ExternalIVR interface.

```
; Run a test external IVR program, passing an argument
exten => 123,1,ExternalIVR(test_program,${MYARGUMENT})
```

## See Also

AGI()

## Flash()

Flash() — Flashes a Zap trunk

## Synopsis

```
Flash()
```

Sends a flash on a Zap channel. This is only a hack for people who want to perform transfers and other actions that require a flash via an AGI script. It is generally quite useless otherwise.

Returns 0 on success, or -1 if this is not a Zap trunk.

```
exten => 123,1,Flash()
```

## FollowMe()

FollowMe() — Find me/follow me functionality

## Synopsis

```
FollowMe(followmeid[,options])
```

This application attempts to locate the callee by dialing many different destinations either serially or in parallel, as defined in `followme.conf`.

The *followmeid* identifies the section of `followme.conf` that specifies how this callee should be found. The *options* parameter can be zero or more of the following:

- s Playback the incoming status message prior to starting the follow-me step(s)
- a Record the caller's name so it can be announced to the callee on each step
- n Playback the unreachable status message if we've run out of steps to reach the callee or the callee has elected not to be reachable

```
exten => 123,1,Answer()  
exten => 123,2,FollowMe(123,san)  
exten => 123,3,VoiceMail(123,u)
```

## ForkCDR()

ForkCDR() — Creates an additional CDR from the current call

## Synopsis

```
ForkCDR([options])
```

Creates an additional Call Detail Record for the remainder of the current call.

This application is often used in calling-card applications to distinguish the inbound call (the original CDR) from the billable call time (the second CDR).

If the `v` option is specified, all the CDR variables from the current record will be inherited by the new CDR record.

```
exten => 123,1,Answer()  
exten => 123,2,ForkCDR(v)  
exten => 123,3,Playback(tt-monkeys)  
exten => 123,4,Hangup()
```

## See Also

CDR function, NoCDR(), ResetCDR()

## GetCPEID()

GetCPEID() — Gets the CPE ID from an ADSI-capable telephone

## Synopsis

```
GetCPEID()
```

Obtains the CPE ID and other information and displays it on the Asterisk console. This information is often needed in order to properly set up `zapata.conf` for on-hook operations with ADSI-capable telephones.

Returns `-1` on hangup only.

```
; use this extension to get the necessary information to set up ADSI  
; telephones  
exten => 123,1,GetCPEID()
```

## See Also

ADSIProg(), `adsi.conf`, `zapata.conf`

## Gosub()

Gosub() — Branches to a new location, saving the return address

## Synopsis

```
Gosub(context,extension,priority)  
Gosub(extension,priority)  
Gosub(priority)
```

Branches to the location specified, similar to `Goto()`, except that `Gosub()` saves the return location, to be returned to later by invoking `Return()`.

## See Also

GosubIf(), Macro(), Goto(), Return(), StackPop()

# Gosublf()

Gosublf() — Conditionally branches to a new location, saving the return address

## Synopsis

```
GosubIf(condition?labeliftrue:labeliffalse)
```

Based upon the evaluation of *condition*, Gosub will branch execution either to *labeliftrue*( ) or *labeliffalse*. You may return to this same place in the dialplan by later calling Return.



### Tip

The word *label* is often used to denote that you may specify a *priority*; an *extension* and a *priority*; or a *context*, an *extension* and a *priority*. We use the word *label* to avoid having to spell out all of the possible options each time.

```
; Specify a default outgoing Caller*ID if one is not set by a specific channel.
exten => _NXXXXXX,1,GosubIf($["${CALLERID(num)}" = ""]?setcallerid,1)
exten => _NXXXXXX,n,Dial(Zap/g1/${EXTEN})
exten => _1NXXNXXXXXX,1,GosubIf($["${CALLERID(num)}" = ""]?setcallerid,1)
exten => _1NXXNXXXXXX,n,Dial(Zap/g1/${EXTEN})
exten => setcallerid,1,Set(CALLERID(num)=6152345678)
exten => setcallerid,n,Return
```

## See Also

Gosub(), Return(), MacroIf(), IF, GotoIf(),

# Goto()

Goto() — Sends the call to the specified priority, extension, and context

## Synopsis

```
Goto([[context],[extension],[priority])
Goto(named_priority)
```

Sends control of the current channel to the specified *priority*, optionally setting the destination *extension* and *context*.

Optionally, you can use the application to go to the named priority specified by the *named\_priority* argument. Named priorities work only within the current extension.

```
exten => 123,1,Answer()
exten => 123,2,Set(COUNT=1)
exten => 123,3,SayNumber(${COUNT})
exten => 123,4,Set(COUNT=${COUNT} + 1)
exten => 123,5,Goto(3)
```

```

; same as above, but using a named priority
exten => 124,1,Answer()
exten => 124,2,Set(COUNT=1)
exten => 124,3(repeat),SayNumber(${COUNT})
exten => 124,4,Set(COUNT=[ ${COUNT} + 1 ])
exten => 124,5,Goto(repeat)

```

## See Also

GotoIf(), GotoIfTime(), Gosub(), Macro()

## Gotolf()

GotoIf() — Conditionally goes to the specified priority

## Synopsis

```
GotoIf(condition?label1:label2)
```

Sends the call to *label1* if *condition* is true or to *label2* if *condition* is false. Either *label1* or *label2* may be omitted (in that case, we just don't take the particular branch), but not both.

A label can be any one of the following:

- A priority, such as 10
- An extension and a priority, such as 123,10
- A context, extension, and priority, such as incoming,123,10
- A named priority within the same extension, such as passed

Each type of label is explained in this example:

```

[globals]
; set TEST to something else besides 101 to see what GotoIf()
; does when the condition is false
TEST=101
;
[incoming]
; set a variable
; go to priority 10 if ${TEST} is 101, otherwise go to priority 20
exten => 123,1,GotoIf([ ${TEST} = 101 ]?10:20)
exten => 123,10,Playback(the-monkeys-twice)
exten => 123,20,Playback(tt-somethingwrong)
;
; same thing as above, but this time we'll specify an extension
; and a priority for each label
exten => 124,1,GotoIf([ ${TEST} = 101 ]?123,10:123,20)
;
; same thing as above, but these labels have a context, extension, and
; priority
exten => 125,1,GotoIf([ ${TEST} = 101 ]?incoming,123,10:incoming,123,20)

```

```

;
; same thing as above, but this time we'll go to named priorities
exten => 126,1,GotoIf($[ ${TEST} = 101 ]?passed:failed)
exten => 126,15(passed),Playback(the-monkeys-twice)
exten => 126,25(failed),Playback(the-monkeys-twice)

```

## See Also

Goto(), GotoIfTime(), GosubIf(), MacroIf()

## GotolfTime()

GotoIfTime() — Conditionally branches, depending on the time and day

## Synopsis

```
GotoIfTime(times,days_of_week,days_of_month,months?label)
```

Branches to the specified extension, if the current time matches the specified time. Each of the elements may be specified either as \* (for always) or as a range.

The arguments to this application are:

<i>times</i>	Time ranges, in 24-hour format
<i>days_of_week</i>	Days of the week (mon, tue, wed, thu, fri, sat, sun)
<i>days_of_month</i>	Days of the month (1-31)
<i>months</i>	Months (jan, feb, mar, apr, etc.)

```

; If we're open, then go to the open context
; We're open from 9am to 6pm Monday through Friday
exten => s,1,GotoIfTime(09:00-17:59,mon-fri,*,*?open,s,1)
;
; We're also late on Tuesday and Thursday
exten => s,n,GotoIfTime(09:00-19:59,tue&thru,*,*?open,s,1)
;
; We're also open from 9am to noon on Saturday
exten => s,n,GotoIfTime(09:00-11:59,sat,*,*?open,s,1)
;
; Otherwise, we're closed
exten => s,n,Goto(closed,s,1)

```

## See Also

GotoIf(), IFTIME

## Hangup()

Hangup() — Unconditionally hangs up the current channel

## Synopsis

```
Hangup(cause-code)
```

Unconditionally hangs up the current channel. If supported on the channel, *cause-code* will be specified to the remote end as the reason for ending the call. *cause-code* defaults to 16 (normal call clearing). Acceptable values for *cause-code* are the following:

- 16 Normal call clearing
- 17 Busy
- 19 No answer
- 21 Rejected
- 34 Congestion

```
exten => 123,1,Answer()
exten => 123,2,Playback(im-sorry)
exten => 123,3,Hangup()
```

## See Also

Answer(), Busy(), Congestion()

## HasNewVoicemail()

HasNewVoicemail() — Checks to see if there is new voicemail in the indicated voicemail box

## Synopsis

```
HasNewVoicemail(vmbox[@context][:folder][:varname[:options]])
```



### Note

The application has been deprecated in favor of the VMCOUNT() function.

Similar to HasVoicemail(). This application sets the VMSTATUS to 1 or 0, to indicate whether there is new (unheard) voicemail in the voicemail box indicated by *vmbox*. The *context* argument corresponds to the voicemail context, and *folder* corresponds to a voicemail folder. If the voicemail folder is not specified, it defaults to the INBOX folder. If the *varname* argument is present, HasNewVoicemail() assigns the number of messages in the specified folder to that variable.

If the *options* argument is set to the letter j, then Asterisk will send the call to priority n+101 if there is new voicemail.

```
; check to see if there's unheard voicemail in INBOX of mailbox 123
; in the default voicemail context
exten => 123,1,Answer()
exten => 123,n,HasNewVoicemail(123@default)
```



```

exten => 123,n,GotoIf($[${HASVMSTATUS} > 0]?newvm)
exten => 123,n,Playback(vm-youhave)
exten => 123,n,Playback(vm-no)
exten => 123,n,Playback(vm-messages)
exten => 123,n,Goto(done)
exten => 123,n(newvm),Playback(vm-youhave)
exten => 123,n,SayNumber(${HASVMSTATUS})
exten => 123,n,Playback(vm-INBOX)
exten => 123,n,Playback(vm-messages)
exten => 123,n(done),NoOp()

```

## See Also

HasVoicemail(), MailboxExists(), VMCOUNT

# HasVoicemail()

HasVoicemail() — Indicates whether there is voicemail in the indicated voicemail box

## Synopsis

```
HasVoicemail(vmbox[@context][:folder][:varname[,options]])
```

Sets the HASVMSTATUS channel variable to indicate whether there is voicemail in the voicemail box indicated by *vmbox*. The *context* argument corresponds to the voicemail context, and *folder* corresponds to a voicemail folder. If the folder is not specified, it defaults to the *INBOX* folder. If the *varname* argument is passed, this application assigns the number of messages in the specified folder to that variable.

If the *options* argument is set to the letter *j*, then Asterisk will send the call to priority *n*+101 if there is voicemail in the specified *folder*.

```

; check to see if there's any voicemail at all in INBOX of mailbox 123
; in the default voicemail context
exten => 123,1,Answer()
exten => 123,2,HasVoicemail(123@default,COUNT)
exten => 123,3,GotoIf(${VMSTATUS}?1000)
exten => 123,4,Playback(vm-youhave)
exten => 123,5,Playback(vm-no)
exten => 123,6,Playback(vm-messages)
exten => 123,1000,Playback(vm-youhave)
exten => 123,1001,SayNumber($COUNT)
exten => 123,1002,Playback(vm-messages)

```

## See Also

The HasVoicemail(), MailboxExists()

# IAX2Provision()

IAX2Provision() — Provisions a calling IAXy device

## Synopsis

```
IAX2Provision([template])
```

Provisions a calling IAXy device (assuming that the calling entity is an IAXy) with the given *template*. If no template is specified, the default template is used. IAXy provisioning templates are defined in the `iaxprov.conf` configuration file.

```
; provision IAXy devices with the default template when they dial this extension  
exten => 123,1,IAX2Provision(default)
```

## ImportVar()

ImportVar() — Sets a variable based on a channel variable from a different channel

## Synopsis

```
ImportVar(newvar=channel,variable)
```

Sets variable *newvar* to *variable* as evaluated on the specified *channel* (instead of the current channel). If *newvar* is prefixed with `_`, single inheritance is assumed. If prefixed with `_ _`, infinite inheritance is assumed.

```
; read the Caller ID information from channel Zap/1  
exten => 123,1,Answer()  
exten => 123,n,ImportVar(cidinfo=Zap/1,CALLERID(all))
```

## See Also

Set()

## Log()

Log() — Logs a custom message from the dialplan

## Synopsis

```
Log(level|message)
```

Sends a custom message to the logfiles from the dialplan. This application can be useful to log an exceptional condition to the logfiles, for later examination. *Level* may be one of the following:

DEBUG	Debugging message. This is generally not logged on a production system.
NOTICE	An informational message.
WARNING	A condition that may be serious, but is not a definite error.

ERROR     Something went terribly wrong.

## See Also

NoOp(), Verbose()

# LookupBlacklist()

LookupBlacklist() — Performs a lookup of a Caller ID name/number from the blacklist database

## Synopsis

```
LookupBlacklist([options])
```



### Note

This application has been deprecated in favor of `GotoIf($[${BLACKLIST()}]?context|extension|priority)`

Looks up the Caller ID number on the active channel in the Asterisk database (family `blacklist`). If the Caller ID number is found in the blacklist, Asterisk sets the `LOOKUPBLSTATUS` channel variable to `FOUND`. Otherwise, the variable is set to `NOTFOUND`.

If the `j` option is used in the `options` parameter, and the number is found, and if there exists a priority `n+101` (where `n` is the priority of the current instance), the channel will be set up to continue at that priority level.

To add to the blacklist from the Asterisk CLI, type **database put blacklist name / number**.

```
; send blacklisted numbers to an endless loop
; otherwise, dial the number defined by the variable ${JOHN}
exten => 123,1,Answer()
exten => s,2,LookupBlacklist()
; if the Caller ID number is found in the blacklist, jump to the "goaway" label
exten => 123,n,GotoIf("${LOOKUPBLSTATUS}" = "FOUND"?goaway)
; otherwise, go ahead and call John
exten => 123,n,Dial("${JOHN}")
exten => 123,n(goaway),Busy(5)
exten => 123,n,Hangup()
```

## See Also

BLACKLIST

# LookupCIDName()

LookupCIDName() — Performs a lookup of a Caller ID name from the AstDB

## Synopsis

```
LookupCIDName( )
```



## Note

This application has been deprecated in favor of `Set(CALLERID(name)={DB(cidname/{CALLERID(num)})})`

Uses the Caller ID number on the active channel to retrieve the Caller ID name from the AstDB (family `cidname`). This application does nothing if no Caller ID was received on the channel. This is useful if you do not subscribe to Caller ID name delivery, or if you want to change the Caller ID names on some incoming calls.

```
; look up the Caller ID information from the AstDB, and pass it along
; to Jane's phone
exten => 123,1,Answer()
exten => 123,2,LookupCIDName()
exten => 123,3,Dial(SIP/Jane)
```

## See Also

DB

## Macro()

Macro() — Calls a previously defined dialplan macro

## Synopsis

```
Macro(macroname,arg1,arg2...)
```

Executes a macro defined in the context named `macro-macroname`, jumping to the `s` extension of that context and executing each step, then returning when the steps end.

The calling extension, context, and priority are stored in `{MACRO_EXTEN}`, `{MACRO_CONTEXT}`, and `{MACRO_PRIORITY}`, respectively. Arguments *arg1*, *arg2*, etc. become `{ARG1}`, `{ARG2}`, etc. in the macro context.

Macro() exits abnormally if any step in the macro exited abnormally or indicated a hangup. If `{MACRO_OFFSET}` is set at termination, this application will attempt to continue at priority `MACRO_OFFSET+n+1` if such a step exists, and at `n+1` otherwise. (In both cases, `n` stands for the current priority.)

If you call the `Goto()` application inside of the macro to specify a context outside of the currently executing macro, the macro will terminate and control will go to the destination of the `Goto()`.

```
; define a macro to count down from the specified value
[macro-countdown]
exten => s,1,Set(COUNT={ARG1})
exten => s,2,While([ {COUNT} > 0 ])
exten => s,3,SayNumber({COUNT})
exten => s,4,Set(COUNT=[ {COUNT} - 1 ])
exten => s,5,EndWhile()

; call our macro with two different values
[example]
exten => 123,1,Macro(countdown,10)
```

```
exten => 124,1,Macro(countdown,5)
```



### Note

While a macro is being executed, it becomes the current context. This means that if a hangup occurs, for instance, the macro will be searched for an `h` extension, *not* the context from which the macro was called. So, make sure to define all appropriate extensions in your macro (you can use `catch` in AEL).



### Warning

Because of the way `Macro()` is implemented (it executes the priorities contained within it via sub-engine), and a fixed per-thread memory stack allowance, macros are limited to seven levels of nesting (macro calling macro calling macro, etc.); It may be possible that stack-intensive applications in deeply nested macros could cause Asterisk to crash earlier than this limit.

## See Also

`MacroExit()`, `Goto()`, `Gosub()`

## MacroExclusive()

`MacroExclusive()` — Runs a macro, exclusive of any other channel

## Synopsis

```
MacroExclusive(macroname[ ,arguments])
```

Runs the specified macro, but ensures that only one channel is running inside that macro at one time. If another channel is already executing that macro, then `MacroExclusive()` will pause the channel until that channel has exited the macro.

## See Also

`Macro()`

## MacroExit()

`MacroExit()` — Explicitly returns from a macro

## Synopsis

```
MacroExit()
```

Explicitly return from a macro. Normally, `Macro()` automatically exits when it runs out of priorities. `MacroExit()` provides a method by which a macro may be terminated early.

## See Also

`Macro()`

# MacroIf()

MacroIf() — Conditionally calls a previously defined macro

## Synopsis

```
MacroIf(condition?macroiftrue, args:macroiffalse, args)
```

Evaluates *condition*, then executes one of either *macroiftrue* or *macroiffalse*. Once beyond the *condition*, however, note that MacroIf() behaves identically to Macro().

```
; define a macro to count down from the specified value
[macro-countdown]
exten => s,1,Set(COUNT=${ARG1})
exten => s,2,While($[ ${COUNT} > 0])
exten => s,3,SayNumber(${COUNT})
exten => s,4,Set(COUNT=${COUNT} - 1)
exten => s,5,EndWhile()

; define a macro to count up to the specified value
[macro-countup]
exten => s,1,Set(COUNT=1)
exten => s,2,While($[ ${COUNT} < ${ARG1}])
exten => s,3,SayNumber(${COUNT})
exten => s,4,Set(COUNT=${COUNT} + 1)
exten => s,5,EndWhile()

; call our macro with two different values
[example]
exten => 123,1,MacroIf($[ ${foo} < 5 ]?countup,${foo}:countdown,${foo})
```

## See Also

GotoIf(), GosubIf(), IF

# MailboxExists()

MailboxExists() — Conditionally branches if the specified voicemail box exists

## Synopsis

```
MailboxExists(mailbox[@context[,options]])
```

Checks to see if the mailbox specified by the *mailbox* argument exists in the Asterisk voicemail system. You may pass a voicemail *context* if the mailbox is not in the default voicemail context.

This application sets a channel variable named VMBOXEXISTSSTATUS. If the mailbox exists, it will be set to SUCCESS. Otherwise, it will be set to FAILED.

If the *j* option is passed as the *options* parameter, the application will jump to priority *n*+101 (where *n* is the current priority) if the voicemail box specified by the *mailbox* argument exists.

```

exten => 123,1,Answer()
exten => 123,n,Set(MYMAILBOX=123@default)
exten => 123,n,MailboxExists(${MYMAILBOX})
exten => 123,n,GotoIf($["${VMBOXEXISTSSTATUS}" = "SUCCESS"]?exists)
exten => 123,n,Playback(im-sorry)
exten => 123,n,Hangup()
exten => 123,n(exists),Voicemail(u123)

```

## See Also

HasVoicemail(), HasNewVoicemail()

## MeetMe()

MeetMe() — Puts the caller in to a MeetMe conference bridge

## Synopsis

```
MeetMe([confno[,options[,PIN]]])
```

Places the caller in to the audio conference bridge specified by the *confno* argument. If the conference number is omitted, the user will be prompted to enter one.

If the *PIN* argument is passed, the caller must enter that PIN number to successfully enter the conference.

The *options* string may contain zero or more of the characters in the following list:

- |   |  |
|---|--|
| a | Sets admin mode.   |
| A | Sets marked mode.  |
| b | Runs the AGI script specified in <code>\${MEETME_AGI_BACKGROUND}</code> ; default: <i>conf-background.agi</i> . (Note: this does not work with non-Zap channels in the same conference.) |
| c | Announces user(s) count upon joining a conference.   |
| d | Dynamically adds conference.   |
| D | Dynamically adds conference, prompting for a PIN.  |
| e | Selects an empty conference.   |
| E | Selects an empty Pinless conference.   |
| F | Passes DTMF digits through the conference to other participants. DTMF digits used to enable conference features will not be passed through.  |
| i | Announces user join/leave with review.   |
| I | Announces user join/leave without review.  |
| l | Sets listen only mode (listen only, no talking).   |
| m | Sets the participant as initially muted.   |

M	Enables music on hold when the conference has a single caller.
o	Turns on talker optimization. With talker optimization, Asterisk treats talkers who aren't speaking as being muted, meaning that no encoding is done on transmission and that received audio that is not registered as talking is omitted, causing no buildup in background noise.
p	Allows user to exit the conference by pressing #.
P	Always prompts for the PIN even if it is specified.
q	Sets quiet mode. In quiet mode, Asterisk won't play sounds as conference participants enter or leave.
r	Records conference (as <code>\${MEETME_RECORDINGFILE}</code> using format <code>\${MEETME_RECORDINGFORMAT}</code> ). The default filename is <i>meetme-conf-rec-\${CONFNO}-\${UNIQUEID}</i> and the default format is <i>.wav</i> .
s	Presents the menu (user menu or admin menu, depending on whether the caller is marked as an administrator) when <code>*</code> is received.
t	Sets talk-only mode (talk only, no listening).
T	Sets talker detection. Asterisk will send events on the Manager Interface identifying the channel that is talking. The talker will also be identified on the output of the <code>meetme list</code> CLI command.
w[ (seconds) ]	Waits for a marked admin to join the conference. If <i>seconds</i> is not specified, the conference will wait indefinitely for the admin to join. If <i>seconds</i> is specified, the conference will wait the specified number of seconds. If the admin still hasn't joined, the call will continue on with the next priority in the dialplan.
x	Closes the conference when the last marked user exits.
X	Allows user to exit the conference by entering a valid single-digit extension (set via the variable <code>\${MEETME_EXIT_CONTEXT}</code> ), or the number of an extension in the current context if that variable is not defined.
1	Doesn't play initial message when the first person joins the conference.

```

exten => 123,1,Answer()
; add the caller to conference number 501 with pin 1234
exten => 123,2,MeetMe(501,DpM,1234)

```



### Warning

A suitable Zaptel timing interface must be installed for MeetMe conferencing to work.

## See Also

MeetMeAdmin(), MeetMeCount()

## MeetMeAdmin()

MeetMeAdmin() — Performs MeetMe conference administration



## Synopsis

```
MeetMeAdmin(confno,command[,user])
```

Runs the specified MeetMe administration *command* on the specified conference. On some commands, you may specify the *user* on which to run the specified command. The *command* may be one of the following:

- e Ejects the last user that joined.
- k Kicks the specified *user* out of the conference.
- K Kicks all users out of the conference.
- l Unlocks the conference.
- L Locks the conference.
- m Unmutes the specified *user*.
- M Mutes the specified *user*.
- n Unmutes the entire conference.
- N Mutes all non-admin participants in the conference.
- r Resets the volume settings for the specified *user*.
- R Resets the volume settings for all participants.
- s Lowers the speaking volume for the entire conference.
- S Raises the speaking volume for the entire conference.
- t Lowers the speaking volume for the specified *user*.
- T Raises the speaking volume for the specified *user*.
- u Lowers the listening volume for the specified *user*.
- U Raises the listening volume for the specified *user*.
- v Lowers the listening volume for the entire conference.
- V Raises the listening volume for the entire conference.

```
; mute conference 501
exten => 123,1,MeetMeAdmin(501,N)

; kick user 1234 from conference 501
exten => 124,1,MeetMeAdmin(501,k,1234)
```



### Tip

You can find a list of users in the conference by using the `meetme list` command from the Asterisk CLI, or by using the Asterisk Manager Interface.

## See Also

MeetMe(), MeetMeCount()

## MeetMeCount()

MeetMeCount() — Counts the number of participants in a MeetMe conference

## Synopsis

```
MeetMeCount(confno[,variable])
```

Plays back the number of users in the MeetMe conference identified by *confno*. If a variable is specified by the *variable* argument, playback will be skipped and the count will be assigned to *variable*.

```
; count the number of users in conference 501, and assign that number  
to ${COUNT}  
exten => 123,1,MeetMeCount(501,COUNT)
```

## See Also

MeetMe(), MeetMeAdmin()

## Milliwatt()

Milliwatt() — Generates a 1,000 Hz tone

## Synopsis

```
Milliwatt()
```

This application generates a constant 1,000 Hz tone at 0 dbm (µlaw). This application is often used for testing the audio properties of a particular channel.

```
; generate a 1000HZ tone  
exten => 123,1,Milliwatt()
```



### Warning

Please note that there is a service that the carriers use to test circuits for loss, that folks in the industry have nicknamed “1,000-cycles.” The thing is, the tone the carrier equipment sends is actually 1,004 Hz, so if you want to test circuit loss on an analog channel from Asterisk, `Milliwatt()` may not give you exactly what you want.

## See Also

Echo(), Playtones()

# MixMonitor()

MixMonitor() — Records a channel in the background, mixing both directions synchronously

## Synopsis

```
MixMonitor(filename.ext,options,command)
```

Records the audio on the current channel to the specified file. If the filename is an absolute path, MixMonitor() uses that path; otherwise it creates the file in the configured monitoring directory from asterisk.conf.

If *command* is specified, it will be run when recording ends, either by hangup or by calling StopMixMonitor().

The *options* parameter can contain zero or more of the following options:

- a Append to the file, instead of overwriting it.
- b Only save audio when the channel is bridged.



### Note

This does not include conferences or sounds played to each bridged party.

v(x) Adjust the heard volume by a factor of *x* (range -4 to 4).

V(x) Adjust the spoken volume by a factor of *x* (range -4 to 4).

W(x) Adjust both the heard and the spoken volumes by a factor of *x* (range -4 to 4).

```
; Record channel
exten => 123,1,MixMonitor(/var/lib/asterisk/sounds/123.wav)
```

## See Also

Monitor(), StopMixMonitor(), PauseMonitor(), UnpauseMonitor()

# Monitor()

Monitor() — Monitors (records) the audio on the current channel

## Synopsis

```
Monitor([file_format[:urlbase]][,fname_base][,options])
```

Starts monitoring a channel. The channel's input and output voice packets are logged to files until the channel hangs up or monitoring is stopped by the StopMonitor() application.

Monitor() takes the following arguments:

*file\_format* Specifies the file format. If not set, defaults to wav.

*fname\_base* If set, changes the filename used to the one specified.

*options*

One of two options can be specified:

- m When the recording ends, mix the two leg files in to one and delete the original leg files. If the variable `${MONITOR_EXEC}` is set, the application referenced in it will be executed instead of *soxmix*, and the raw leg files will *not* be deleted automatically. *soxmix* (or `${MONITOR_EXEC}`) is handed three arguments: the two leg files and the filename for the target mixed file, which is the same as the leg filenames but without the in/out designator. If `${MONITOR_EXEC_ARGS}` is set, the contents will be passed on as additional arguments to `${MONITOR_EXEC}`. Both `${MONITOR_EXEC}` and the `m` flag can be set from the administrator interface.
- b Don't begin recording unless a call is bridged to another channel.

```
exten => 123,1,Answer()  
; record the current channel, and mix the audio channels at the end of  
; recording  
exten => 123,2,Monitor(wav,monitor_test,mb)  
exten => 123,3,SayDigits(12345678901234567890)  
exten => 123,4,StopMonitor()
```

## See Also

ChangeMonitor(), StopMonitor(), MixMonitor(), PauseMonitor(), UnpauseMonitor()

## MorseCode()

MorseCode() — Plays Morse code

## Synopsis

MorseCode(*string*)

Plays the *string*, encoded in International Morse Code. The following channel variables will affect the playback:

MORSEDTLEN The length, in milliseconds, of a DIT. Defaults to 80 ms.



### Note

All of the other tone and silence lengths are defined in the International Morse Code standard with respect to the length of a DIT, and therefore, each of the other lengths will be adjusted suitably.

MORSETONE The tone, in Hertz (Hz), which will be used. Defaults to 800 Hz.

```
; dah-dit-dah dit-dit dit-dit-dit-dit-dah dah-dit-dah dit-dit-dah dit-dah  
exten => 123,1,Answer()  
exten => 123,2,MorseCode(KI4KUA)
```

## See Also

SayAlpha(), SayPhonetic()

# MP3Player()

MP3Player() — Plays an MP3 file or stream

## Synopsis

```
MP3Player(location)
```

Uses the mpg123 program to play the given *location* to the caller. The specified *location* can be either a filename or a valid URL. The caller can exit by pressing any key.



### Warning

The correct version of mpg123 must be installed for this application to work properly. Asterisk currently works best with mpg123-0.59r. Other versions may give less than desirable results.

```
exten => 123,1,Answer()  
exten => 123,2,MP3Player(test.mp3)  
  
exten => 123,1,Answer()  
exten => 123,2,MP3Player(http://example.com/test.mp3)
```

# MusicOnHold()

MusicOnHold() — Plays music on hold indefinitely

## Synopsis

```
MusicOnHold(class)
```

Plays hold music specified by *class*, as configured in `musiconhold.conf`. If omitted, the default music class for the channel will be used. You can use the MUSICCLASS dialplan function to set the default music class for the channel.

```
; transfer telemarketers to this extension to keep them busy  
exten => 123,1,Answer()  
exten => 123,n,Playback(tt-allbusy)  
exten => 123,n,MusicOnHold(default)
```

## See Also

SetMusicOnHold(), WaitMusicOnHold(), MUSICCLASS

# NBScat()

NBScat() — Plays an NBS local stream

## Synopsis

```
NBScat( )
```

Uses the `nbscat8k` program to listen to the local Network Broadcast Sound (NBS) stream. (For more information, see the `nbs` module in Digium's Subversion server.) The caller can exit by pressing any key.

Returns `-1` on hangup; otherwise, does not return.

```
exten => 123,1,Answer()  
exten => 123,2,NBScat()
```

## NoCDR()

NoCDR() — Disables Call Detail Records for the current call

## Synopsis

```
NoCDR( )
```

Disables CDRs for the current call.

```
; don't log calls to 555-1212  
exten => 5551212,1,Answer()  
exten => 5551212,2,NoCDR()  
exten => 5551212,3,Dial(Zap/4/5551212)
```

## See Also

AppendCDRUserField(), ForkCDR(), SetCDRUserField()

## NoOp()

NoOp() — Does nothing

## Synopsis

```
NoOp( text )
```

Does nothing—this application is simply a placeholder. This application is often used as a debugging tool. Whenever Asterisk's core verbosity level is set to 3 or above, Asterisk evaluates and prints each line of the dialplan before executing it. This means that any arguments passed to the `NoOp( )` application (in the `text` parameter) are printed to the console. By watching the console output, a skilled Asterisk administrator can use this output to debug problems in the dialplan.

```
exten => 123,1,NoOp(CallerID is ${CALLERID})
```



## Tip

You don't have to place quotes around the text. If quotes are placed within the brackets, they will show up on the console.

### When to use NoOp() and Verbose()

The difference between `Verbose()` and `NoOp()` is subtle. Here are some suggestions as to how you can discern when to use each. The `Verbose()` application is handy when you want to output something to the console. Using the `set verbose` command (followed by the level of verbosity you desire—0 to 4), you can set the output to a level that will not show you all of the activity on the system, but rather only those things that are equal to or less than that the current level. (Actually, you can set the verbosity to whatever you want. The `set verbose 999` will work just fine, but we have not been able to find anything that outputs at a level higher than 4, so anything from 4 to infinity will be effectively the same for the time being.) This means that you can display all kinds of information pertaining to a section of code you are testing, without having to see all of the other activity in the system. If you set the following in your dial plan:

```
exten => _X.,n,Verbose(2, ${SOME_VAR})
```

You can then use the CLI to set the verbosity to 2 or less (`core set verbose 2`), and you will see output from the various calls to `Verbose()`, but very little else.

Read the section on `Verbose()` later in this appendix for more on how to use it. The `NoOp()` application is best used as a place holder. For example, if you are setting a `Goto()` point in your dialplan that is using a priority label, you can use `NoOp()` as the destination point for that goto. For example,

```
exten => _X.,n(call_forward),NoOp()
```

is an excellent marker for pointing a jump in your dialplan to a spot. From that point you can carry on with whatever logic you wanted to apply to that part of the extension (judging by the label, it'd have something to do with call forwarding). The value of the `NoOp()` is that when you don't really know what sort of things you might want to move around in relation to what follows that label, you can be sure you'll never have to recode the label itself. It will never do anything other than provide a destination for the `Goto()`, so you can put it wherever you like and be sure it won't introduce any unexpected behavior.

If this seems confusing, it is due to our inability to describe it right. Experiment with `Verbose()` and `NoOp()` all in your dialplan (you can use them anywhere), and you will quickly gain an understanding of how they can help you (especially if you are like us and cause a lot of syntax errors).

## See Also

`Verbose()`, `Log()`

## Page()

`Page()` — Opens one-way audio to multiple phones

## Synopsis

```
Page(Tech/chan1[&Tech/chan2][...][&Tech/chanN][options])
```

Places outbound calls to the given technology/resource and dumps them in to a conference bridge as muted participants. The original caller is dumped in to the conference as a speaker, and the room is destroyed when the original caller leaves. The following options may be specified:

- d Full duplex audio. Allow the paged persons to respond to the caller.
- q Quiet. Do not play a beep to the caller.
- r Record the page. See the `r` option to `MeetMe` for more information.

```
exten => 123,1,Page(SIP/101&SIP/102&IAX2/iaxy123)
```

## See Also

`MeetMe()`

## Park()

`Park()` — Parks the current call

## Synopsis

```
Park()
```

Parks the current call (typically in combination with a supervised transfer to determine the parking space number). This application is always registered internally and does not need to be explicitly added in to the dialplan, although you should include the `parkedcalls` context. Parking configuration is set in `features.conf`.

```
; explicitly park the caller
include => parkedcalls
exten => 123,1,Answer()
exten => 123,n,Park()
```

## See Also

`ParkAndAnnounce()`, `ParkedCall()`

## ParkAndAnnounce()

`ParkAndAnnounce()` — Parks the current call and announces the call over the specified channel

## Synopsis

```
ParkAndAnnounce(template,timeout,channel[,return_context])
```



Parks the current call in the parking lot and announces the call over the specified *channel*. The *template* is a colon-separated list of files to announce; the word PARKED is replaced with the parking space number of the call. The *timeout* argument is the time in seconds before the call returns to the *return\_context*. The *channel* argument is the channel to call to make the announcement. *Console/dsp* calls the console. The *return\_context* argument is a *Goto()*-style label to jump the call back in to after timeout, which defaults to *n+1* (where *n* is the current priority) in the *return\_context* context.

```
include => parkedcalls
exten => 123,1,Answer()
exten => 123,2,ParkAndAnnounce(vm-youhave:a:pbx-transfer:at:vm-extension:PARKED,120,
Console/dsp)
exten => 123,3,Playback(vm-nobodyavail)
exten => 123,4,Playback(vm-goodbye)
exten => 123,5,Hangup()
```

## See Also

Park(), ParkedCall()

## ParkedCall()

ParkedCall() — Answers a parked call

## Synopsis

```
ParkedCall(parkingslot)
```

Connects the caller to the parked call in the parking space identified by *parkingslot*. This application is always registered internally and does not need to be explicitly added in to the dialplan, although you should include the *parkedcalls* context.

```
; pick up the call parked in parking space 701
exten => 123,1,Answer()
exten => 123,2,ParkedCall(701)
```

## See Also

Park(), ParkAndAnnounce()

## PauseMonitor()

PauseMonitor() — Suspends monitoring of a channel

## Synopsis

```
PauseMonitor()
```

Temporarily suspends the monitoring (recording) of the current channel

```

exten => 123,1,Answer()
exten => 123,n,Monitor(wav,monitor_test)
exten => 123,n,Playback(demo-congrats)
    ; temporarily pause the monitoring while we gather some secret info
exten => 123,n,PauseMonitor()
exten => 123,n,Read(NEWPASS,vm-newpassword)
exten => 123,n,SayDigits(${NEWPASS})
exten => 123,n,UnpauseMonitor()
exten => 123,n,Dial(${JOHN})

```

## See Also

Monitor(), StopMonitor(), UnpauseMonitor()

## PauseQueueMember()

PauseQueueMember() — Temporarily blocks a queue member from receiving calls

## Synopsis

```
PauseQueueMember([queuename],interface[,options])
```

Pauses the specified queue *interface*. This prevents any calls from being distributed from the queue to the *interface* until it is unpaused by the `UnpauseQueueMember()` application or the Manager Interface. If no *queue-name* is given, the interface is paused in every queue it is a member of.

This application sets a channel variable named PQMSTATUS to either PAUSED or NOTFOUND upon completion.

If the *options* parameter is set to `j` and the *interface* is not in the named queue, or if no queue is given and the *interface* is not in any queue, it will jump to priority `n+101` (where `n` is the current priority), if it exists.

```

exten => 123,1,PauseQueueMember(,SIP/300)
exten => 124,1,UnpauseQueueMember(,SIP/300)

```

## See Also

UnpauseQueueMember()

## Pickup()

Pickup() — Answers a ringing call from another phone

## Synopsis

```
Pickup(extension[@context][&extension2[@context2][...]])
```

Picks up any ringing channel that is ringing the specified *extension*. If multiple extensions are specified, `Pickup()` will retrieve the first matching call found. If no *context* is specified, the current context will be used.

There is also a special *context* name of PICKUPMARK. If specified, Pickup will find the first ringing channel with the channel variable PICKUPMARK set, with a value corresponding to the value of *extension*.

## Playback()

Playback() — Plays the specified audio file to the caller

### Synopsis

```
Playback(filename[&filename2...][,options])
```

Plays back a given filename to the caller. The filename should not contain the file extension, as Asterisk will automatically choose the audio file with the lowest conversion cost. Zero or more *options* may also be included. The *skip* option causes the playback of the message to be skipped if the channel is not in the “up” state (i.e., it hasn’t yet been answered). If *skip* is specified, the application will return immediately should the channel not be off-hook. Otherwise, unless *noanswer* is specified, the channel will be answered before the sound file is played. (Not all the channels support playing messages while still on-hook.) If *j* is passed as one of *options* and the file does not exist, this application jumps to priority *n*+101 (where *n* is the current priority), if it exists.

```
exten => 123,1,Answer()  
exten => 123,n,Playback(tt-weasels)
```

### See Also

Background(), ControlPlayback()

## Playtones()

Playtones() — Plays a tone list

### Synopsis

```
Playtones(tonelist)
```

Plays a tone list. Execution immediately continues with the next step, while the tones continue to play. The *tonelist* is either the tone name defined in the *indications.conf* configuration file, or a specified list of frequencies and durations. See *indications.conf* for a description of the specification of a tone list.

Use the `StopPlaytones()` application to stop the tones from playing.

```
; play a busy signal for two seconds, and then a congestion tone  
for two seconds  
exten => 123,1,Playtones(busy)  
exten => 123,2,Wait(2)  
exten => 123,3,StopPlaytones()  
exten => 123,4,Playtones(congestion)  
exten => 123,5,Wait(2)  
exten => 123,6,StopPlaytones()
```

```
exten => 123,7,Goto(1)
```

## See Also

StopPlaytones(), indications.conf, Busy(), Congestion(), Progress(), Ringing()

# PrivacyManager()

PrivacyManager() — Requires a caller to enter his phone number, if no Caller ID information is received

## Synopsis

```
PrivacyManager([maxretries[,minlength[,options]])
```

If no Caller ID is received, this application answers the channel and asks the caller to enter his phone number. By default, the caller is given three attempts. `PrivacyManager()` sets a channel variable named `PRIVACYMGRSTATUS` to either `SUCCESS` or `FAILURE`. If Caller ID is received on the channel, `PrivacyManager()` does nothing.

If the *options* parameter is set to *j* and the caller fails to enter his Caller ID number, the call will continue at priority *n*+101 (where *n* is the current priority).

The `privacy.conf` configuration file changes the functionality of the `PrivacyManger()` application. It contains the following two lines:

`maxretries` Specifies the maximum number of attempts the caller is allowed to input a Caller ID number (default: 3).

`minlength` Specifies the minimum allowable digits in the input Caller ID number (default: 10).

The *maxretries* and *minlength* settings may also be passed to the application as parameters. Parameters passed to the application override any settings in `privacy.conf`.

```
exten => 123,1,Answer()
exten => 123,n,PrivacyManager()
exten => 123,n,GotoIf($["${PRIVACYMGRSTATUS}" = "FAILURE"]?bad)
exten => 123,n,Dial(Zap/1)
exten => 123,n,Hangup()
exten => 123,n(bad),Playback(im-sorry)
exten => 123,n,Playback(vm-goodbye)
exten => 123,n,Hangup()
```

## See Also

Zapateller()

# Progress()

Progress() — Indicates progress

## Synopsis

```
Progress()
```

Requests that the channel indicate that in-band progress is available to the user. Each channel type in Asterisk has its own way of signaling progress on the call.

```
; indicate progress to the calling channel, wait 5 seconds,  
; and then answer the call  
exten => 123,1,Progress()  
exten => 123,n,Wait(5)  
exten => 123,n,Answer()
```

## See Also

Busy(), Congestion(), Ringing(), Playtones()

## Queue()

Queue() — Places the current call in to the specified call queue

## Synopsis

```
Queue(queuename[, options[, URL[, announceoverride[, timeout[, AGI]]]]])
```

Places an incoming call in to the call queue specified by *queuename*, as defined in `queues.conf`.

The *options* argument may contain zero or more of the following characters:

- d Specifies a data-quality (modem) call (minimum delay).
- h Allows callee to hang up by hitting \*.
- H Allows caller to hang up by hitting \*.
- i Ignores call forward requests from queue members and does nothing when they are requested.
- n Disallows retries on the timeout; exits this application and goes to the next step.
- r Rings instead of playing music on hold.
- t Allows the called user to transfer the call.
- T Allows the calling user to transfer the call.
- w Allows the called user to write the conversation to disk.
- W Allows the calling user to write the conversation to disk.

In addition to being transferred, a call may be parked and then picked up by another user.

The *announceoverride* argument overrides the standard announcement played to queue agents before they answer the specified call.

The optional *URL* will be sent to the called party if the channel supports it.

The *timeout* will cause the queue to fail out after a specified number of seconds, checked between each `queues.conf timeout` and *retry* cycle. The call will continue on with the next priority in the dialplan.

This application sets a channel variable named `QUEUESTATUS` upon completion. It will take one of the following values:

<code>TIMEOUT</code>	The call was in the queue too long, and timed out. See the <i>timeout</i> parameter.
<code>FULL</code>	The queue was already full. See the <code>maxlen</code> setting for the queue in <code>queues.conf</code> .
<code>JOINEMPTY</code>	The caller could not join the queue, as there were no queue members to answer the call. See the <code>joinempty</code> setting for the queue in <code>queues.conf</code> .
<code>LEAVEEMPTY</code>	The caller joined the queue, but then all queue members left. See the <code>leavewhenempty</code> setting for the queue in <code>queues.conf</code> .
<code>JOINUNAVAIL</code>	The caller could not join the queue, as there were no queue members available to answer the call. See the <code>joinempty</code> setting for the queue in <code>queues.conf</code> .
	The caller joined the queue, but then all of the queue members became unavailable. See the <code>leavewhenempty</code> setting for the queue in <code>queues.conf</code> .

```
; place the caller in the techsupport queue
exten => 123,1,Answer()
exten => 123,2,Queue(techsupport,t)
```

## See Also

`AddQueueMember()`, `RemoveQueueMember()`, `PauseQueueMember()`, `UnpauseQueueMember()`, `AgentLogin()`, `queues.conf`, `QUEUE_MEMBER_COUNT`, `QUEUE_MEMBER_LIST`, `QUEUE_WAITING_COUNT`

## QueueLog()

`QueueLog()` — Writes arbitrary queue events to the queue log

## Synopsis

```
QueueLog(queueName,uniqueid,member,event[,additionalInfo])
```

Writes an arbitrary queue event to the queue log. The *queueName* parameter specifies the name of the queue. The *uniqueid* parameter specifies the unique identifier for the channel. The *member* parameter specifies which queue member the event pertains to. The *event* and *additionalInfo* parameters may be set to arbitrary data, as needed.

```
; Write an arbitrary event to the queue log
exten => 123,1,QueueLog(myqueue,${UNIQUEID},Agent/123,MyTestEvent)
```

## See Also

`Queue()`

# Random()

Random() — Conditionally branches, based upon a probability

## Synopsis

```
Random([probability]:[[context],extension],priority)
```



### Note

This application has been deprecated in favor of:

```
GotoIf(${$RAND(1,100)} > num)?label)
```

Conditionally jumps to the specified *priority* (and optional *extension* and *context*), based on the specified *probability*. *probability* should be an integer between 1 and 100. The application will jump to the specified destination *priority* percent of the time.

```
; choose a random number over and over again
exten => 123,1,SayNumber(${RAND(1|10)})
exten => 123,n,Goto(1)
```

## See Also

RAND

# Read()

Read() — Reads DTMF digits from the caller and assigns the result to a variable

## Synopsis

```
Read(variable[,filename[,maxdigits[,option[,attempts[,timeout]]]]])
```

Reads a #-terminated string of digits from the user in to the given *variable*.

Other arguments include:

*filename* Specifies the file to play before reading digits.

*maxdigits* Sets the maximum acceptable number of digits. If this argument is specified, the application stops reading after *maxdigits* have been entered (without requiring the user to press the # key). Defaults to 0 (no limit, wait for the user to press the # key). Any value below 0 means the same. The maximum accepted value is 255.

*option* Zero or more of the following options:

- s Return immediately if the line is not answered.
- i Interpret the filename as an indication tone setting from `indications.conf`.

*n* Read digits even if the line has not been answered.

*attempts* If greater than 1, that many attempts will be made in the event that no data is entered.

*timeout* If greater than 0, that value will override the default timeout.

```
; read a two-digit number and repeat it back to the caller
exten => 123,1,Read(NUMBER,,2)
exten => 123,2,SayNumber(${NUMBER})
exten => 123,3,Goto(1)
```

## See Also

SendDTMF()

## ReadFile()

ReadFile() — Reads the contents of a file in to a variable

## Synopsis

```
ReadFile(variable=filename,length)
```

ReadFile captures the contents of *filename*, with a maximum size of *length*.

```
; read the first 80 characters of a file in to a variable
exten => 123,1,Answer()
exten => 123,n,ReadFile(TEST=/tmp/test.txt,80)
exten => 123,n,SayAlpha(${TEST})
```

## See Also

System(), Read()

## RealTime

RealTime — Looks up information from the RealTime configuration handler

## Synopsis

```
RealTime(family,colmatch,value[,prefix])
```

Uses the RealTime configuration handler system to read data in to channel variables. All unique column names (from the specified *family*) will be set as channel variables, with an optional *prefix* to the name (e.g., a prefix of *var\_* would make the column name become the variable `${var_name}`).

```
; retrieve all columns from the sipfriends table where the name column
; matches "John", and prefix all the variables with "John_"
```



```

exten => 123,1,RealTime(sipfriends,name,John,John_)
; now, let's read the value of the column named "port"
exten => 123,n,SayNumber(${John_port})

```

## See Also

RealTimeUpdate()

# RealTimeUpdate()

RealTimeUpdate() — Updates a value via the RealTime configuration handler

## Synopsis

```
RealTimeUpdate(family,colmatch,value,newcol,newval)
```

Uses the RealTime configuration handler system to update a value. The column *newcol* in *family* matching column *colmatch = value* will be updated to *newval*.

A channel variable named REALTIMECOUNT will be set with the number of rows updated or -1 if an error occurs.

```

; this will update the port column in the sipfriends table to a new
; value of 5061, where the name column matches "John"
exten => 123,1,RealTimeUpdate(sipfriends,name,John,port,5061)

```

## See Also

RealTime

# Record()

Record() — Records channel audio to a file

## Synopsis

```
Record(filename.format[,silence[,maxduration[,options]]])
```

Records audio from the channel in to the given *filename*. If the file already exists, it will be overwritten.

Optional arguments include:

<i>format</i>	Specifies the format of the file type to be recorded.
<i>silence</i>	Specifies the number of seconds of silence to allow before ending the recording and continuing on with the next priority in the dialplan.
<i>maxduration</i>	Specifies the maximum recording duration, in seconds. If not specified or 0, there is no maximum.
<i>options</i>	May contain any of the following letters:

- a Append to the recording, instead of overwriting it.
- n Do not answer, but record anyway if the line is not yet answered.
- q Quiet mode; do not play a beep tone at the beginning of the recording.
- s Skip recording if the line is not yet answered.
- t Use the alternate \* terminator key instead of the default #.
- x Ignore all termination keys and keep recording until hangup.

If the *filename* contains %d, these characters will be replaced with a number incremented by one each time the file is recorded.

The user can press # to terminate the recording and continue to the next priority in the dialplan.

```
; record the caller's name
exten => 123,1,Playback(pls-rcrd-name-at-tone)
exten => 123,n,Record(/tmp/name.gsm,3,30)
exten => 123,n,Playback(/tmp/name)
```

## RemoveQueueMember()

RemoveQueueMember() — Dynamically removes queue members

### Synopsis

```
RemoveQueueMember(queuename[,interface[,options]])
```

Dynamically removes the specified *interface* from the *queuename* call queue. If *interface* is not specified, this application removes the current channel from the queue.

If the *options* parameter is set to j, and the interface is not in the queue and there exists a priority n+101 (where n is the current priority), the application will jump to that priority.

```
; remove SIP/3000 from the techsupport queue
exten => 123,1,RemoveQueueMember(techsupport,SIP/3000)
```

### See Also

Queue(), AddQueueMember(), PauseQueueMember(), UnpauseQueueMember()

## ResetCDR()

ResetCDR() — Resets the Call Detail Record

### Synopsis

```
ResetCDR([options])
```

Causes the Call Detail Record to be reset for the current channel. The *options* parameter can be zero or more of the following options:

- a Store any stacked records.
- w Store the current CDR record before resetting it.
- v Save CDR variables.

```
; write a copy of the current CDR record, and then reset the CDR
exten => 123,1,Answer()
exten => 123,2,Playback(tt-monkeys)
exten => 123,3,ResetCDR(wv)
exten => 123,4,Playback(tt-monkeys)
```

## See Also

ForkCDR(), NoCDR()

## RetryDial()

RetryDial() — Attempts to place a call, and retries on failure

## Synopsis

```
RetryDial(announce,sleep,loops,technology/resource[&Technology2/resource2...]  
[,timeout][,options][,URL])
```

Attempts to place a call. If no channel can be reached, plays the file defined by *announce*, waiting *sleep* seconds to retry the call. If the specified number of attempts matches *loops*, the call will continue with the next priority in the dialplan. If *loops* is set to 0, the call will retry endlessly.

While waiting, a one-digit extension may be dialed. If that extension exists in either the context defined in `$_EXIT-CONTEXT` (if defined) or the current one, the call will transfer to that extension immediately.

All arguments after *loops* are passed directly to the `Dial()` application.

```
; attempt to dial the number three times via IAX, retrying every five
seconds
exten => 123,1,RetryDial(priv-trying,5,3,IAX2/VOIP/8885551212,30)
; if the caller presses 9 while waiting, dial the number on the Zap/4
channel
exten => 9,1,RetryDial(priv-trying,5,3,Zap/4/8885551212,30)
```

## See Also

Dial()

## Return()

Return() — Returns from a `Gosub()` or `GosubIf()`

## Synopsis

```
Return()
```

Returns from a previously invoked `Gosub()` or `GosubIf()`. If there was no previous invocation of `Gosub()` or `GosubIf()`, `Return()` exits abnormally.

## See Also

`Gosub()`, `StackPop()`

## Ringing()

`Ringing()` — Indicates ringing tone

## Synopsis

```
Ringing()
```

Requests that the channel indicate ringing tone to the user. It is up to the channel driver to specify exactly how ringing is indicated.

Note that this application does not actually provide audio ringing to the caller. Use the `Playtones()` application to do this.

```
; indicate that the phone is ringing, even though it isn't
exten => 123,1,Ringing()
exten => 123,2,Wait(5)
exten => 123,3,Playback(tt-somethingwrong)
```

## See Also

`Busy()`, `Congestion()`, `Progress()`, `Playtones()`

## SayAlpha()

`SayAlpha()` — Spells a string

## Synopsis

```
SayAlpha(string)
```

Spells out the specified *string*, using the current language setting for the channel. See the `CHANNEL` function for more information on changing the language for the current channel.

```
exten => 123,1,SayAlpha(ABC123XYZ)
```

## See Also

SayDigits(), SayNumber(), SayPhonetic(), CHANNEL

## SayDigits()

SayDigits() — Says the specified digits

## Synopsis

```
SayDigits(digits)
```

Says the specified *digits*, using the current language setting for the channel. See the CHANNEL function for more information on changing the language for the current channel.

```
exten => 123,1,SayDigits(1234)
```

## See Also

SayAlpha(), SayNumber(), SayPhonetic(), CHANNEL

## SayNumber()

SayNumber() — Says the specified number

## Synopsis

```
SayNumber(digits[,gender])
```

Says the specified number, using the current language setting for the channel. See the CHANNEL function for more information on changing the language for the current channel.

If the current language supports different genders, you can pass the *gender* argument to change the gender of the spoken number. You can use the following *gender* arguments:

- Use the *gender* arguments *f* for female, *m* for male, and *n* for neuter in European languages such as Portuguese, French, Spanish, and German.
- Use the *gender* argument *c* for commune and *n* for neuter in Nordic languages such as Danish, Swedish, and Norwegian.
- Use the *gender* argument *p* for plural enumerations in German.

```
; say the number in English
```

```
exten => 123,1,Set(CHANNEL(language)=en)
exten => 123,2,SayNumber(1234)
```



## Tip

For this application to work in languages other than English, you must have the appropriate sounds for the language you wish to use.

## See Also

SayAlpha(), SayDigits(), SayPhonetic(), CHANNEL

# SayPhonetic()

SayPhonetic() — Spells the specified string phonetically

## Synopsis

```
SayPhonetic(string)
```

Spells the specified *string* using the NATO phonetic alphabet.

```
exten => 123,1,SayPhonetic(asterisk)
```

## See Also

SayAlpha(), SayDigits(), SayNumber()

# SayUnixTime()

SayUnixTime() — Says the specified time in a custom format

## Synopsis

```
SayUnixTime([unixtime][,timezone][,format]))
```

Speaks the specified time according to the specified time zone and format. The arguments are:

*unixtime* The time, in seconds, since January 1, 1970. May be negative. Defaults to now.

*timezone* The time zone. See `/usr/share/zoneinfo/` for a list. Defaults to the machine default.

*format* The format in which the time is to be spoken. See `voicemail.conf` for a list of formats. Defaults to "ABdY 'digits/at' IMp".

```
exten => 123,1,SayUnixTime(,,IMp)
```

## See Also

STRFTIME, STRPTIME, IFTIME

## SendDTMF()

SendDTMF() — Sends arbitrary DTMF digits to the channel

## Synopsis

```
SendDTMF(digits[,timeout_ms])
```

Sends the specified DTMF digits on a channel. Valid DTMF digits include 0–9, \*, #, and A–D. You may also use the letter w as a digit, which indicates a 500 millisecond wait. The *timeout\_ms* argument is the amount of time between digits, in milliseconds. If not specified, *timeout\_ms* defaults to 250 milliseconds.

```
exten => 123,1,SendDTMF(3212333w222w366w3212333322321,250)
```

## See Also

Read()

## SendImage()

SendImage() — Sends an image file

## Synopsis

```
SendImage(filename,options)
```

Sends an image on a channel, if image transport is supported. This application sets a channel variable named SENDIMAGESTATUS to either OK or NOSUPPORT upon completion.

If the *options* parameter is set to j and the channel does not support image transport, and there exists a priority n+101 (where n is the current priority), execution will continue at that step.

```
exten => 123,1,SendImage(logo.jpg)
```

## See Also

SendText(), SendURL()

# SendText()

SendText() — Sends text to the channel

## Synopsis

```
SendText(text,options)
```

Sends *text* on a channel, if text transport is supported. Upon completion, a channel variable named SENDTEXTSTATUS will be set to one of the following values:

**SUCCESS**      The transmission of the text was successful.

**FAILURE**      The transmission of the text failed.

**NOSUPPORT**    The underlying channel does not support the transmission of text.

If the *options* parameter is set to *j* and the channel does not support text transport, and there exists a priority *n*+101 (where *n* is the current priority), execution will continue at that step.

```
exten => 123,1,SendText(Welcome to Asterisk)
```

## See Also

SendImage(), SendURL()

# SendURL()

SendURL() — Sends the specified URL to the channel (if supported)

## Synopsis

```
SendURL(URL[,options])
```

Requests that the client go to the specified URL. The application also sets a channel variable named SENDURLSTATUS to one of the following values upon completion:

**SUCCESS**      The transmission of the URL was successful.

**FAILURE**      The transmission of the URL failed.

**NOLOAD**      The is web-enabled but failed to load the URL.

**NOSUPPORT**    The underlying channel does not support the transmission of the URL.

If the *options* parameter contains the *wait* option, execution will wait for an acknowledgment that the URL has been loaded before continuing.

If the *options* parameter contains the *j* option and the client does not support HTML transport, and there exists a priority *n*+101 (where *n* is the number of the current priority), execution will continue at that step.



```
exten => 123,1,SendURL(www.asterisk.org,wait)
```

## See Also

SendImage(), SendText()

## Set()

Set() — Sets a variable to the specified value

## Synopsis

```
Set(n=value, [n2=value2...[,options]])
```

Sets the variable *n* to the specified *value*. Also sets the variable *n2* to the value of *value2*. If the variable name is prefixed with `_`, single inheritance is assumed. If the variable name is prefixed with `_ _`, infinite inheritance is assumed. Inheritance is used when you want channels created from the current channel to inherit the variable from the current channel.

If the *options* parameter is set to `g`, the variables will be set as global variables instead of channel variables.

```
; set a variable called DIALTIME, then use it
exten => 123,1,Set(DIALTIME=20)
exten => 123,1,Dial(Zap/4/5551212,,${DIALTIME})
```



### Note

The setting of multiple variables and the use of the `g` option have been deprecated. Please use multiple calls to `Set()` and the `GLOBAL()` dialplan function instead.

## See Also

GLOBAL, SET, ENV, channelvariables.txt

## SetAMAFlags()

SetAMAFlags() — Sets AMA flags in the Call Detail Record

## Synopsis

```
SetAMAFlags(flag)
```

Sets the AMA flags in the Call Detail Record for billing purposes, overriding any AMA settings in the channel configuration files. Valid choices are `default`, `omit`, `billing`, and `documentation`.

```
exten => 123,1,SetAMAFlags(billing)
```

## See Also

SetCDRUserField(), AppendCDRUserField()

## SetCallerID()

SetCallerID() — Sets the Caller ID for the channel

## Synopsis

```
SetCallerID(clid[, a])
```



### Note

This application has been deprecated in favor of:

```
Set(CALLERID(all)=Some Name <1234>)
```

Sets the Caller ID on the channel to a specified value. If the *a* argument is passed, ANI is also set to the specified value.

```
; override the Caller ID for this call
exten => 123,1,Set(CALLERID(all)="John Q. Public <8885551212>")
```

## See Also

The CALLERID

## SetCallerPres()

SetCallerPres() — Sets Caller ID presentation flags

## Synopsis

```
SetCallerPres(presentation)
```

Sets the Caller ID presentation flags on a Q931 PRI connection.

Valid presentations are:

<code>allowed_not_screened</code>	Presentation allowed, not screened
<code>allowed_passed_screen</code>	Presentation allowed, passed screen
<code>allowed_failed_screen</code>	Presentation allowed, failed screen
<code>allowed</code>	Presentation allowed, network number
<code>prohib_not_screened</code>	Presentation prohibited, not screened
<code>prohib_passed_screen</code>	Presentation prohibited, passed screen

<code>prohib_failed_screen</code>	Presentation prohibited, failed screen
<code>prohib</code>	Presentation prohibited, network number
<code>unavailable</code>	Number unavailable

```
exten => 123,1,SetCallerPres(allowed_not_screened)
exten => 123,2,Dial(Zap/g1/8885551212)
```

## See Also

`CALLERID()`

# SetCDRUserField()

`SetCDRUserField()` — Sets the Call Detail Record user field

## Synopsis

```
SetCDRUserField(value)
```

Sets the CDR user field to the specified *value*. The CDR user field is an extra field that you can use for data not stored anywhere else in the record. CDR records can be used for billing purposes or for storing other arbitrary data about a particular call.

```
exten => 123,1,SetCDRUserField(testing)
exten => 123,2,Playback(tt-monkeys)
```



### Note

This application has been deprecated in favor of the `CDR()` function.

```
exten => 123,1,Set(CDR(userfield)=54321)
```

## See Also

`AppendCDRUserField()`, `SetAMAFlags()`

# SetGlobalVar()

`SetGlobalVar()` — Sets a global variable to the specified value

## Synopsis

```
SetGlobalVar(n=value)
```



### Note

This application has been deprecated in favor of:

```
Set (GLOBAL(var)=...)
```

Sets a global variable called *n* to the specified *value*. Global variables are available across channels.

```
; set the NUMRINGS global variable to 3
exten => 123,1,SetGlobalVar(NUMRINGS=3)
```

## See Also

Set()

## SetMusicOnHold()

SetMusicOnHold() — Sets the default music-on-hold class for the current channel

## Synopsis

```
SetMusicOnHold(class)
```



### Note

This application has been deprecated in favor of:

```
Set (CHANNEL(musicclass)=...)
```

Sets the default *class* for music on hold for the current channel. When music on hold is activated, this class will be used to select which music is played. Classes are defined in the configuration file `musiconhold.conf`.

```
exten=s,1,Answer( )
exten=s,2,SetMusicOnHold(default)
exten=s,3,WaitMusicOnHold( )
```

## See Also

WaitMusicOnHold(), `musiconhold.conf`, MusicOnHold()

## SetTransferCapability()

SetTransferCapability() — Sets the ISDN transfer capability of a channel

## Synopsis

```
SetTransferCapability(transfercapability)
```

This application sets the ISDN transfer capability of the current channel to a new value. Valid values for *transfer-capability* are:

SPEECH	0x00, speech (default, voice calls)
DIGITAL	0x08, unrestricted digital information (data calls)

RESTRICTED_DIGITAL	0x09, restricted digital information
3K1AUDIO	0x10, 3.1kHz Audio (fax calls)
DIGITAL_W_TONES	0x11, unrestricted digital information with tones/announcements
VIDEO	0x18, video



### Note

This application is deprecated and the functionality has been replaced with `Set(CHANNEL(transfercapability)=transfercapability)` syntax.

```
exten => 123,1,Set(CHANNEL(transfercapability)=SPEECH)
```

## SIPAddHeader()

SIPAddHeader() — Adds a SIP header to the outbound call

### Synopsis

```
SIPAddHeader(Header: Content)
```

Adds a header to a SIP call placed with the `Dial()` application. A nonstandard SIP header should begin with X-, such as X-Asterisk-Accountcode:. Use this application with care—adding the wrong headers may cause any number of problems.

For more flexibility, see the `SIP_HEADER()` dialplan function.

```
exten => 123,1,SIPAddHeader(X-Asterisk-Testing: Just testing!)
exten => 123,2,Dial(SIP/123)
```

### See Also

`SIP_HEADER`

## SIPDtmfMode()

SIPDtmfMode() — Changes the DTMF method for a SIP call

### Synopsis

```
SIPDtmfMode(method)
```

Changes the DTMF method for a SIP call. The *method* can be either `inband`, `info`, or `rfc2833`.

```
exten => 123,1,SIPDtmfMode(rfc2833)
exten => 123,2,Dial(SIP/123)
```

# SLAStation()

SLAStation() — Shared line appearance station

## Synopsis

```
SLAStation(station)
```

This application should be executed by an SLA station. The format of the *station* parameter depends on how the call was initiated. If the phone was just taken off hook, then the *station* parameter should contain only the station name. If the call was initiated by pressing a line key, then the station name should be preceded by an underscore and the trunk name associated with that line button (*station1\_line2*, for example).

For more information on shared line appearances, see the `doc/sla.pdf` file in the Asterisk source.

```
exten => 123,1,SLAStation(station1)
exten => 124,1,SLAStation(station1_line2)
```

## See Also

SLATrunk(), `sla.conf`

# SLATrunk()

SLATrunk() — Shared line appearance trunk

## Synopsis

```
SLATrunk(trunk)
```

This application should be executed by an SLA trunk on an inbound call. The channel calling this application should correspond to the SLA trunk specified by the *trunk* parameter.

For more information on shared line appearances, see the `doc/sla.pdf` file in the Asterisk source.

```
exten => 123,1,SLATrunk(line2)
```

## See Also

SLAStation(), `sla.conf`

# SoftHangup()

SoftHangup() — Performs a soft hangup of the requested channel

## Synopsis

```
SoftHangup(technology/resource,options)
```

Hangs up the requested channel. The *options* argument may contain the letter a, which causes all channels on the specified device to be hung up (currently, the *options* argument may contain only a).

```
; hang up all calls using Zap/4 so we can use it
exten => 123,1,SoftHangup(Zap/4,a)
exten => 123,2,Wait(2)
exten => 123,3,Dial(Zap/4/5551212)
```

## See Also

Hangup()

## StackPop()

StackPop() — Removes last address from Gosub( ) stack

## Synopsis

```
StackPop( )
```

Removes the last address from the Gosub( ) stack. This is frequently used when handling error conditions within Gosub( ) routines when it is no longer appropriate to return control of the dialplan back to where the Gosub( ) routine was called from.

```
exten => s,1,Read(input,get-input)
exten => s,n,Gosub(validate,1)
exten => s,n,Dial(SIP/${input})
; Ensure that input is between 400 and 499
exten => validate,1,GotoIf($[ ${input} > 499 ]?error,1)
exten => validate,n,GotoIf($[ ${input} < 400 ]?error,1)
exten => validate,n,Return
exten => error,1,StackPop( )
exten => error,2,Goto(s,1)
```

## See Also

Return(), Gosub()

## StartMusicOnHold()

StartMusicOnHold() — Starts music on hold

## Synopsis

```
StartMusicOnHold([class])
```

Plays hold music specified by *class*, as configured in `musiconhold.conf`. If omitted, the default music class for the channel will be used. You can use the `CHANNEL(musicclass)` function to set the default music class for the channel.

Returns immediately.

```
; transfer telemarketers to this extension to keep them busy
exten => 123,1,Answer()
exten => 123,2,Playback(tt-allbusy)
exten => 123,3,StartMusicOnHold(default)
exten => 123,4,Wait(600)
exten => 123,5,StopMusicOnHold()
```

## See Also

`WaitMusicOnHold()`, `StopMusicOnHold()`

## StopMixMonitor()

`StopMixMonitor()` — Stops monitoring a channel

### Synopsis

```
StopMixMonitor()
```

Stops monitoring (recording) a channel. This application has no effect if the channel is not currently being monitored.

```
exten => 123,1,Answer()
exten => 123,2,MixMonitor(monitor_test.wav)
exten => 123,3,SayDigits(12345678901234567890)
exten => 123,4,StopMixMonitor()
```

## See Also

`MixMonitor()`

## StopMonitor()

`StopMonitor()` — Stops monitoring a channel

### Synopsis

```
StopMonitor()
```

Stops monitoring (recording) a channel. This application has no effect if the channel is not currently being monitored.

```
exten => 123,1,Answer()
exten => 123,2,Monitor(wav,monitor_test,mb)
```



```
exten => 123,3,SayDigits(12345678901234567890)
exten => 123,4,StopMonitor()
```

## See Also

ChangeMonitor()

# StopPlaytones()

StopPlaytones() — Stops playing a tone list

## Synopsis

```
StopPlaytones()
```

Stops playing the currently playing tone list.

```
exten => 123,1,Playtones(busy)
exten => 123,2,Wait(2)
exten => 123,3,StopPlaytones()
exten => 123,4,Playtones(congestion)
exten => 123,5,Wait(2)
exten => 123,6,StopPlaytones()
exten => 123,7,Goto(1)
```

## See Also

Playtones(), indications.conf

# StopMusicOnHold()

StopMusicOnHold() — Stops music on hold

## Synopsis

```
StopMusicOnHold()
```

Ends playing music on hold on a channel. If music on hold was not already playing, has no effect.

```
; transfer telemarketers to this extension to keep them busy
exten => 123,1,Answer()
exten => 123,2,Playback(tt-allbusy)
exten => 123,3,StartMusicOnHold(default)
exten => 123,4,Wait(600)
exten => 123,5,StopMusicOnHold()
```

## See Also

WaitMusicOnHold(), StartMusicOnHold()

# System()

System() — Executes an operating system command

## Synopsis

```
System(command)
```

Executes a *command* in the underlying operating system. This application sets a channel variable named SYSTEM-STATUS to either FAILURE or SUCCESS, depending on whether or not Asterisk was successfully able to run the command.

This application is very similar to the TrySystem( ) application, except that it will return -1 if it is unable to execute the system command, whereas the TrySystem( ) application will always return 0.

```
exten => 123,1,System(echo hello > /tmp/hello.txt)
```

## See Also

TrySystem()

# Transfer()

Transfer() — Transfers the caller to a remote extension

## Synopsis

```
Transfer([Technology]/]destination[,options])
```

Requests that the remote caller be transferred to the given (optional *Technology* and) *destination*. If the *Technology* is set to IAX2, SIP, Zap, etc., then transfer will happen only if the incoming call is of the same channel type.

Upon completion, this application sets a channel variable named TRANSFERSTATUS to one of the following values:

SUCCESS        The transfer was successful.

FAILURE        The transfer was not successful.

UNSUPPORTED    The transfer was not supported by the underlying channel driver.

If the *options* parameter is set to the letter j and the transfer is *not* supported or successful, and there exists a priority n+101 (where n is the current priority), that priority will be taken next.

```
; transfer calls from extension 123 to extension SIP/123@otherserver
exten => 123,1,Transfer(SIP/123@otherserver)
```

# TryExec()

TryExec() — Tries to execute an Asterisk application

## Synopsis

```
TryExec(app(args))
```

Attempts to run the specified Asterisk application.

This application is very similar to the `Exec()` application, except that it always returns normally, whereas the `Exec()` application will act as if the underlying application was natively called, including exit status. This application can be used to catch a condition that would normally cause the underlying application to exit abnormally.

```
exten => 123,1,TryExec(VMAuthenticate(@default))
```

## See Also

`Exec()`

## TrySystem()

`TrySystem()` — Tries to execute an operating system command

## Synopsis

```
TrySystem(command)
```

Attempts to execute a *command* in the underlying operating system. The result of the *command* will be placed in the `SYSTEMSTATUS` channel variable and will be one of the following:

**FAILURE**    The specified *command* could not be executed.

**SUCCESS**   The specified *command* executed successfully.

**APPERROR**   The specified *command* executed, but returned an error code.

This application is very similar to the `System()` application, except that it always returns normally, whereas the `System()` application will return abnormally if it is unable to execute the system command.

```
exten => 123,1,TrySystem(echo hello > /tmp/hello.txt)
```

## See Also

`System()`

## UnpauseMonitor()

`UnpauseMonitor()` — Resumes monitoring of a channel

## Synopsis

```
UnpauseMonitor()
```

Resumes monitoring a channel, after being suspended by `PauseMonitor()`.

```
exten => 123,1,Answer()
exten => 123,n,Monitor(wav,monitor_test)
exten => 123,n,Playback(demo-congrats)
    ; temporarily pause the monitoring while we gather some secret info
exten => 123,n,PauseMonitor()
exten => 123,n,Read(NEWPASS,vm-newpassword)
exten => 123,n,SayDigits(${NEWPASS})
    ; unpause the recording and continue recording the call
exten => 123,n,UnpauseMonitor()
exten => 123,n,Dial(${JOHN})
```

## See Also

`Monitor()`, `StopMonitor()`, `Page()`

## UnpauseQueueMember()

`UnpauseQueueMember()` — Unpauses a queue member

## Synopsis

```
UnpauseQueueMember([queue_name],[interface],[options])
```

Unpauses (resumes calls to) a queue member. This is the counterpart to `PauseQueueMember()`, and it operates exactly the same way, except it unpauses instead of pausing the given interface.

Upon completion, this application sets a channel variable named `UPQMSTATUS` to either `UNPAUSED` or `NOTFOUND`.

If the *options* parameter is set to the letter `j` and the queue member can not be found, and there exists a priority `n+101` (where `n` is the current priority), control of the call will continue at that priority

```
exten => 123,1,PauseQueueMember(myqueue,SIP/300)
exten => 124,1,UnpauseQueueMember(myqueue,SIP/300)
```

## See Also

`PauseQueueMember()`

## UserEvent()

`UserEvent()` — Sends an arbitrary event to the Manager Interface

## Synopsis

```
UserEvent(eventname[,body])
```

Sends an arbitrary event to the Manager Interface, with an optional body representing additional arguments. The format of the event is:

```
Event: UserEvent
UserEvent: eventname
           body
```

If the body is not specified, only the Event and UserEvent fields will be present in the Manager event.

```
exten => 123,1,UserEvent(BossCalled,${CALLERID(name)} has called the boss!)
exten => 123,2,Dial(${BOSS})
```

## See Also

manager.conf, Asterisk Manager Interface

## Verbose()

Verbose() — Sends arbitrary text to verbose output

## Synopsis

```
Verbose([level],message)
```

Sends the specified *message* to verbose output. The *level* must be an integer value. If not specified, *level* defaults to 0.

```
exten => 123,1,Verbose(Somebody called extension 123)
exten => 123,2,Playback(extension)
exten => 123,3,SayDigits(${EXTEN})
```



### Warning

The optional argument *level* is not so optional, if you include the delimiter `|` in your invocation of `Verbose()`. If the delimiter is found, `Verbose()` assumes that you meant to specify *level* (and chops off everything preceding the initial `|`). It is therefore probably best to get in the habit of always specifying the *level*.

## See also

NoOp(), Log()

## VMAuthenticate()

VMAuthenticate() — Authenticates the caller from voicemail passwords

## Synopsis

```
VMAuthenticate([mailbox][@context[,options]])
```

Behaves identically to the `Authenticate()` application, with the exception that the passwords are taken from `voicemail.conf`.

If *mailbox* is specified, only that mailbox's password will be considered valid. If *mailbox* is not specified, the channel variable `AUTH_MAILBOX` will be set with the authenticated mailbox.

If the *options* parameter is set to the letter *s*, Asterisk will skip the initial prompts.

```
; authenticate off of any mailbox password in the default voicemail context
; and tell us the matching mailbox number
exten => 123,1,VMAuthenticate(@default)
exten => 123,2,SayDigits(${AUTH_MAILBOX})
```

## See Also

`Authenticate()`, `voicemail.conf`

## VoiceMail()

`VoiceMail()` — Leaves a voicemail message in the specified mailbox

## Synopsis

```
VoiceMail(mailbox[@context][&mailbox[@context]][...] | options)
```

Leaves voicemail for a given *mailbox* (must be configured in `voicemail.conf`). If more than one mailbox is specified, the greetings will be taken from the first mailbox specified.

Options:

- s*        Skip the playback of instructions.
- u*        Play the user's unavailable greeting.
- b*        Play the user's busy greeting.
- g(num)* Amplify the recording by *num* decibels (dB).
- j*        If the requested mailbox does not exist, and there exists a priority *n*+101 (where *n* is the current priority), that priority will be taken next.

If the caller presses 0 (zero) during the prompt, the call jumps to the *o* (lowercase letter *o*) extension in the current context, if `operator=yes` was specified in `voicemail.conf`.

If the caller presses *\** during the prompt, the call jumps to extension *a* in the current context. This is often used to send the caller to a personal assistant.

Upon completion, this application sets a channel variable named `VMSTATUS`. It will contain one of the following values:

**SUCCESS**    The call was successfully sent to voicemail.

**USEREXIT** The caller exited from the voicemail system.

**FAILED** The system was not able to send the call to voicemail.

```
; send caller to unavailable voicemail for mailbox 123
exten => 123,1,VoiceMail(123@default,u)
```

## See Also

VoiceMailMain(), voicemail.conf

# VoiceMailMain()

VoiceMailMain() — Enters the voicemail system

## Synopsis

```
VoiceMailMain([mailbox][@context][,options])
```

Enters the main voicemail system for the checking of voicemail. Passing the *mailbox* argument will stop the voicemail system from prompting the user for the mailbox number. You should also specify a voicemail *context*.

The *options* string can contain zero or more of the following options:

**s** Skip the password check.

**p** This option tells Asterisk to interpret the *mailbox* as a number that should be prepended to the mailbox entered by the caller. This is most often used when many voicemail boxes for different companies are hosted on the same Asterisk server.

**g(gain)** When recording voicemail messages, increase the volume by *gain*. This option should be specified in an integer number of decibels.

**a(folder)** Skip the folder prompt and go directly to the specified *folder*. This option defaults to the INBOX.

```
; go to voicemail menu for mailbox 123 in the default voicemail context
exten => 123,1,VoiceMailMain(123@default)
```

## See Also

VoiceMail(), voicemail.conf

# Wait()

Wait() — Waits for a specified number of seconds

## Synopsis

```
Wait(seconds)
```

Waits for the specified number of *seconds*. You can pass fractions of a second. For example, setting *seconds* to 1.5 would make the dialplan wait 1.5 seconds before going on to the next priority in the dialplan.

```
; wait 1.5 seconds before playing the prompt
exten => s,1,Answer()
exten => s,2,Wait(1.5)
exten => s,3,Background(enter-ext-of-person)
```

## WaitExten()

WaitExten() — Waits for an extension to be entered

### Synopsis

```
WaitExten([seconds][,options])
```

Waits for the user to enter a new extension for the specified number of seconds. You can pass fractions of a second (e.g., 1.5 = 1.5 seconds). If *seconds* is unspecified, the default extension timeout will be used. Most often, this application is used without specifying the *seconds* options.

The *options* parameter may be set to the following option:

m[ (*class*) ]    Provide music on hold to the caller while waiting for an extension. Optionally, you can specify the music on hold *class* within parentheses.

```
; wait 15 seconds for the user to dial an extension
exten => s,1,Answer()
exten => s,2,Playback(enter-ext-of-person)
exten => s,3,WaitExten(15)
```

### See Also

Background(), TIMEOUT

## WaitForRing()

WaitForRing() — Waits the specified number of seconds for a ring

### Synopsis

```
WaitForRing(timeout)
```

Waits at least *timeout* seconds after the next ring has completed.

```
; wait five seconds for a ring, and then send some DTMF digits
exten => 123,1,Answer()
exten => 123,2,WaitForRing(5)
exten => 123,3,SendDTMF(1234)
```



## See Also

WaitForSilence()

## WaitForSilence()

WaitForSilence() — Waits for a specified amount of silence

### Synopsis

```
WaitForSilence(silencerequired[,repeat[,timeout]])
```

Waits for *repeat* instances of *silencerequired* milliseconds of silence. If *repeat* is omitted, the application waits for a single instance of *silencerequired* milliseconds of silence.

If the *timeout* option is specified, this application will return to the next priority in the dialplan after the specified number of seconds, even if silence has not been detected.



### Warning

Please use the *timeout* with caution, as it may defeat the purpose of this application, which is to wait indefinitely until silence is detected on the line. You may want to set the timeout to a high value only to avoid an infinite loop in cases where silence is never detected.

This applications sets a channel variable named WAITSTATUS to either SILENCE or TIMEOUT.

```
; wait for three instances of 300 ms of silence
exten => 123,WaitForSilence(300,3)
```

## See Also

WaitForRing()

## WaitMusicOnHold()

WaitMusicOnHold() — Waits the specified number of seconds, playing music on hold

### Synopsis

```
WaitMusicOnHold(delay)
```

Plays hold music for the specified number of seconds. If no hold music is available, the delay will still occur but with no sound.

Returns 0 when done, or -1 on hangup.

```
; allow caller to hear Music on Hold for five minutes
exten => 123,1,Answer()
```

```
exten => 123,2,WaitMusicOnHold(300)
exten => 123,3,Hangup()
```

## See Also

SetMusicOnHold(), musiconhold.conf

## While()

While() — Starts a while loop

## Synopsis

```
While(expr)
```

Starts a while loop. Execution will return to this point when EndWhile() is called, until *expr* is no longer true. If a condition is met causing the loop to exit, Asterisk continues execution of the dialplan on the next priority after the corresponding EndWhile().

```
exten => 123,1,Set(COUNT=1)
exten => 123,2,While($[ ${COUNT} < 5 ])
exten => 123,3,SayNumber(${COUNT})
exten => 123,4,Set(COUNT=${COUNT} + 1)
exten => 123,5,EndWhile()
```

## See Also

EndWhile(), ExitWhile(), GotIf()

## Zapateller()

Zapateller() — Uses a special information tone to block telemarketers

## Synopsis

```
Zapateller(options)
```

Generates a special information tone to block telemarketers and other computer-dialed calls from bothering you.

The *options* argument is a pipe-delimited list of options. The following options are available:

- answer** Causes the line to be answered before playing the tone.
- nocallerid** Causes Zapateller to play the tone only if no Caller ID information is available.

```
; answer the line, and play the SIT tone if there is no Caller
ID information
exten => 123,1,Zapateller(answer|nocallerid)
```

## See Also

PrivacyManager()

## ZapBarge()

ZapBarge() — Barges in on (monitors) a Zap channel

## Synopsis

```
ZapBarge([channel])
```

Barges in on a specified Zap *channel*, or prompts if one is not specified. The people on the channel won't be able to hear you and will have no indication that their call is being monitored.

If *channel* is not specified, you will be prompted for the channel number. Enter **4#** for Zap/4, for example.

```
exten => 123,1,ZapBarge(Zap/2)
exten => 123,2,Hangup()
```

## See Also

ZapScan()

## ZapRAS()

ZapRAS() — Executes the Zaptel ISDN Remote Access Server

## Synopsis

```
ZapRAS(args)
```

Executes an ISDN RAS server using *pppd* on the current channel. The channel must be a clear channel (i.e., PRI source) and a Zaptel channel to be able to use this function (no modem emulation is included).

Your *pppd* must be patched to be Zaptel-aware. *args* is a pipe-delimited list of arguments.

This application is for use only on ISDN lines, and your kernel must be patched to support ZapRAS(). You must also have *ppp* support in your kernel.

```
exten => 123,1,Answer()
exten => 123,1,ZapRas(debug|64000|noauth|netmask|255.255.255.0|
10.0.0.1:10.0.0.2)
```

## ZapScan()

ZapScan() — Scans Zap channels to monitor calls

## Synopsis

```
ZapScan([group])
```

Allows a call center manager to monitor Zap channels in a convenient way. Use # to select the next channel, and use \* to exit. You may limit scanning to a particular channel group (as set with the GROUP( ) function) by setting the *group* argument.

```
exten => 123,1,ZapScan()
```

## See Also

ZapBarge()

# Appendix B. Asterisk Dialplan Functions

Dialplan functions are very powerful, and once you begin using them, you will wonder how you got along without them. Functions are used in the dialplan in a similar manner to variables. If it helps, you can think of them as intelligent variables (or for those of you from the database world, variables with triggers). When you invoke them, they perform a specific action, and their result becomes a part of the command in which you have included the function (in exactly the same way as a variable would).

## AGENT

AGENT — Returns information about an agent

### Synopsis

```
AGENT(agentid[ :item])
```

This function allows you to retrieve information pertaining to agents and may only be read, not set.

The valid items to retrieve are:

<i>status</i> ( <i>default</i> )	The status of the agent (LOGGEDIN   LOGGEDOUT)
<i>password</i>	The password of the agent
<i>name</i>	The name of the agent
<i>mohclass</i>	Music-on-hold class
<i>exten</i>	The callback extension for the Agent (AgentCallbackLogin)
<i>channel</i>	The name of the active channel for the Agent (AgentLogin)

## ARRAY

ARRAY — Allows one to define several variables at one time

### Synopsis

```
ARRAY(var1[ |var2[...][ |varN]])
```

The comma-separated list, which the function equals, will be interpreted as a set of values to which the comma-separated list of variable names in the argument should be set. This function may only be set, not read.

```
; Set var1 to 1 and var2 to 2.
exten => 123,1,Set(ARRAY(var1,var2)=1\,2)
```

**Tip**

Remember to either backslash your commas in *extensions.conf* or quote the entire argument, since `Set ( )` can take multiple arguments itself.

## See Also

`Set()`

## BASE64\_DECODE

`BASE64_DECODE` — Decodes a BASE64 encoded string

### Synopsis

```
BASE64_DECODE(base64_string)
```

Decodes a BASE64 string. This function may only be read, not set.

## See Also

`BASE64_ENCODE ( )`

## BASE64\_ENCODE

`BASE64_ENCODE` — Encodes a string in BASE64

### Synopsis

```
BASE64_ENCODE(string)
```

Encodes a string in BASE64. This function may only be read, not set.

## See Also

`BASE64_DECODE ( )`

## BLACKLIST

`BLACKLIST` — Checks if the Caller ID is on the blacklist

### Synopsis

When read, `BLACKLIST ( )` uses the AstDB to check if the Caller ID is in family `blacklist`. Returns 1 or 0.

This function may only be read, not set.

## See Also

DB ( )

## CALLERID

CALLERID — Gets or sets Caller ID data on the channel

## Synopsis

```
CALLERID(datatype[, optional-CID])
```

CALLERID ( ) parses the Caller ID string within the current channel and returns all or part, as specified by *datatype*. The allowable datatypes are *all*, *name*, *num*, *ani*, *dnid*, or *rdnis*. Optionally, an alternative Caller ID may be specified if you wish to parse that string instead of the Caller ID set on the channel.

This function may be both read and set.

## CDR

CDR — Gets or sets CDR information for this call (which will be written to the CDR log)

## Synopsis

```
CDR(fieldname[, options])
```

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

<code>clid</code>	Read-only. Use the CALLERID( <i>all</i> ) function to set this value.
<code>lastapp</code>	Read-only. Denotes the last application run.
<code>lastdata</code>	Read-only. Denotes the arguments to the last application run.
<code>src</code>	Read-only. Use the CALLERID( <i>ani</i> ) function to set this value.
<code>dst</code>	Read-only. Corresponds to the final extension in the dialplan.
<code>dcontext</code>	Read-only. Corresponds to the final context in the dialplan.
<code>channel</code>	Read-only. The name of the channel on which the call originated.
<code>dstchannel</code>	Read-only. The name of the channel on which the call terminated.
<code>disposition</code>	Read-only. Maximum reached state of the channel. If the <i>u</i> option is specified, this value will be returned as an integer, instead of a string: 1 = NO ANSWER, 2 = BUSY, 3 = FAILED, 4 = ANSWERED.
<code>amaflags</code>	Read/write. Billing flags. If the <i>u</i> option is specified, this value will be returned as an integer, instead of a string: 1 = OMIT, 2 = BILLING, 3 = DOCUMENTATION.

<code>accountcode</code>	Read/write. Billing account (19 char maximum).
<code>userfield</code>	Read/write. User-defined field.
<code>start</code>	Read-only. Time when the call started. If the <code>u</code> option is specified, this value will be returned as an integer (seconds since the epoch) instead of a formatted date/time string.
<code>answer</code>	Read-only. Time when the call was answered (may be blank if the call is not yet answered). If the <code>u</code> option is specified, this value will be returned as an integer (seconds since the epoch) instead of a formatted date/time string.
<code>end</code>	Read-only. Time when the call was completed (may be blank if the call is not yet complete). If the <code>u</code> option is specified, this value will be returned as an integer (seconds since the epoch) instead of a formatted date/time string.
<code>duration</code>	Read-only. The difference between start and end, in seconds. May be 0, if the call is not yet complete.
<code>billsec</code>	Read-only. The difference between answer and end, in seconds. May be 0, if the call is not yet complete.
<code>uniqueid</code>	Read-only. A string that will be unique per-call within this Asterisk instance.

The following options may be specified:

<code>l</code>	All results will be retrieved from the last Call Detail Record for the call, in the case of using multiple CDRs via <code>ForkCDR()</code> .
<code>r</code>	Custom CDR variables will be retrieved from the last Call Detail Record, but the standard fields will be retrieved from the first.
<code>u</code>	The unparsed value will be returned. See the fieldname list above for entries that are affected by this flag.

You may also supply a *fieldname* not on the above list, and create your own variable, whose value can be changed with this function, and this variable will be stored in the CDR.

## See Also

`CHANNEL()`

# CHANNEL

**CHANNEL** — Gets or sets various channel parameters

## Synopsis

`CHANNEL(item)`

Standard items (provided by all channel technologies) are:

<code>audioreadformat</code>	Read-only. Format currently being read.
------------------------------	---



<code>audionativeformat</code>	Read-only. Format used natively for audio.												
<code>videonativeformat</code>	Read-only. Format used natively for video.												
<code>audiowriteformat</code>	Read-only. Format currently being written.												
<code>callgroup</code>	Read/write. Callgroups for call pickup.												
<code>channeltype</code>	Read-only. Technology used for channel.												
<code>language</code>	Read/write. Language used for sounds played and recorded.												
<code>musicclass</code>	Read/write. Class used (from <i>musiconhold.conf</i> ) for hold music.												
<code>rxgain</code>	Read/write. Receive gain (in dB) on channel drivers that support it.												
<code>txgain</code>	Read/write. Transmit gain (in dB) on channel drivers that support it.												
<code>tonezone</code>	Read/write. Regional zone for indications played.												
<code>state</code>	Read-only. Current channel state.												
<code>transfercapability</code>	Read/write. What can be transferred on an ISDN circuit. Current valid values are:												
	<table> <tr> <td><code>SPEECH</code></td><td>Speech (default, voice calls)</td></tr> <tr> <td><code>DIGITAL</code></td><td>Unrestricted digital information (data calls)</td></tr> <tr> <td><code>RESTRICTED_DIGITAL</code></td><td>Restricted digital information</td></tr> <tr> <td><code>3K1AUDIO</code></td><td>3.1 kHz Audio (fax calls)</td></tr> <tr> <td><code>DIGITAL_W_TONES</code></td><td>Unrestricted digital information with tones/announcements</td></tr> <tr> <td><code>VIDEO</code></td><td>Video</td></tr> </table>	<code>SPEECH</code>	Speech (default, voice calls)	<code>DIGITAL</code>	Unrestricted digital information (data calls)	<code>RESTRICTED_DIGITAL</code>	Restricted digital information	<code>3K1AUDIO</code>	3.1 kHz Audio (fax calls)	<code>DIGITAL_W_TONES</code>	Unrestricted digital information with tones/announcements	<code>VIDEO</code>	Video
<code>SPEECH</code>	Speech (default, voice calls)												
<code>DIGITAL</code>	Unrestricted digital information (data calls)												
<code>RESTRICTED_DIGITAL</code>	Restricted digital information												
<code>3K1AUDIO</code>	3.1 kHz Audio (fax calls)												
<code>DIGITAL_W_TONES</code>	Unrestricted digital information with tones/announcements												
<code>VIDEO</code>	Video												

Additional items may be available from the channel driver providing the channel; see its documentation for details. Any item requested that is not available on the current channel will return an empty string.

## See Also

`CDR ( )`

## CHECK\_MD5

`CHECK_MD5` — Validate an MD5 digest

## Synopsis

```
CHECK_MD5(digest,data)
```

Returns 1 on success, 0 on failure.

This function is *deprecated* in favor of using the `MD5 ( )` function with the built-in expression parser.

## See Also

MD5 ( )

# CHECKSIPDOMAIN

CHECKSIPDOMAIN — Checks if a domain is local

## Synopsis

```
CHECKSIPDOMAIN(domain | IP)
```

This function checks if the domain in the argument is configured as a local SIP domain that this Asterisk server is configured to handle. Returns the domain name if it is locally handled, otherwise an empty string. Check the domain configuration option in `sip.conf`.

# CURL

CURL — Returns the data resulting from a GET or POST to a URI

## Synopsis

```
CURL(url [ post-data ])
```

By default, `CURL( )` will perform an HTTP GET to retrieve the *url*. However, if *post-data* is specified, an HTTP POST will be performed instead.

## See Also

SendURL()

# CUT

CUT — Cuts a string based on a given delimiter

## Synopsis

```
CUT(varname, char-delim, range-s)
```

`CUT( )` works in a similar manner to the `cut(1)` Unix command-line tool, and is, in fact, designed based upon that tool.

In the dialplan, you may specify character offsets to select a substring of a variable based purely on the uniform length of characters (namely 1). `CUT( )` is designed to help you work with data that may have multiple, variable-length sections, divided by a common delimiter.

The most common case is the name of a channel, which is composed of two parts, a base name and a unique identifier (e.g., *SIP/tom-abcd1234* or *SIP/bert-1a2b3c4d*). `CUT( )` may be used to trim the unique identifier, no matter how long the base name may be:

```
; Trim the unique identifier from the current channel name
exten => 123,1,Set(chan=${CUT(CHANNEL,-,1)})
```

*varname* is the name of the variable that will be operated on. Note that `CUT( )` operates on the *name* of a variable, rather than upon the *value* of a variable. `CUT( )` is somewhat unique in this regard.

*char-delim* is the character that will serve as the delimiter ('-' is the default)

*range-spec* allows you to define which fields are returned. The *range-spec* can be specified as a range with - (e.g., 1-3) or a group of ranges and field numbers by separating each with & (e.g., 1&3-4). Note that if multiple field numbers are specified, the resulting value will have its fields separated by the same delimiter.



### Tip

*range-spec* uses a 1-based offset. That is, the first field is field 1 (as opposed to a 0-based offset, where the first field would be 0).

## See Also

`FIELDQTY( )`

## DB

DB — Read or write to AstDB

## Synopsis

```
DB(family/key)
```

Will return the value of the entry in the database (or blank if it does not exist), or set the value in the database.

## See Also

`DBdel()`, `DB_DELETE( )`, `DBdeltree()`, `DB_EXISTS( )`

## DB\_DELETE

DB\_DELETE — Deletes a key or key family from the AstDB database

## Synopsis

```
DB_DELETE(family/key)
```

Returns a value from the database and delete it.

## See Also

DBdel(), DB( ), DBdeltree()

## DB\_EXISTS

DB\_EXISTS — Checks AstDB for specified key

### Synopsis

```
DB_EXISTS(family/key)
```

Check to see if a key exists in the Asterisk database.

## See Also

DB( )

## DUNDILOOKUP

DUNDILOOKUP — Queries DUNDi peers for a particular number

### Synopsis

```
DUNDILOOKUP(number[ | context[ | option]
```

Does a DUNDi lookup of a phone number.

## ENUMLOOKUP

ENUMLOOKUP — Queries the ENUM database for a particular number

### Synopsis

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

Allows for general or specific querying of NAPTR records or counts of NAPTR types for ENUM or ENUM-like DNS pointers.

## ENV

ENV — References environment variables

### Synopsis

---

`ENV(envname)`

Gets or sets the environment variable specified by *envname*.

## EVAL

EVAL — Evaluates stored variables

## Synopsis

---

`EVAL(variable)`

`EVAL()` is one of the most powerful dialplan functions. It permits one to store variable expressions in a location other than `extensions.conf`, such as a database, yet evaluate them in the dialplan, as if they were included there all along. You can bet that `EVAL()` is a cornerstone in making a dialplan truly dynamic.

```
; We might store something like "SIP/${DB(ext2chan/123)}" in the
; database entry for extension/123, which tells us to look up yet
; another database entry.
exten => _XXX,1,Set(dialline=${DB(extension/${EXTEN})})
exten => _XXX,n,Dial(${EVAL(${dialline})})

; Real world example (taken from production code)
exten => _1NXXNXXXXXX,n(generic),Set(provider=${DB(rt2provider/${route})}-nanp)
exten => _1NXXNXXXXXX,n(provider),Dial(${EVAL(${DB(provider/${provider})})})
exten => _1NXXNXXXXXX,n,Goto(nextroute)
```

## See Also

`Exec()`

## EXISTS

EXISTS — Checks if value is non-blank

## Synopsis

---

`EXISTS(data)`

Existence test: returns 1 if non-blank, 0 otherwise

## FIELDQTY

FIELDQTY — Counts fields

## Synopsis

---

`FIELDQTY(varname | delim)`

Counts the fields, with an arbitrary delimiter

## See Also

CUT( )

## FILTER

FILTER — Strips string of illegal characters

## Synopsis

```
FILTER(allowed-chars | string)
```

Filters the *string* to include only the characters shown in *allowed-chars*:

```
; Ensure that the Caller*ID number contains only digits
exten => Set(CALLERID(num))=${FILTER(0123456789,${CALLERID(num)})})
```

This function may only be read, not set.

## See Also

QUOTE( )

## GLOBAL

GLOBAL — References global namespace

## Synopsis

```
GLOBAL(varname)
```

Gets or sets the global variable specified.

## GROUP

GROUP — Associates the channel into a set group

## Synopsis

```
GROUP([category])
```

Gets or sets the channel group.

```
; Permit only one user to access the paging system at once.
exten => 8000,1,Set(GROUP( )=pager)
```

```
exten => 8000,n,GotoIf($[${GROUP_COUNT(pager)} > 1]?hangup)
exten => 8000,n,Page(SIP/101&SIP/102&SIP/103&SIP/104)
exten => 8000,n(hangup),Hangup
```

## See Also

`GROUP_COUNT()`, `GROUP_LIST()`, `GROUP_MATCH_COUNT()`

# GROUP\_COUNT

`GROUP_COUNT` — Counts the number of channels in the specified group.

## Synopsis

```
GROUP_COUNT([groupname][@category])
```

Counts the number of channels in the specified group. Will return the count of the current channel if the groupname is not specified.

## See Also

`GROUP()`, `GROUP_LIST()`, `GROUP_MATCH_COUNT()`

# GROUP\_LIST

`GROUP_LIST` — Lists channel groups

## Synopsis

```
GROUP_LIST()([groupname][@category])
```

Gets a list of the groups set on a channel.

## See Also

`GROUP()`, `GROUP_COUNT()`, `GROUP_MATCH_COUNT()`

# GROUP\_MATCH\_COUNT

`GROUP_MATCH_COUNT` — Counts channels in a matching group name

## Synopsis

```
GROUP_MATCH_COUNT(groupmatch[@category])
```

Counts the number of channels in the groups matching the specified pattern.

## See Also

`GROUP()`, `GROUP_COUNT()`, `GROUP_LIST()`

# IAXPEER

IAXPEER — Obtains IAX channel information

## Synopsis

```
IAXPEER(peername[ | item])
IAXPEER(CURRENTCHANNEL[ | item])
```

Gets IAX peer information.

If *peername* is specified, valid items are:

<code>ip</code>	The IP address of this peer. If <i>item</i> is not specified, the IP address will be given.
<code>status</code>	The peer's status (if <code>qualify=yes</code> ).
<code>mailbox</code>	The peer's configured mailbox.
<code>context</code>	The peer's configured context.
<code>expire</code>	The epoch time of the next registration expiration for this peer.
<code>dynamic</code>	Does this peer register with Asterisk? (yes/no)
<code>callerid_name</code>	The Caller ID name configured on this peer.
<code>callerid_num</code>	The Caller ID number configured on this peer.
<code>codecs</code>	The peer's configured codecs.
<code>codec[x]</code>	Preferred codec index number <i>x</i> (beginning with zero).

## See Also

`SIPPEER()`

# IF

IF — Conditional value selection

## Synopsis

```
IF(expr?[true][:false])
```

Conditional: returns the data following `?` if true, otherwise the data following `:`.

```
; Returns foo
```



```
exten => 123,1,Set(something=${IF($[2 > 1]?foo:bar)})  
; Returns bar  
exten => 123,n,Set(something=${IF($[2 < 1]?foo:bar)})
```

## See Also

GotoIf()

## IFTIME

IFTIME — Compares the current system time to a time specification

## Synopsis

```
IFTIME(times,days_of_week,days_of_month,months?[true][:false])
```

Conditional: Returns the data following ? if true, otherwise the data following :

<i>times</i>	Time ranges, in 24-hour format
<i>days_of_week</i>	Days of the week (mon, tue, wed, thu, fri, sat, sun)
<i>days_of_month</i>	Days of the month (1–31)
<i>months</i>	Months (jan, feb, mar, apr, etc.)

## See Also

GotoIfTime()

## ISNULL

ISNULL — Checks if a value is blank

## Synopsis

```
ISNULL(data)
```

Returns 1 if *data* is blank or 0 otherwise.

## See Also

LEN(), EXISTS()

## KEYPADHASH

KEYPADHASH — Converts letters into numbers

## Synopsis

```
KEYPADHASH(string)
```

Hashes the letters in *string* into the equivalent keypad numbers.

```
; Calculate the hashes of the authors' last names.  So, the  
; corresponding values would be 623736, 76484, and 82663443536.  
exten => 123,1,Set(lastname1=${KEYPADHASH(Madsen)})  
exten => 123,n,Set(lastname2=${KEYPADHASH(Smith)})  
exten => 123,n,Set(lastname3=${KEYPADHASH(VanMeggelen)})
```

## See Also

Directory()

## LANGUAGE

LANGUAGE — Accesses the channel language

## Synopsis

```
LANGUAGE ( )
```

Gets or sets the channel's language.

This function is deprecated in favor of CHANNEL(*language*).

## See Also

CHANNEL ( )

## LEN

LEN — Calculates the string length

## Synopsis

```
LEN(string)
```

Returns the length of *string*.

## MATH

MATH — Mathematical calculations

## Synopsis

```
MATH(number1 op number2 [, type_of_result])
```

Performs mathematical functions.

```
exten => 123,1,Set(value1=${MATH(1+2)})
```

## MD5

MD5 — Calculates MD5 digest

## Synopsis

```
MD5(data)
```

Computes the MD5 digest of *data*.

## See Also

SHA1 ( )

## MUSICCLASS

MUSICCLASS — Access a channel's music-on-hold setting

## Synopsis

```
MUSICCLASS ( )
```



### Note

This function has been deprecated in favor of `CHANNEL(musicclass)`.

Reads or sets the music-on-hold class.

## See Also

CHANNEL ( )

## QUEUE\_MEMBER\_COUNT

QUEUE\_MEMBER\_COUNT — Counts queue members

## Synopsis

```
QUEUE_MEMBER_COUNT(queuename)
```

Counts the number of members answering a queue.

## See Also

```
QUEUE_MEMBER_LIST()
```

## QUEUE\_MEMBER\_LIST

QUEUE\_MEMBER\_LIST — Lists queue members

## Synopsis

```
QUEUE_MEMBER_LIST(queuename)
```

Returns a list of interfaces in a queue.

## See Also

```
QUEUE_MEMBER_COUNT()
```

## QUEUE\_WAITING\_COUNT

QUEUE\_WAITING\_COUNT — Count waiting calls

## Synopsis

```
QUEUE_WAITING_COUNT(queuename)
```

Counts number of calls currently waiting in a queue.

## QUEUEAGENTCOUNT

QUEUEAGENTCOUNT

## Synopsis

```
QUEUEAGENTCOUNT(queuename)
```



### Note

This function has been deprecated in favor of `QUEUE_MEMBER_COUNT()`.

Counts number of agents answering a queue.

## See Also

`QUEUE_MEMBER_COUNT()`, `QUEUE_MEMBER_LIST()`

## QUOTE

QUOTE — Escapes a string

## Synopsis

```
QUOTE(string)
```

Quotes a given string, escaping embedded quotes as necessary.

## See Also

`FILTER()`

## RAND

RAND — Random number

## Synopsis

```
RAND([min][max])
```

Chooses a random number within a range.

`RAND()` randomly picks an integer between *min* and *max*, inclusive, and returns that integer. If *min* is not specified, it defaults to 0. If *max* is not specified, it defaults to the C constant `INT_MAX`, which is 2,147,483,647 on 32-bit platforms. Note that `INT_MAX` is quite a bit larger on 64-bit platforms.

## REALTIME

REALTIME — Retrieves real-time data

## Synopsis

```
REALTIME(family|fieldmatch[value[delim1[delim2]]])
```

Real-time read/write functions. Use the above syntax for a read and the following syntax for a write:

```
REALTIME(family|fieldmatch|value|field)
```

# REGEX

REGEX — Compares based upon a regular expression

## Synopsis

```
REGEX("regular expression" data)
```

Matches based upon a regular expression.

# SET

SET — Sets a variable

## Synopsis

```
SET(varname=[value])
```

SET assigns a value to a channel variable. It is frequently used to set values that contain the character `|`, since that character is normally a delimiter when used with the application `Set()`.

## See Also

`Set()`

# SHA1

SHA1 — SHA-1 digest

## Synopsis

```
SHA1(data)
```

Computes the SHA-1 digest of *data*.

## See Also

`MD5()`

# SIP\_HEADER

SIP\_HEADER — Retrieves a SIP header

## Synopsis

```
SIP_HEADER(name [, number ] )
```

Gets the specified SIP header.

## SIPCHANINFO

SIPCHANINFO — Retrieves info on a SIP channel

### Synopsis

```
SIPCHANINFO(item)
```

Gets the specified SIP parameter from the current channel.

Valid items are:

peerip	The IP address of this SIP peer
recvip	The source IP address of this SIP peer
from	The SIP URI from the From: header
uri	The SIP URI from the Contact: header
useragent	The name of the SIP user agent
peername	The name of the SIP peer
t38passthrough	1 if T38 is offered or enabled in this channel, otherwise 0

## SIPPEER

SIPPEER — Retrieves info about a SIP peer

### Synopsis

```
SIPPEER(peername [ | item ] )
```

Gets SIP peer information.

Valid items are:

ip	The IP address of this peer. If <i>item</i> is not specified, the IP address will be given.
mailbox	This peer's configured mailbox.
context	The peer's configured context.
expire	The epoch time of the next registration expiration.
dynamic	Does this device register with Asterisk? (yes/no)

<code>callerid_name</code>	The Caller ID name configured on this peer.
<code>callerid_number</code>	The Caller ID number configured on this peer.
<code>status</code>	This peer's status (if <code>qualify=yes</code> ).
<code>regexten</code>	This peer's registration extension, if configured.
<code>limit</code>	This peer's call limit.
<code>curcalls</code>	Current number of calls. Only available if <code>call-limit</code> is set on this peer.
<code>language</code>	Default language for this peer.
<code>accountcode</code>	Account code for this peer.
<code>useragent</code>	The name of the SIP user agent.
<code>codecs</code>	The configured codecs for this peer.
<code>codec[x]</code>	Preferred codec index number <i>x</i> (beginning with zero).

## See Also

`IAXPEER()`

## SORT

**SORT** — Sorts a list

### Synopsis

```
SORT(key1:val1[...][,keyN:valN])
```

Sorts a list of `key/vals` into a list of keys, based upon the `vals` which can be any real number (float).

## SPEECH

**SPEECH** — Retrieves info on speech recognition results

### Synopsis

```
SPEECH(argument)
```

Gets information about speech recognition results.

## SPEECH\_ENGINE

**SPEECH\_ENGINE** — Modifies speech engine property



## Synopsis

```
SPEECH_ENGINE(name)=value)
```

Changes a specific attribute of the speech engine.

## SPEECH\_GRAMMAR

SPEECH\_GRAMMAR — Retrieves speech grammar information

## Synopsis

```
SPEECH_GRAMMAR(result number)
```

Gets the matched grammar of a result if available.

## SPEECH\_SCORE

SPEECH\_SCORE — Retrieves speech recognition confidence score

## Synopsis

```
SPEECH_SCORE(result number)
```

Gets the confidence score of a result.

## SPEECH\_TEXT

SPEECH\_TEXT — Retrieves recognized text

## Synopsis

```
SPEECH_TEXT(result number)
```

Gets the recognized text of a result.

## SPRINTF

SPRINTF — Formats a string

## Synopsis

```
SPRINTF(format | arg1 | ... argN)
```

Formats a variable or set of variables according to a format string.

The most common case for the use of `SPRINTF` is to zero-pad a number to a certain length:

```
; Returns 00123
exten => 123,1,Set(padfive=${SPRINTF(%05d,${EXTEN})})
```

Most of the format options listed in the manpage for `sprintf(3)` are also implemented in this dialplan function.

## See Also

`STRFTIME()`

# STAT

STAT — Evaluates filesystem attributes

## Synopsis

```
STAT(flag,filename)
```

Does a check on the specified file.

*flag* may be one of the following options:

e	Returns 1 if the file exists; 0 otherwise
s	Returns the size of the file, in bytes
f	Returns 1 if the path referenced is a regular file (and not a directory, symlink, socket, or device) or 0 otherwise
d	Returns 1 if the path referenced is a directory (and not a regular file, symlink, socket, or device) or 0 otherwise
M	Returns the epoch time when the file contents were last modified
C	Returns the epoch time when the file inode was last modified
m	Returns the permissions mode of the file (as an octal number)

# STRFTIME

STRFTIME — Formats the date and time

## Synopsis

```
STRFTIME([epoch][[timezone][format]])
```

Returns the current date/time in a specified format.

STRFTIME passes the *epoch* and *format* arguments directly to the underlying `strftime(3)` C library call, so check out that manpage for more information. The *timezone* parameter should be the name of a directory/file in `/usr/share/zoneinfo` (e.g., `America/Chicago` or `America/New_York`).

## See Also

STRPTIME( )

# STRPTIME

STRPTIME — Converts a string into a date and time

## Synopsis

```
STRPTIME(datetime | timezone | format)
```

Returns the epoch of the arbitrary date/time string structured as described in the format.

The purpose of this function is to take a formatted date/time and convert it back into seconds since the epoch (January 1st, 1970, at midnight GMT), so that you may do calculations with it, or simply convert it into some other date/time format.

STRPTIME passes the string and format directly to the underlying C library call `strptime(3)`, so check out that manpage for more information. The *timezone* parameter should be the name of a directory/file in `/usr/share/zoneinfo` (e.g., `America/Chicago` or `America/New_York`).

## See Also

STRFTIME( )

# TIMEOUT

TIMEOUT — Accesses channel timeout values

## Synopsis

```
TIMEOUT(timeouttype)
```

Gets or sets timeouts on the channel.

The timeouts that can be manipulated are:

<code>absolute</code>	The absolute maximum amount of time permitted for a call. A setting of 0 disables the timeout.
<code>digit</code>	The maximum amount of time permitted between digits when the user is typing in an extension. When this timeout expires, after the user has started to type in an extension, the extension will be considered complete, and will be interpreted. Note that if an extension typed in is valid, it will not have to timeout to be tested,

so typically at the expiry of this timeout, the extension will be considered invalid (and thus control will be passed to the `i` extension, or if it doesn't exist the call will be terminated). The default timeout is five seconds.

response

The maximum amount of time permitted after falling through a series of priorities for a channel in which the user may begin typing an extension. If the user does not type an extension in this amount of time, control will pass to the `t` extension if it exists, and if not the call will be terminated. The default timeout is 10 seconds.

## TXTCIDNAME

TXTCIDNAME — DNS lookup

### Synopsis

```
TXTCIDNAME(number)
```

Looks up a caller name via DNS.

## URIDECODE

URIDECODE — Decodes a URI

### Synopsis

```
URIDECODE(data)
```

Decodes a URI-encoded string according to RFC 2396.

### See Also

URIENCODE( )

## URIENCODE

URIENCODE — Encodes a URI

### Synopsis

```
URIENCODE(data)
```

Encodes a string to URI-safe encoding according to RFC 2396.

### See Also

URIDECODE( )

# VMCOUNT

VMCOUNT — Counts voicemail messages

## Synopsis

```
VMCOUNT(mailbox[@context][ |folder])
```

Counts the voicemail in a specified mailbox.

# Appendix C. Command-line Reference

This is the Asterisk command line interface reference. All available commands for Asterisk Business Edition version C.1 are listed below. Some commands have been marked as deprecated.



## Note

The Asterisk command line interface is currently undergoing an update to make the command syntax more consistent. Those commands which do not meet the current syntax format that are deprecated, but have not yet received the required update have been marked with a star (\*). All deprecated commands will refer to the new and preferred syntax, but those deprecated commands marked with a star may not yet have the new command implemented, but will be implemented in a future version (and is possible that the latest version of Asterisk Business Edition may have those commands implemented by the time you read this).

## bang

! — Execute a shell command

## Synopsis

This command will allow you to execute a shell command from the Asterisk console by prefixing any shell command with the bang (!) character.

## Example

```
*CLI> !echo "hello world"  
hello world
```

## abort halt

abort halt — Cancel a running halt

## Synopsis

Deprecated\* in favor of **core abort halt**

## add

add — Used to add dialplan entries from the Asterisk command line interface

## Synopsis

Deprecated in favor of **dialplan add**

# ael

ael — Commands for dealing with AEL

## Synopsis

This set command deals with AEL -- the alternative extension language for dialplan scriping.

Here we can go into greater detail.

## ael debug

This turns on debugging for a variety of things

## ael debug contexts

Debug the ael contexts

## ael debug macros

Debug the ael macros

## ael debug read

Debug the ael read, whatever that means

## ael debug tokens

Debug the ael tokens (you know, the kind of tokens they sell you at amusement parks)

## ael no

This turns off debugging for a variety of things

## ael no debug

Turns off all ael debugging

# agent

agent — Commands related to queue agents

## Synopsis

These commands are used to view the status of online agents and to logoff agents from the console.

## agent logoff <channel> [soft]

Set an agent offline.

## agent show online

Show status of agents.

## agi

agi — Various commands related to the Asterisk Gateway Interface (AGI)

### Synopsis

The following commands will give you more information about the Asterisk Gateway Interface, including commands you can run from your scripts which Asterisk calls with the **AGI()** dialplan application.

### agi debug

Deprecated\* in favor of **agi set debug**

### agi set

Set AGI options

### agi set debug

Enable AGI debugging

### agi set debug off

Disable AGI debugging

### agi dumphtml <filename>

Dump the AGI commands to <filename> in HTML format

### agi no debug

Deprecated\* in favor of **agi set debug off**

### agi show

List AGI commands or specific help

## answer

answer — Answer an incoming console call

### Synopsis

Deprecated in favor of **console answer**



## autoanswer

autoanswer — Sets/displays autoanswer

### Synopsis

Deprecated in favor of **console autoanswer**

## cdr status

cdr status — Display the Call Detail Record (CDR) status

### Synopsis

## clear profile

clear profile — Clear profiling information

### Synopsis

Deprecated in favor of **core clear profile**

## console

console — Various commands used for the control of the console channel

### Synopsis

Various commands that allow you to control the console channel driver (soundcard).

### console active [device]

If used without a parameter, displays which device is the current console. If a device is specified, the console sound device is changed to the device specified.

### console answer

Answers an incoming call on the console (OSS) channel.

### console autoanswer <on | off>

Enables or disables autoanswer feature. If used without argument, displays the current on/off status of autoanswer. The default value of autoanswer is in `oss.conf`.

### console boost <decibels>

Set the amount of boost on the console by <decibels> decibels.

## **console dial [extension[@context]]**

Dials a given extension (and context if specified)

## **console flash**

Flashes the call currently placed on the console.

## **console hangup**

Hangs up any call currently placed on the console.

## **console mute**

Mutes the microphone

## **console send text <message>**

Sends a text message for display on the remote terminal.

## **console transfer <extension>[@context]**

Transfers the currently connected call to the given extension (and context if specified)

## **console unmute**

Unmutes the microphone

## **convert**

convert — Convert an audio file from one format to another

## **Synopsis**

Deprecated in favor of **file convert**

## **core**

core — A set of core system commands

## **Synopsis**

These are a set of commands used to set and view information about your system.

## **core clear profile**

Clear profiling information

## core set

Options used to set various levels of core debugging information along with global variables.

### core set debug <level>

Set the debug level. Useful values are 0 through 4. See `logger.conf` for enabling debugging on the console.

### core set debug channel <channel> [off]

Enable or disable debugging information for a channel.

### core set global <name> <value>

Set a global variable <name> to <value>

### core set verbose <level>

Set the verbosity level. Useful values are 0 through 4. Verbosity will allow you to see what is happening in your system during a call.

## core show

Show information about related to a particular module in the system.

### core show application <application>

Displaying information about a particular application

### core show applications

Show all available applications

### core show audio codecs

Displays a list of audio codecs

### core show channel <channel>

Display information on a specific channel

### core show channels

Display information on channels

### core show channeltype <type>

Give more details on that channel type

### core show channeltypes

List available channel types

## **core show codec <codec>**

Shows a specific codec

## **core show codecs**

Displays a list of codecs

## **core show config mappings**

Display config mappings (file names to config engines)

## **core show file**

Display information about files used by Asterisk

## **core show file formats**

Display file formats

## **core show file mappings**

Display config mappings (file names to config engines)

## **core show function <function>**

Describe a specific dialplan function

## **core show functions**

Shows registered dialplan functions

## **core show globals**

Show global dialplan variables

## **core show hints**

Show dialplan hints

## **core show image**

Get information about image codecs and formats

## **core show image codecs**

Displays a list of image codecs

## **core show image formats**

Displays a list of image formats

## **core show license**

Show the license(s) for this copy of Asterisk

## **core show profile**

Display profiling info

## **core show switches**

Show alternative switches

## **core show sysinfo**

Show system information

## **core show threads**

Show running threads

## **core show translation**

Display translation matrix

## **core show uptime**

Show uptime information

## **core show version**

Display version info

## **core show video codecs**

Displays a list of video codecs

## **core show warranty**

Show the warranty (if any) for this copy of Asterisk

# **database**

database — A set of commands to manage the local Asterisk database (AstDB)

## **Synopsis**

These commands can be used to manage your local Asterisk database. The Asterisk database consists of keytrees which are made up of a family name (top level), followed by various key names which then contain a value. For example: /phones/100/username = jimmy, would be a family named 'phones', with a key of '100/username' (notice how we perform a 2nd level -- '100' alone could have been the keyname), followed by 'jimmy' as the value.

## database del <family> <key>

Removes database key/value

## database deltree <family>

Removes database keytree/values

## database get <family> <key>

Gets database value

## database put <family> <key> <value>

Adds/updates database value

## database show [family [keytree]]

Shows database contents

## database showkey <keytree>

Shows database contents

## Examples

```
*CLI> database del phones 100/username
*CLI> database deltree phones
*CLI> database get phones 100/username
*CLI> database put phones 100/username jimmy
*CLI> database showkey phones/100
```

## debug

debug — Get debugging information about your system

## Synopsis

Deprecated in favor of **core set debug**

## dialplan

dialplan — Various commands for getting and setting information about your dialplan

## Synopsis

These commands are used for getting and setting information about your dialplan from the Asterisk console. Recommended features are **dialplan reload** and **dialplan show**. Other dialplan commands can have unexpected effects and should only be used after testing. Always be sure to have a backup of your Asterisk configuration files.

### dialplan add

Add information to your dialplan from the Asterisk console

### dialplan add extension

Add new extension into context

### dialplan add ignorepat

Add new ignore pattern

### dialplan add include

Include context in other context

### dialplan reload

Reload extensions and *\*only\** extensions

### dialplan remove

Remove elements from your dialplan

### dialplan remove extension

Remove a specified extension

### dialplan remove ignorepat

Remove ignore pattern from context

### dialplan remove include

Remove a specified include from context

### dialplan save

Save dialplan changes you've made from the Asterisk console (not recommended). Saving the dialplan from the Asterisk console will strip out comments from `extensions.conf`.

### dialplan show [context]

Show dialplan, optionally a specific context

# dial

dial — Dial from the Asterisk console

## Synopsis

Deprecated in favor of **console dial**

# dnsmgr

dnsmgr — Asterisk DNS manager

## Synopsis

The Asterisk DNS manager is used to control how often Asterisk will lookup DNS information for caching. See the `dnsmgr.conf` file for configuration information.

## dnsmgr reload

Reloads the DNS manager configuration

## dnsmgr status

Display the DNS manager status

# dont include

dont include — Remove an included context from the dialplan

## Synopsis

Deprecated in favor of **dialplan remove include**

# dump agihtml

dump agihtml — Dump the AGI commands to <filename> in HTML format

## Synopsis

Deprecated in favor of **agi dumphtml**

# dundi

dundi — Distributed Universal Number Directory (DUNDi) related commands

## Synopsis



The following commands are related to the DUNDi protocol, used for distributing information across multiple Asterisk systems. DUNDi is configured in the `dundi.conf` file.

## **dundi debug**

Deprecated\* in favor of **dundi set debug**

## **dundi flush**

Flush DUNDi cache

## **dundi lookup <number>[@context] [bypass]**

Lookup an extension from the Asterisk console. If no context is specified, then lookup in the default context. If bypass keyword is present, then don't look in the DUNDi cache.

## **dundi no**

Deprecated\* in favor of **dundi set debug off** and **dundi store history off**

## **dundi precache <number>[@context]**

Precache a number in DUNDi

## **dundi query <entity>[@context]**

Query a DUNDi EID to get information about a remote system

## **dundi show**

Get information about DUNDi servers and peers

## **dundi show entityid**

Display Global Entity ID

## **dundi show mappings**

Show DUNDi mappings

## **dundi show peer <peer>**

Show information on a specific DUNDi peer

## **dundi show peers**

Show defined DUNDi peers

## **dundi show precache**

Show DUNDi precache

## dundi show requests

Show DUNDi requests

## dundi show trans

Show active DUNDi transactions

## dundi store history

Enable DUNDi historic records

## extensions reload

extensions reload — Reload dialplan

## Synopsis

Deprecated in favor of **dialplan reload**

## features show

features show — Lists configured features

## Synopsis

Lists configured features in Asterisk as configured from the `features.conf` file

## file convert

file convert — Convert an audio file

## Synopsis

Convert from *file\_in* to *file\_out*. If an absolute path is not given, the default Asterisk sounds directory will be used.

## file convert <file\_in> <file\_out>

Example:

```
*CLI> file convert tt-weasels.gsm tt-weasels.ulaw
```

## flash

flash — Flash an active console channel

## Synopsis

Deprecated in favor of **console flash**

## funcdevstate list

funcdevstate list — List of custom device states

## Synopsis

A list of custom device states as set by the `DEVICE_STATE ( )` dialplan function

## group show channels

group show channels — Display active channels with group(s)

## Synopsis

Display the currently active channels within group(s) as set by the `GROUP ( )` dialplan function

## help

help — Display help list, or specific help on a command

## Synopsis

Get a list of available options at the Asterisk console, or get more information about a specific command

## http show status

http show status — Display HTTP server status

## Synopsis

Get the status information about the built in mini-HTTP server, used by the Asterisk GUI

## iax2

iax2 — Commands related to the IAX2 channel

## Synopsis

The following are various commands you can use to get more information about the Inter-Asterisk eXchange version 2 (IAX2) protocol

## iax2 jb

Deprecated in favor of **iax2 set debug jb**

## iax2 no

Deprecated in favor of **iax2 set <feature> off**

## iax2 provision <host> <template> [forced]

Provisions the given peer or IP address using a template matching either 'template' or '\*' if the template is not found. If 'forced' is specified, even empty provisioning fields will be provisioned as empty fields.

## iax2 prune realtime [<peername> | all]

Prunes object(s) from the cache

## iax2 reload

Reloads IAX configuration from `iax.conf`

## iax2 set

Enable or disable debugging information for the IAX2 protocol

### iax2 set debug [off]

Enable or disable IAX debugging

### iax2 set debug jb [off]

Enable or disable IAX jitterbuffer debugging

### iax2 set debug trunk [off]

Enable or disable IAX trunk debugging

### iax2 set mtu <value>

Set the system-wide IAX IP mtu to <value> bytes net or zero to disable. Disabling means that the operating system must handle fragmentation of UDP packets when the IAX2 trunk packet exceeds the UDP payload size. This is substantially below the IP mtu. Try 1240 on ethernet. Must be 172 or greater for G.711 samples.

## iax2 show

Show various information about the IAX2 protocol

### iax2 show cache

Display currently cached IAX dialplan results

## **iax2 show channels**

Lists all currently active IAX channels

## **iax2 show firmware**

Lists all known IAX firmware images

## **iax2 show netstats**

Lists network status for all currently active IAX channels

## **iax2 show peer <peer>**

Display details on specific IAX peer

## **iax2 show peers [registered] [like <pattern>]**

Lists all known IAX2 peers. Optional 'registered' argument lists only peers with known addresses. Optional regular expression pattern is used to filter the peer list.

## **iax2 show provisioning**

Lists all known IAX provisioning templates or a specific one if specified

## **iax2 show registry**

Lists all registration requests and status

## **iax2 show stats**

Display statistics on IAX channel driver

## **iax2 show threads**

Lists status of IAX helper threads

## **iax2 show users [like <pattern>]**

Lists all known IAX users. Optional regular expression pattern is used to filter the user list.

## **iax2 test loss\_pct <percentage>**

For testing, throws away <percentage> percent of incoming packets

## **iax2 trunk**

Deprecated in favor of **iax2 set trunk debug**

## **include context**

include context — Include a dialplan context into another context

## Synopsis

Deprecated in favor of **dialplan include**

## indication

indication — Get information about country indications

### Synopsis

Get information about country indications. Indications can be found in the `indications.conf` file.

### **indication add <country> <indication> "<tonelists>"**

Add the given indication to the country

### **indication remove <country> <indication>**

Remove the given indication from the country

### **indication show [country [country [...]]]**

Display either a condensed for of all country/indications, or the indications for the specified countries

## init keys

init keys — Initializes private keys (by reading in pass code from the user)

### Synopsis

Deprecated in favor of **keys init**

## keys

keys — Commands related to initializing and viewing loaded keys

### Synopsis

Commands related to initializing and viewing loaded keys

### **keys init**

Initializes private keys (by reading in pass code from the user)

### **keys show**

Displays information about RSA keys known by Asterisk

# load

load — Load modules

## Synopsis

Deprecated in favor of **module load**

# local

local show channels — Display currently active Local channels

## Synopsis

Provides summary information on active local proxy channels

# logger

logger — Information about logging

## Synopsis

Get information about the logger module. Also see `logger.conf`.

## logger mute

Disables logging output to the current console, making it possible to gather information without being disturbed by scrolling lines.

## logger reload

Reloads the logger subsystem state. Use after restarting `syslogd(8)` if you are using syslog logging.

## logger rotate

Rotates and Reopens the log files

## logger show channels

List configured logger channels

# manager

manager — Display information about the Asterisk Manager Interface (AMI)

## Synopsis

Display information about the Asterisk Manager Interface (AMI). Also see `manager.conf`.

## **manager show command <command>**

Shows the detailed description for a specific Asterisk manager interface command

## **manager show commands**

Prints a listing of all the available Asterisk manager interface commands

## **manager show connected**

Prints a listing of the users that are currently connected to the Asterisk manager interface

## **manager show eventq**

Prints a listing of all events pending in the Asterisk manger event queue

## **manager show users**

Prints a listing of all managers that are currently configured on that system

## **manager show user <user>**

Display all information related to the manager user specified

## **meetme**

meetme — Control and display information about active MeetMe conferences

## **Synopsis**

Control and display information about active MeetMe conferences. Also see `meetme.conf`.

## **meetme kick <confno> <usernumber>**

Remove a user from a conference

## **meetme lock <confno>**

Lock a conference room

## **meetme mute <confno> <usernumber>**

Mute a user in a conference

## **meetme unlock <confno>**

Unlock a conference room



## **meetme unmute <confno> <username>**

Unmute a user in a conference

## **meetme list [concise] <confno>**

List users in a conference room

## **mgcp**

mgcp — Information about the Media Gateway Control Protocol (MGCP) channel driver

### **Synopsis**

Information about the MGCP channel driver

## **mgcp audit endpoint**

Lists the capabilities of an endpoint in the MGCP (Media Gateway Control Protocol) subsystem. mgcp debug **MUST** be on to see the results of this command.

## **mgcp reload**

Reload the MGCP channel driver

## **mgcp set debug [off]**

Enable or disable dumping of MGCP packets for debugging purposes

## **mgcp show endpoints**

Lists all endpoints known to the MGCP subsystem

## **mixmonitor**

mixmonitor — Start or stop MixMonitor for a provided channel

### **Synopsis**

Start or stop MixMonitor for a provided channel

## **mixmonitor <start|stop> <chan\_name> [args]**

The optional arguments are passed to the MixMonitor application when the 'start' command is used.

## **module**

module — Commands to control modules

## Synopsis

Commands to control modules in Asterisk, such as loading, unloading, and reloading, in addition to module information.

### **module load <module>**

Load the specified module into Asterisk

### **module reload [module]**

Reloads configuration files for all listed modules which support reloading, or for all supported modules if none are listed.

### **module show [like <keyword>]**

Shows Asterisk modules currently in use, and usage statistics.

### **module unload [-f|-h] <module\_1> [<module\_2> ... ]**

Unloads the specified module from Asterisk. The `-f` option causes the module to be unloaded even if it is in use (may cause a crash) and the `-h` module causes the module to be unloaded even if the module says it cannot, which almost always will cause a crash.

## moh

moh — Music on Hold module

## Synopsis

Get information about the currently loaded classes available to the Music on Hold module

### **moh show classes**

Lists all Music on Hold classes

### **moh files**

Lists all loaded file-based Music on Hold classes and their files.

### **moh reload**

Reloads the Music on Hold module

## mute

mute — Mute a console channel

## Synopsis

Deprecated in favor of **console mute**

## no debug channel

no debug channel — Stop debugging a channel

## Synopsis

Deprecated in favor of **core set debug channel <channel> off**

## odbc

odbc show — Show currently loaded ODBC classes

## odbc show [<class>]

List settings of a particular ODBC class. or, if not specified, all classes.

## originate

originate — Originate a call from the console

## Synopsis

There are two ways to use this command. A call can be originated between a channel and a specific application, or between a channel and an extension in the dialplan. This is similar to call files or the manager originate action. Calls originated with this command are given a timeout of 30 seconds.

Usage1: **originate <tech/data> application <appname> [appdata]**

This will originate a call between the specified channel tech/data and the given application. Arguments to the application are optional. If the given arguments to the application include spaces, all of the arguments to the application need to be placed in quotation marks.

Usage2: **originate <tech/data> extension [exten@][context]**

This will originate a call between the specified channel tech/data and the given extension. If no context is specified, the 'default' context will be used. If no extension is given, the 's' extension will be used.

## osp

osp show — Displays information on Open Settlement Protocol support

## Synopsis

Displays information on Open Settlement Protocol support

## oss

oss boost — Change the volume on the console channel

## Synopsis

Deprecated in favor of **console boost**

## pri

pri — Commands related to debugging Primary Rate Interfaces (PRI)

## Synopsis

Commands related to debugging Primary Rate Interfaces (PRI) as configured by Zaptel. See `zaptel.conf` and `zapata.conf`.

### pri debug span <span>

Enables debugging on a given PRI span

### pri intense debug <span>

Enables debugging down to the Q.921 level

### pri no debug span <span>

Disables debugging on a given PRI span

### pri set debug file <filename>

Sends PRI debug output to the specified file

### pri show

Show information about PRI spans

### pri show debug <span>

Displays current PRI debug settings

### pri show span <span>

Displays PRI Information

### pri show spans

Displays PRI Information

## **pri unset debug file**

Ends PRI debug output to file

## **queue**

queue — Command related to the Queue module

### **Synopsis**

Commands related to the Queue module such as adding, removing, and showing members of a queue.

**queue add member <channel> to <queue> [penalty <penalty>]**

Add a member to the queue

**queue remove member <channel> from <queue>**

Remove a member from the queue

**queue show**

List available queues

## **realtime**

realtime — Load and update variables in realtime

### **Synopsis**

Load and update variables in realtime.

**realtime load <family> <colmatch> <value>**

Prints out a list of variables using the Realtime driver.

**realtime update <family> <colmatch> <value>**

Update a single variable using the Realtime driver.

## **reload**

reload — Reload one or more modules

## Synopsis

Deprecated in favor of **module reload**

## remove

remove — Remove data from your dialplan

## Synopsis

Deprecated in favor of **dialplan remove**

## restart

restart — Restart Asterisk

## Synopsis

Restart your Asterisk system either immediately, or when all calls have completed.

## restart now

Restart immediately, dropping all active calls

## restart gracefully

Restart after all current calls have completed, but don't accept any new calls

## restart when convenient

Restart when the system has no active calls

## rtcp

rtcp — Debug Real Time Control Protocol (RTCP)

## Synopsis

Debug RTCP

## rtcp debug [off]

Enable or disable dumping of all RTCP packets

## rtcp debug [ip host[:port]]

Enable dumping of all RTCP packets to and from host

## rtcp stats [off]

Enable or disable dumping of RTCP stats

## rtp

rtp — Debug Real-time Transport Protocol

## Synopsis

The Real-time Transport Protocol is used to transmit the media (audio) for some VoIP protocols, such as SIP

## rtp debug

Enable RTP debugging

## rtp no debug

Disable RTP debugging

## save

save dialplan — Save changes to the dialplan

## Synopsis

Deprecated in favor of **dialplan save**

## say

say load

## Synopsis

**say load <new | old>**

## send

send text — Send text to a console channel

## Synopsis

Deprecated in favor of **console send text**

## set

set — Set various levels of debugging information

## Synopsis

Deprecated in favor of **core set**

## show

show — Show information about a module

## Synopsis

Deprecated in favor of **<module> show**

## sip

sip — Information about the SIP channel driver

## Synopsis

Get information about various aspects of the SIP channel driver

## sip debug

Deprecated in favor of **sip set debug**

## sip history [off]

Enables or disables recording of SIP dialog history for debugging purposes

## sip no debug

Deprecated in favor of **sip set debug off**

## sip notify <type> <peer> [<peer> [...]]

Send a NOTIFY message to a SIP peer or peers. Message types are defined in `sip_notify.conf`.

## sip prune realtime <peer | user> [<name> | all | like <pattern>]

Prunes object(s) from the cache. Optional regular expression pattern is used to filter the objects.

## sip reload

Reloads SIP configuration from `sip.conf`.



## **sip set**

Set various debugging options

### **sip set debug**

Enable SIP message debugging

### **sip set debug ip <ip host>[:port]**

Enable SIP message debugging for a specific peer

### **sip set debug peer <peer>**

Enables dumping of SIP packets to and from host. Requires peer to be registered.

## **sip show**

Show information about various aspects of the SIP channel driver

### **sip show channel <channel>**

Provides detailed status on a given SIP channel

### **sip show channels**

Lists all currently active SIP channels

### **sip show domains**

Lists all configured SIP local domains. Asterisk only responds to SIP messages to local domains.

### **sip show history**

Provides detailed dialog history on a given SIP channel.

### **sip show inuse [all]**

List all SIP users and peers usage counters and limits. Add option "all" to show all devices, not only those with a limit.

### **sip show objects**

Lists status of known SIP objects

### **sip show peer <name> [load]**

Shows all details on one SIP peer and the current status. Option "load" forces lookup of peer in realtime storage.

### **sip show peers [like <pattern>]**

Lists all known SIP peers. Optional regular expression pattern is used to filter the peer list.

## **sip show registry**

Lists all registration requests and status.

## **sip show settings**

Provides detailed list of the configuration of the SIP channel.

## **sip show subscriptions**

Lists active SIP subscriptions for extension states

## **sip show users [like <pattern>]**

Lists all known SIP users. Optional regular expression pattern is used to filter the user list.

## **sip show user <name> [load]**

Shows all details on one SIP user and the current status. Option "load" forces lookup of peer in realtime storage.

# **skinny**

skinny — Get information about the Skinny channel driver

## **Synopsis**

Get information about the Skinny channel driver and devices using it.

## **skinny reset <device\_id | all> [restart]**

Causes a Skinny device to reset itself, optionally with a full restart.

## **skinny set debug [off]**

Enables dumping of Skinny packets for debugging purposes.

## **skinny show**

Show information about Skinny devices and lines

## **skinny show devices**

Lists all devices known to the Skinny subsystem.

## **skinny show lines**

Lists all lines known to the Skinny subsystem.

# **sla**

sla — Show information about Shared Line Appearance

## Synopsis

Show information about Shared Line Appearance trunks and stations.

### **sla show trunks**

This will list all trunks defined in `sla.conf`.

### **sla show stations**

This will list all stations defined in `sla.conf`.

## soft

soft hangup — Hangup a call

## Synopsis

Deprecated\* in favor of **core send soft hangup <channel>**

### **soft hangup <channel>**

Request that a channel be hung up. The hangup takes effect the next time the driver reads or writes from the channel

## stop

stop — Stop the Asterisk process

## Synopsis

Deprecated\* in favor of **core stop**

### **stop gracefully**

Causes Asterisk to not accept new calls, and exit when all active calls have terminated normally.

### **stop now**

Shuts down a running Asterisk immediately, hanging up all active calls.

### **stop when convenient**

Causes Asterisk to perform a shutdown when all active calls have ended.

## stun

stun — Simple Traversal of UDP through NAT (STUN) debugging

## Synopsis

Debugging options related to STUN.

### stun debug

Enable STUN (Simple Traversal of UDP through NATs) debugging

### stun no debug

Disable STUN (Simple Traversal of UDP through NATs) debugging

## transcoder

transcoder show — Displays channel utilization of Zaptel transcoder(s)

## Synopsis

Displays channel utilization of Zaptel transcoder(s)

## transfer

transfer — Transfer a console channel

## Synopsis

Deprecated in favor of **console transfer**

## udptl

udptl — Debug UDPTL / T.38 Fax over IP

## Synopsis

Debug UDPTL / T.38 Fax over IP

### udptl debug [ip host[:port]]

Enable dumping of all UDPTL packets to and from host

### udptl no debug

Disable dumping of all UDPTL packets

## unload

unload — Unload a module

## Synopsis

Deprecated in favor of **module unload**

## unmute

unmute — Unmute a console channel

## Synopsis

Deprecated in favor of **console unmute**

## voicemail

voicemail — Commands related to the Voicemail module

## Synopsis

View information about mailboxes in the Voicemail system.

### voicemail show users [for <vm\_context>]

Lists all mailboxes currently set up

### voicemail show zones

Lists zone message formats

## zap

zap — Commands related to the Zaptel channel driver

## Synopsis

Commands to show information about Zaptel channels

### zap destroy channel <chan num>

DON'T USE THIS UNLESS YOU KNOW WHAT YOU ARE DOING. Immediately removes a given channel, whether it is in use or not.

### zap restart

Restarts the zaptel channels: destroys them all and then re-reads them from `zapata.conf`. Note that this will STOP any running CALL on zaptel channels.

## **zap show**

Show information about Zaptel channels

### **zap show cadences**

Shows all cadences currently defined

### **zap show channels**

Shows a list of available channels

### **zap show channel <chan num>**

Detailed information about a given channel

### **zap show status**

Shows a list of Zaptel cards with status

# Appendix D. AGI Reference

## ANSWER

### ANSWER

Answers the channel (if it is not already in an answered state).

Return values:

-1	Failure
0	Success

## CHANNEL STATUS

### CHANNEL STATUS

## Synopsis

```
CHANNEL STATUS [channelname]
```

Queries the status of the channel indicated by *channelname* or, if no channel is specified, the current channel.

Return values:

0	Channel is down and available
1	Channel is down, but reserved
2	Channel is off-hook
3	Digits have been dialed
4	Line is ringing
5	Line is up
6	Line is busy

## DATABASE DEL

### DATABASE DEL

## Synopsis

```
DATABASE DEL family key
```

Deletes an entry from the Asterisk database for the specified family and key.

Return values:

0	Failure
1	Success

## DATABASE DELTREE

DATABASE DELTREE

### Synopsis

```
DATABASE DELTREE family [keytree]
```

Deletes a family and/or keytree from the Asterisk database.

Return values:

0	Failure
1	Success

## DATABASE GET

DATABASE GET

### Synopsis

```
DATABASE GET family key
```

Retrieves a value from the Asterisk database for the specified family and key.

Return values:

0	Not set
1 ( <i>value</i> )	Value is set (and is included in parentheses)

## DATABASE PUT

DATABASE PUT

### Synopsis

```
DATABASE PUT family key value
```

Adds or updates an entry in the Asterisk database for the specified family and key, with the specified value.

Return values:



0	Failure
1	Success

## EXEC

EXEC

### Synopsis

```
EXEC application options
```

Executes the specified dialplan application, including options.

Return values:

-2	Failure to find the application
<i>value</i>	Return value of the application

## GET DATA

GET DATA

### Synopsis

```
GET DATA filename [timeout] [max_digits]
```

Plays the audio file specified by *filename* and accepts DTMF digits, up to the limit set by *max\_digits*. Similar to the Background( ) dialplan application.

Return value:

<i>value</i>	Digits received from the caller
--------------	---------------------------------

## GET FULL VARIABLE

GET FULL VARIABLE

### Synopsis

```
GET FULL VARIABLE variablename [channelname]
```

If the variable indicated by *variablename* is set, returns its value in parentheses. This command understands complex variable names and built-in variable names, unlike GET VARIABLE.

Return values:

0	No channel, or variable not set
1 ( <i>value</i> )	Value is retrieved (and is included in parentheses)

## GET OPTION

GET OPTION

### Synopsis

```
GET OPTION filename escape_digits [timeout]
```

Behaves the same as `STREAM FILE`, but has a *timeout* option (in seconds).

Return value:

<i>value</i>	ASCII value of digits received, in decimal
--------------	--

## GET VARIABLE

GET VARIABLE

### Synopsis

```
GET VARIABLE variablename
```

If the variable is set, returns its value in parentheses. This command does not understand complex variables or built-in variables; use the `GET FULL VARIABLE` command if your application requires these types of variables.

Return values:

0	No channel, or variable not set
1 ( <i>value</i> )	Value is retrieved (and is included in parentheses)

## HANGUP

HANGUP

### Synopsis

```
HANGUP [channelname]
```

Hangs up the specified channel or, if no channel is given, the current channel.

Return values:

-1	Specified channel does not exist
----	----------------------------------

1 Hangup was successful

## NoOp

NoOp

### Synopsis

```
NoOp [text]
```

Performs no operation. As a side effect, this command prints *text* to the Asterisk console. Usually used for debugging purposes.

Return value:

0 No channel, or variable not set

## RECEIVE CHAR

RECEIVE CHAR

### Synopsis

```
RECEIVE CHAR timeout
```

Receives a character of text on a channel. Specify a *timeout* in milliseconds as the maximum amount of time to wait for input, or set to 0 to wait infinitely. Note that most channels do not support the reception of text.

Return values:

-1 (hangup)	Failure or hangup
<i>char</i> (timeout)	Timeout
<i>value</i>	ASCII value of character, in decimal

## RECORD FILE

RECORD FILE

### Synopsis

```
RECORD FILE filename format escape_digits timeout [offset_samples] [BEEP] [s=silence]
```

Records the channel audio to the specified file until the reception of a defined escape (DTMF) digit. The *format* argument defines the type of file to be recorded (wav, gsm, etc.). The *timeout* argument is the maximum number of milliseconds the recording can last, and can be set to -1 for no timeout. The *offset\_samples* argument is optional; if provided, it will seek to the offset without exceeding the end of the file. The BEEP argument will play

a beep to the user to signify the start of the record operation. The *silence* argument is the number of seconds of silence allowed before the function returns despite the lack of DTMF digits or reaching the timeout. The silence value must be preceded by *s=* and is also optional.

Return values:

-1	Failure
0	Successful recording

## SAY ALPHA

SAY ALPHA

### Synopsis

```
SAY ALPHA number escape_digits
```

Says a given character string, returning early if any of the given DTMF digits are received on the channel.

Return values:

-1	Error or hangup
0	Playback completed without being interrupted by an escape digit
<i>value</i>	ASCII value of digit (if pressed), in decimal

## SAY DATE

SAY DATE

### Synopsis

```
SAY DATE date escape_digits
```

Says a given *date*, returning early if any of the given DTMF digits are received on the channel. The *date* is the number of seconds elapsed since 00:00:00 on January 1, 1970, Coordinated Universal Time (UTC).

Return values:

-1	Error or hangup
0	Playback completed without being interrupted by an escape digit
<i>value</i>	ASCII value of digit (if pressed), in decimal

## SAY DATETIME

SAY DATETIME

## Synopsis

```
SAY DATETIME datetime escape_digits [format] [timezone]
```

Says the given *datetime*, returning early if any of the given DTMF digits are received on the channel. The *datetime* is the number of seconds elapsed since 00:00:00 on January 1, 1970, Coordinated Universal Time (UTC). The optional *format* argument is the format in which the time should be spoken. (See *voicemail.conf* for a complete description of the format options.) *format* defaults to "ABdY 'digits/at' IMp". Acceptable values for *timezone* can be found in */usr/share/zoneinfo/*. *timezone* defaults to the default time zone of the Asterisk server.

Return values:

-1	Error or hangup
0	Playback completed without being interrupted by an escape digit
<i>value</i>	ASCII value of digit (if pressed), in decimal

## SAY DIGITS

SAY DIGITS

## Synopsis

```
SAY DIGITS number escape_digits
```

Says a given digit string, returning early if any of the given DTMF digits are received on the channel.

Return values:

-1	Error or hangup
0	Playback completed without being interrupted by an escape digit
<i>value</i>	ASCII value of digit (if pressed), in decimal

## SAY NUMBER

SAY NUMBER

## Synopsis

```
SAY NUMBER number escape_digits
```

Says a given number, returning early if any of the given DTMF digits are received on the channel.

Return values:

-1	Error or hangup
----	-----------------

0 Playback completed without being interrupted by an escape digit  
*value* ASCII value of digit (if pressed), in decimal

## SAY PHONETIC

SAY PHONETIC

### Synopsis

SAY PHONETIC *string escape\_digits*

Says a given character string with phonetics, returning early if any of the given DTMF digits are received on the channel.

Return values:

-1 Error or hangup  
0 Playback completed without being interrupted by an escape digit  
*value* ASCII value of digit (if pressed), in decimal

## SAY TIME

SAY TIME

### Synopsis

SAY TIME *time escape\_digits*

Says the indicated *time*, returning early if any of the given DTMF digits are received on the channel. The *time* is the number of seconds elapsed since 00:00:00 on January 1, 1970, Coordinated Universal Time (UTC).

Return values:

-1 Error or hangup  
0 Playback completed without being interrupted by an escape digit  
*value* ASCII value of digit (if pressed), in decimal

## SEND IMAGE

SEND IMAGE

### Synopsis

SEND IMAGE *image*

Sends the given image on the current channel. Most channels do not support the transmission of images. Image names should not include extensions.

Return values:

-1	Error or hangup
0	Image sent, or channel does not support sending an image

## SEND TEXT

SEND TEXT

### Synopsis

```
SEND TEXT "text_to_send"
```

Sends the specified text on the current channel. Most channels do not support the transmission of text. Text consisting of more than one word should be placed in quotes, since the command accepts only a single argument.

Return values:

-1	Error or hangup
0	Text sent, or channel does not support sending text

## SET AUTOHANGUP

SET AUTOHANGUP

### Synopsis

```
SET AUTOHANGUP time
```

Causes the channel to automatically be hung up once *time* seconds have elapsed. Of course, it can be hung up before then as well. Setting *time* to 0 will cause the `autohangup` feature to be disabled on this channel.

Return value:

0	Autohangup has been set
---	-------------------------

## SET CALLERID

SET CALLERID

### Synopsis

```
SET CALLERID number
```

Changes the Caller ID of the current channel.

Return value:

1                                      Caller ID has been set

## SET CONTEXT

SET CONTEXT

### Synopsis

```
SET CONTEXT context
```

Sets the *context* for continuation upon exiting the AGI application.

Return value:

0                                      Context has been set

## SET EXTENSION

SET EXTENSION

### Synopsis

```
SET EXTENSION extension
```

Changes the *extension* for continuation upon exiting the AGI application.

Return value:

0                                      Extension has been set

## SET MUSIC ON

SET MUSIC ON

### Synopsis

```
SET MUSIC ON [on|off] [class]
```

Enables/disables the music-on-hold generator. If *class* is not specified, the default music-on-hold class will be used.

Return value:

0                                      Always returns 0



# SET PRIORITY

SET PRIORITY

## Synopsis

```
SET PRIORITY priority
```

Changes the priority for continuation upon exiting the AGI application. *priority* must be a valid priority or label.

Return value:

0	Priority has been set
---	-----------------------

# SET VARIABLE

SET VARIABLE

## Synopsis

```
SET VARIABLE variablename value
```

Sets or updates the *value* for the variable name specified by *variablename*. If the variable does not exist, it is created.

Return value:

1	Variable has been set
---	-----------------------

# STREAM FILE

STREAM FILE

## Synopsis

```
STREAM FILE filename escape_digits [sample_offset]
```

Play the audio file indicated by *filename*, allowing playback to be interrupted by the digits specified by *escape\_digits*, if any. Use double quotes for the digits if you wish none to be permitted. If *sample\_offset* is provided, the audio will seek to *sample\_offset* before playback starts.

Remember, the file extension must not be included in the filename.

Return values:

0	Playback completed with no digit pressed
-1	Error or hangup

*value* ASCII value of digit (if pressed), in decimal

## TDD MODE

TDD MODE

### Synopsis

Enables and disables Telecommunications Devices for the Deaf (TDD) transmission/reception on this channel. Return values:

0	Channel not TDD-capable
1	Success

## VERBOSE

VERBOSE

### Synopsis

`VERBOSE message level`

Sends *message* to the console via the verbose message system. The *level* argument is the minimum verbosity level at which the message will appear on the Asterisk command-line interface.

Return value:

0	Always returns 0
---	------------------

## WAIT FOR DIGIT

WAIT FOR DIGIT

### Synopsis

`WAIT FOR DIGIT timeout`

Waits up to *timeout* milliseconds for the channel to receive a DTMF digit. Use `-1` for the *timeout* value if you want the call to block indefinitely.

Return values:

-1	Error or channel failure
0	Timeout
<i>value</i>	ASCII value of digit (if pressed), in decimal

# Appendix E. Asterisk Manager Interface Actions

The following is a list of actions you can perform using the Manager Interface, or AMI.

## AbsoluteTimeout

AbsoluteTimeout — Sets the AbsoluteTimeout on a channel

Hangs up a channel after a certain time.

### Parameters

Channel	[required] The name of the channel on which to set the absolute timeout.
Timeout	[required] The maximum duration of the call, in seconds.
ActionID	[optional] An identifier that can be used to identify the response to this action.

### Notes

Asterisk will acknowledge the timeout setting with a `Timeout Set` message.

### Privilege

call, all

### Example

```
Action: AbsoluteTimeout
Channel: SIP/testphone-10210698
Timeout: 15
ActionID: 12345
```

```
Response: Success
Message: Timeout Set
ActionID: 12345
```

## AgentCallbackLogin

AgentCallbackLogin — Sets an agent as logged in to the queue system in callback mode

Logs the specified agent in to the Asterisk queue system in callback mode. When a call is distributed to this agent, it will ring the specified extension.

## Parameters

Agent	[required] Agent ID of the agent to log in to the system, as specified in <i>agents.conf</i> .
Exten	[required] Extension to use for callback.
Context	[optional] Context to use for callback.
AckCall	[optional] Set to <code>true</code> to require an acknowledgement (the agent pressing the # key) to accept the call when agent is called back.
WrapupTime	[optional] The minimum amount of time after disconnecting before the agent will receive a new call.
ActionID	[optional] An identifier which can be used to identify the response to this action.

## Privilege

agent, all

## Example

```
Action: AgentCallbackLogin
Agent: 1001
Exten: 201
Context: Lab
ActionID: 24242424
```

```
Response: Success
Message: Agent logged in
ActionID: 24242424
```

```
Event: Agentcallbacklogin
Privilege: agent,all
Agent: 1001
Loginchan: 201@Lab
```

## Notes

The `AgentCallbackLogin` action (along with the `AgentCallbackLogin( )` application) has been deprecated. It is suggested you use the `QueueAdd` action instead. See *doc/queues-with-callback-members.txt* in the Asterisk source code for more information.

## AgentLogoff

**AgentLogoff** — Sets an agent as no longer logged in

Logs off the specified agent for the queue system.

## Parameters

Agent	[required] Agent ID of the agent to log off.
Soft	[optional] Set to true to not hangup existing calls.
ActionID	[optional] An identifier which can be used to identify the response to this action.

## Privilege

agent, all

## Example

```
Action: AgentLogoff
Agent: 1001
Soft: true
ActionID: blahblahblah
```

```
Response: Success
Message: Agent logged out
ActionID: blahblahblah
```

```
Event: Agentcallbacklogoff
Privilege: agent, all
Agent: 1001
Reason: CommandLogoff
Loginchan: 201@Lab
Logintime: 5698
```

## Agents

Agents — Lists agents and their status

This action lists information about all configured agents.

## Privilege

agent, all

## Example

```
Action: Agents
ActionID: mylistofagents
```

```
Response: Success
Message: Agents will follow
ActionID: mylistofagents
```

```
Event: Agents
Agent: 1001
Name: Jared Smith
Status: AGENT_IDLE
LoggedInChan: 201@Lab
LoggedInTime: 1173237646
TalkingTo: n/a
ActionID: mylistofagents
```

```
Event: Agents
Agent: 1002
Name: Leif Madsen
Status: AGENT_LOGGEDOFF
LoggedInChan: n/a
LoggedInTime: 0
TalkingTo: n/a
ActionID: mylistofagents
```

```
Event: Agents
Agent: 1003
Name: Jim VanMeggelen
Status: AGENT_LOGGEDOFF
LoggedInChan: n/a
LoggedInTime: 0
TalkingTo: n/a
ActionID: mylistofagents
```

```
Event: AgentsComplete
ActionID: mylistofagents
```

## ChangeMonitor

ChangeMonitor — Changes monitoring filename of a channel

The ChangeMonitor action may be used to change the file started by a previous Monitor action. The following parameters may be used to control this.

### Parameters

Channel	[required] Used to specify the channel to record.
File	[required] The new filename in which the monitored channel will be recorded.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

call, all

## Example

```
Action: ChangeMonitor
Channel: SIP/linksys-084c63c0
File: new-test-recording
ActionID: 555544443333
```

```
Response: Success
ActionID: 555544443333
Message: Changed monitor filename
```

## Command

Command — Executes an Asterisk CLI command

Runs an Asterisk CLI command as if it had been run from the CLI.

## Parameters

Command	[required] Asterisk CLI command to run.
ActionID	[optional] An action identifier that can be used to identify the response from Asterisk.

## Privilege

command, all

## Example

```
Action: Command
Command: core show version
ActionID: 0123456789abcdef
```

```
Response: Follows
Privilege: Command
ActionID: 0123456789abcdef
Asterisk SVN-branch-1.4-r55869 built by jsmith @ hockey on a ppc running Linux
on 2007-02-21 16:55:26 UTC
--END COMMAND--
```

# DBGet

DBGet — Gets AstDB entry

This action retrieves a value from the AstDB database.

## Parameters

Family	[required] The AstDB key family from which to retrieve the value
Key	[required] The name of the AstDB key.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

system, all

## Example

```
Action: DBGet
Family: testfamily
Key: mykey
ActionID: 01234-astdb-43210
```

```
Response: Success
Message: Result will follow
ActionID: 01234-astdb-43210
```

```
Event: DBGetResponse
Family: testfamily
Key: mykey
Val: 42
ActionID: 01234-astdb-43210
```

# DBPut

DBPut — Puts DB entry

Sets a key value in the AstDB database.

## Parameters

Family	[required] The AstDB key family in which to set the value.
--------	--



Key	[required] The name of the AstDB key.
Val	[required ]The value to assign to the key.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

system, all

## Example

```
Action: DBPut
Family: testfamily
Key: mykey
Val: 42
ActionID: testing123
```

```
Response: Success
Message: Updated database successfully
ActionID: testing123
```

## Events

Events — Controls event flow

Enables or disables sending of events to this manager connection.

## Parameters

EventMask	[required] Set to on if all events should be sent, off if events should not be sent, or system, call, log to select which type of events should be sent to this manager connection.
ActionID	[optional] An identifier which can be used to identify the response to this action.

## Privilege

none

## Example

```
Action: Events
EventMask: off
ActionID: 2938416
```

```
Response: Events Off
ActionID: 2938416
```

```
Action: Events
EventMask: log,call
ActionID: blah1234
```

```
Response: Events On
ActionID: blah1234
```

## ExtensionState

ExtensionState — Checks extension status

This command reports the extension state for the given extension. If the extension has a hint, this will report the status of the device connected to the extension.

## Parameters

Exten	[required] The name of the extension to check.
Context	[required] The name of the context that contains the extension.
ActionId	[optional] An action identifier that can be used to identify this manager transaction.

## Privilege

call,all

## Example

```
Action: ExtensionState
Exten: 200
Context: lab
ActionID: 54321
```

```
Response: Success
ActionID: 54321
Message: Extension Status
Exten: 200
Context: lab
Hint: SIP/testphone
Status: 0
```

## Notes

The following are the possible extension states:

- 2 Extension removed
- 1 Extension hint not found
- 0 Idle
- 1 In use
- 2 Busy

## GetConfig

GetConfig — Retrieves configuration

Retrieves the data from an Asterisk configuration file.

## Parameters

Filename	[required]Name of the configuration file to retrieve.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

config,all

## Example

```
Action: GetConfig
Filename: musiconhold.conf
ActionID: 09235012

Response: Success
ActionID: 09235012
Category-000000: default
Line-000000-000000: mode=files
Line-000000-000001: directory=/var/lib/asterisk/moh
Line-000000-000002: random=yes
```

## GetVar

GetVar — Retrieves the value of a variable

Gets the value of a local channel variable or global variable

## Parameters

Channel	[optional] The name of the channel from which to retrieve the variable value.
Variable	[required] Variable name.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

call, all

## Example

```
Action: GetVar
Channel: SIP/linksys2-1020e2b0
Variable: SIPUSERAGENT
ActionID: abcd1234
```

```
Response: Success
Variable: SIPUSERAGENT
Value: Linksys/SPA962-5.1.5
ActionID: abcd1234
```

```
Action: GetVar
Variable: TRUNKMSD
```

```
Response: Success
Variable: TRUNKMSD
Value: 1
```

## Hangup

Hangup — Hangs up channel

Hangs up the specified channel.

## Parameters

Channel	[optional] The channel name to be hung up
ActionID	[optional] An identifier that can be used to identify the response to this action

## Privilege

call, all

## Example

```
Action: Hangup
Channel: SIP/labrat-8d3a
Response: Success
Message: Channel Hungup
```

```
Event: Hangup
Privilege: call,all
Channel: SIP/labrat-8d3a
Uniqueid: 1173448206.0
Cause: 0
Cause-txt: Unknown
```

## IAXNetstats

IAXNetstats — Shows IAX statistics

Shows a summary of network statistics for the IAX2 channel driver.

### Privilege

none

### Example

```
Action: IAXNetstats
```

```
IAX2/216.207.245.8:4569-1 608 -1 0 -1 -1 0 -1 1 288 508 10 1 3 0 0
```

## IAXPeers

IAXPeers — Lists IAX peers

Lists all IAX2 peers and their current status.

### Privilege

none

### Example

```
Action: IAXPeers
```

Name/Username	Host	Mask	Port	Status
---------------	------	------	------	--------

```
jared/jared      192.168.0.71    (S)  255.255.255.255  4569    UNREACHABLE
jaredsmith       192.168.0.72    (S)  255.255.255.255  4569    OK (43 ms)
arrivaltel/8017  172.20.95.2     (S)  255.255.255.255  4569    Unmonitored
sokol/jsmith     172.17.122.217 (S)  255.255.255.255  4569    OK (48 ms)
demo/asterisk    216.207.245.47 (S)  255.255.255.255  4569    Unmonitored
5 iax2 peers [2 online, 1 offline, 2 unmonitored]
```

## ListCommands

ListCommands — Lists the manager commands

Lists the action name and synopsis for every Asterisk Manager Interface action.

### Privilege

none

### Example

**Action: ListCommands**

Response: Success

AbsoluteTimeout: Set Absolute Timeout (Priv: call,all)

AgentCallbackLogin: Sets an agent as logged in by callback (Priv: agent,all)

AgentLogoff: Sets an agent as no longer logged in (Priv: agent,all)

. . .

ZapTransfer: Transfer Zap Channel (Priv: <none>)

## Logoff

Logoff — Logs off manager session

Logs off this manager session.

### Privilege

none

### Example

**Action: Logoff**

Response: Goodbye

Message: Thanks for all the fish.

# MailboxCount

MailboxCount — Checks mailbox message count

Retrieves the number of messages for the specified voice mailbox.

## Privilege

call, all

## Example

```
Action: MailboxCount
Mailbox: 100@lab
ActionID: 54321abcde
Response: Success
ActionID: 54321abcde
Message: Mailbox Message Count
Mailbox: 100@lab
NewMessages: 2
OldMessages: 0
```

# MailboxStatus

MailboxStatus — Checks mailbox status

Checks the status for the specified voicemail box.

## Parameters

Mailbox	[required] The full mailbox ID, including mailbox and context ( <i>box context</i> ).
ActionID	[optional] A unique identifier that can be used to identify responses to this manager command.

## Privilege

call, all

## Example

```
Action: MailboxStatus
Mailbox: 100@lab
ActionID: abcdef0123456789
```

```
Response: Success
ActionID: abcdef0123456789
Message: Mailbox Status
Mailbox: 100@lab
Waiting: 1
```

## MeetmeMute

MeetmeMute — Mutes a MeetMe user

Mutes a particular user in a MeetMe conference bridge.

### Parameters

Meetme	[required] The MeetMe conference bridge number.
Usumnum	[required] The user number in the specified bridge.
ActionID	[optional] A unique identifier to help you identify responses to this command.

### Privilege

call,all

### Example

```
Action: MeetmeMute
Meetme: 104
Usumnum: 1
ActionID: 5432154321

Response: Success
ActionID: 5432154321
Message: User muted

Event: MeetmeMute
Privilege: call,all
Channel: SIP/linksys2-10211dc0
Uniqueid: 1174008176.3
Meetme: 104
Usumnum: 1
Status: on
```

### Notes

To find the Usumnum number for a particular caller, watch the Asterisk Manager Interface when a new member joins a conference bridge. When it happens, you'll see an event like this:



```
Event: MeetmeJoin
Privilege: call,all
Channel: SIP/linksys2-10211dc0
Uniqueid: 1174008176.3
Meetme: 104
Usernum: 1
```

## MeetMeUnmute

MeetMeUnmute — Unmutes a MeetMe user

Unmutes the specified user in a MeetMe conference bridge.

### Parameters

Meetme	[required] The MeetMe conference bridge number.
Usernum	[required] The user number in the specified bridge.
ActionID	[optional] A unique identifier to help you identify responses to this command.

### Privilege

call,all

### Example

```
Action: MeetmeUnmute
Meetme: 104
Usernum: 1
ActionID: abcdefghijklmnop
```

```
Response: Success
ActionID: abcdefghijklmnop
Message: User unmuted
```

```
Event: MeetmeMute
Privilege: call,all
Channel: SIP/linksys2-10211dc0
Uniqueid: 1174008176.3
Meetme: 104
Usernum: 1
Status: off
```

## Monitor

Monitor — Monitors a channel

Records the audio on a channel to the specified file.

## Parameters

Channel	[required] Specifies the channel to be recorded.
File	[optional] The name of the file in which to record the channel. The path defaults to the Asterisk monitor spool directory, which is usually <i>/var/spool/asterisk/monitor</i> . If no filename is specified, the filename will be the name of the channel, with slashes replaced with dashes.
Format	[optional] The audio format in which to record the channel. Defaults to wav.
Mix	[optional] A Boolean flag specifying whether or not Asterisk should mix the inbound and outbound audio from the channel in to a single file.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

call, all

## Example

```
Action: Monitor
Channel: SIP/linksys2-10216e38
Filename: test-recording
Format: gsm
Mix: true

Response: Success
Message: Started monitoring channel
```

## Originate

Originate — Originates call

Generates an outbound call from Asterisk, and connect the channel to a context/extension/priority combination or dialplan application.

## Parameters

Channel	[required] Channel name to call. Once the called channel has answered, the control of the call will be passed to the specified Exten/Context/Priority or Application.
Exten	[optional] Extension to use (requires Context and Priority).

Context	[optional] Context to use (requires Exten and Priority).
Priority	[optional] Priority to use (requires Exten and Context).
Application	[optional] Application to use.
Data	[optional] Data to pass as parameters to the application (requires Application).
Timeout	[optional] How long to wait for call to be answered (in ms).
CallerID	[optional] Caller ID to be set on the outgoing channel.
Variable	[optional] Channel variable to set. Multiple variable headers are allowed.
Account	[optional] Account code.
Async	[optional] Set to <code>true</code> for asynchronous origination. Asynchronous origination allows you to originate one or more calls without waiting for an immediate response.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

call, all

## Example

```
Action: Originate
Channel: SIP/linksys2
Context: lab
Exten: 201
Priority: 1
CallerID:
```

```
Response: Success
Message: Originate successfully queued
```

```
Action: Originate
Application: MusicOnHold
Data: default
Channel: SIP/linksys2
```

```
Response: Success
Message: Originate successfully queued
```

## Park

Park — Parks a channel

Parks the specified channel in the parking lot.

## Parameters

Channel	[required] Channel name to park.
Channel2	[required] Channel to announce park info to (and return the call to if the parking times out).
Timeout	[optional] Number of milliseconds to wait before callback.
ActionID	[optional] An identifier which can be used to identify the response to this action.

## Privilege

call, all

## Example

```
Action: Park
Channel: SIP/linksys-10228fb0
Channel2: SIP/linksys2-10231520
Timeout: 45
ActionID: parking-test-01
```

```
Response: Success
ActionID: parking-test-01
Message: Park successful
```

## Notes

The call parking lot is configured in *features.conf* in the Asterisk configuration directory.

## ParkedCalls

ParkedCalls — Lists parked calls

Lists any calls that are parked in the call parking lot.

## Privilege

none

## Example

```
Action: ParkedCalls
```

```
ActionID: 0982350175
```

```
Response: Success
```

```
ActionID: 0982350175
```

```
Message: Parked calls will follow
```

```
Event: ParkedCall
```

```
Exten: 701
```

```
Channel: SIP/linksys2-101f98a8
```

```
From: SIP/linksys2-101f98a8
```

```
Timeout: 26
```

```
CallerID: linksys2
```

```
CallerIDName: linksys2
```

```
ActionID: 0982350175
```

```
Event: ParkedCallsComplete
```

```
ActionID: 0982350175
```

## Notes

The call parking lot is configured in *features.conf* in the Asterisk configuration directory.

## PauseMonitor

PauseMonitor — Pauses the recording of a channel

Pauses the monitoring (recording) of a channel if it is being monitored.

## Parameters

Channel [required] The channel identifier of the channel that is currently being monitored.

ActionID [optional] An identifier that can be used to identify the response to this action.

## Privilege

call, all

## Example

```
Action: PauseMonitor
```

```
Channel: SIP/linksys2-10212040
```

```
ActionID: 987987987987
```

```
Response: Success
```

```
ActionID: 987987987987
Message: Paused monitoring of the channel
```

## Ping

Ping — Keeps connection alive

Queries the Asterisk server to make sure it is still responding. Asterisk will respond with a Pong response. This command can also be used to keep the manager connection from timing out.

## Example

```
Action: Ping
Response: Pong
```

## PlayDTMF

PlayDTMF — Plays DTMF on a channel

Plays a DTMF digit on the specified channel.

## Parameters

Channel	[required] The identifier for the channel on which to send the DTMF digit.
Digit	[required] The DTMF digit to play on the channel.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

call, all

## Example

```
Action: PlayDTMF
Channel: Local/201@lab-157a,1
Digit: 9

Response: Success
Message: DTMF successfully queued
```

# QueueAdd

QueueAdd — Adds a member to the specified queue

Adds a queue member to a call queue.

## Parameters

Queue	[required] The name of the queue.
Interface	[required] The name of the member to add to the queue. This will be a technology and resource, such as SIP/Jane or Local/203@lab/n. Agents (as defined in <i>agents.conf</i> ) can also be added by using the Agent/1234 syntax.
MemberName	[optional] This is a human-readable alias for the interface, and will appear in the queue statistics and queue logs.
Penalty	[optional] A numerical penalty to apply to this queue member. Asterisk will distribute calls to members with higher penalties only after attempting to distribute the call to all members with a lower penalty.
Paused	[optional] Whether or not the member should be initially paused.
ActionID	[optional] An action identifier that you can use to identify the response to this manager transaction.

## Privilege

agent,all

## Example

```
Action: QueueAdd
Queue: myqueue
Interface: SIP/testphone
MemberName: Jared Smith
Penalty: 2
Paused: no
ActionID: 4242424242
```

```
Response: Success
ActionID: 4242424242
Message: Added interface to queue
```

```
Event: QueueMemberAdded
Privilege: agent,all
Queue: myqueue
Location: SIP/testphone
MemberName: Jared Smith
Membership: dynamic
```

```
Penalty: 2
CallsTaken: 0
LastCall: 0
Status: 1
Paused: 0
```

## QueuePause

QueuePause — Pauses or unpauses a member in a call queue

Pauses or unpauses a member in a call queue.

### Parameters

Interface	[required] The name of the interface to pause or unpaue.
Paused	[required] Whether or not the interface should be paused. Set to <code>true</code> to pause the member, or <code>false</code> to unpaue the member.
Queue	[optional] The name of the queue in which to pause or unpaue this member. If not specified, the member will be paused or unpaues in all the queues it is a member of.
ActionID	[optional] An identifier that can be used to identify the response to this action.

### Privilege

agent, all

### Example

```
Action: QueuePause
Interface: SIP/testphone
Paused: true
Queue: myqueue
```

```
Response: Success
Message: Interface paused successfully
```

```
Event: QueueMemberPaused
Privilege: agent, all
Queue: myqueue
Location: SIP/testphone
MemberName: Jared Smith
Paused: 1
```

```
Action: QueuePause
Interface: SIP/testphone
Paused: false
Response: Success
```



```
Message: Interface unpaused successfully
```

```
Event: QueueMemberPaused
Privilege: agent,all
Queue: myqueue
Location: SIP/testphone
MemberName: Jared Smith
Paused: 0
```

## QueueRemove

QueueRemove — Removes interface from queue

Removes interface from queue.

### Parameters

Queue	[required] Which queue to remove the member from.
Interface	[required] The interface (member) to remove from the specified queue.
ActionID	[optional] An identifier that can be used to identify the response to this action.

### Privilege

agent,all

### Example

```
Action: QueueRemove
Queue: myqueue
Interface: SIP/testphone
```

```
Response: Success
Message: Removed interface from queue
```

```
Event: QueueMemberRemoved
Privilege: agent,all
Queue: myqueue
Location: SIP/testphone
MemberName: Jared Smith
```

## QueueStatus

QueueStatus — Checks queue status

Checks the status of one or more queues.

## Parameters

Queue	[optional] If specified, limits the response to the status of the specified queue.
Member	[optional] An action identifier that you can use to identify the response to this manager transaction.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

none

## Example

```
Action: QueueStatus
Queue: inbound-queue
ActionID: 11223344556677889900
```

```
Response: Success
ActionID: 11223344556677889900
Message: Queue status will follow
```

```
Event: QueueParams
Queue: inbound-queue
Max: 0
Calls: 1
Holdtime: 99
Completed: 540
Abandoned: 51
ServiceLevel: 60
ServicelevelPerf: 50.4
Weight: 0
ActionID: 11223344556677889900
```

```
Event: QueueMember
Queue: inbound-queue
Location: Local/4020@agents/n
Membership: dynamic
Penalty: 2
CallsTaken: 25
LastCall: 1175563440
Status: 2
Paused: 0
ActionID: 11223344556677889900
```

```
Event: QueueEntry
Queue: inbound-queue
Position: 1
Channel: Zap/25-1
CallerID: 8012317154
CallerIDName: JOHN Q PUBLIC
```

```
Wait: 377
ActionID: 11223344556677889900

Event: QueueStatusComplete
ActionID: 11223344556677889900
```

## Queues

Queues — Shows basic queue information

Shows the call queues along with the queue members, callers, and basic queue statistics.

## Privilege

none

## Example

### Action: Queues

```
inbound-queue has 0 calls (max unlimited) in 'rrmemory' strategy (81s holdtime),
W:0, C:542, A:51, SL:50.4% within 60s
Members:
    Local/4020@agents/n with penalty 2 (dynamic) (Unknown) has taken
    27 calls (last was 124 secs ago)
No Callers
```

## Notes

This manager command gives the same output as the `show queues` command from the Asterisk command-line interface. However, the output of this command is somewhat hard to parse programmatically. You may want to use the `QueueStatus` command instead.

## Redirect

Redirect — Redirects (transfers) a channel

Redirects a channel to a new context, extension, and priority in the dialplan.

## Parameters

Channel	[required] Channel to redirect.
ExtraChannel	[optional] Channel identifier of the second call leg to transfer.
ActionID	[optional] An identifier that can be used to identify the response to this action.

Exten	[required] Extension in the dialplan to transfer to.
Context	[required] Context to transfer to.
Priority	[required] Priority to transfer to.

## Privilege

call, all

## Example

```
Action: Redirect
Channel: SIP/linksys2-10201e90
Context: lab
Exten: 500
Priority: 1
ActionID: 010123234545
```

```
Response: Success
ActionID: 010123234545
Message: Redirect successful
```

## SIPpeers

SIPpeers — Lists all SIP peers

Lists the currently configured SIP peers along with their status.

## Parameters

ActionID	[optional] An action identifier that you can use to identify the response to this manager transaction.
----------	--

## Privilege

system, all

## Example

```
Action: SIPPeers
ActionID: 555444333222111
```

```
Response: Success
ActionID: 555444333222111
Message: Peer status list will follow
```

```
Event: PeerEntry
ActionID: 555444333222111
Channeltype: SIP
ObjectName: labrat
ChanObjectType: peer
IPaddress: 10.0.0.75
IPport: 5060
Dynamic: no
Natsupport: no
VideoSupport: no
ACL: no
Status: OK (318 ms)
RealtimeDevice: no
```

```
Event: PeerEntry
ActionID: 555444333222111
Channeltype: SIP
ObjectName: guineapig
ChanObjectType: peer
IPaddress: 172.18.227.72
IPport: 5060
Dynamic: no
Natsupport: no
VideoSupport: no
ACL: no
Status: Unmonitored
RealtimeDevice: no
```

```
Event: PeerEntry
ActionID: 555444333222111
Channeltype: SIP
ObjectName: another
ChanObjectType: peer
IPaddress: 172.18.227.73
IPport: 5060
Dynamic: yes
Natsupport: no
VideoSupport: no
ACL: no
Status: Unmonitored
RealtimeDevice: no
```

```
Event: PeerlistComplete
ListItems: 7
ActionID: 555444333222111
```

## SIPShowPeer

SIPShowPeer — Shows information about a SIP peer

Shows detailed information about a configured SIP peer.

## Parameters

Peer	[required] The name of the SIP peer.
ActionID	[optional] An action identifier that you can use to identify the response to this manager transaction.

## Privilege

system, all

## Example

```
Action: SIPShowPeer
Peer: linksys2
ActionID: 9988776655
```

```
Response: Success
ActionID: 9988776655
Channeltype: SIP
ObjectName: linksys2
ChanObjectType: peer
SecretExist: Y
MD5SecretExist: N
Context: lab
Language:
AMAFlags: Unknown
CID-CallingPres: Presentation Allowed, Not Screened
Callgroup:
Pickupgroup:
VoiceMailbox:
TransferMode: open
LastMsgsSent: -1
Call-limit: 0
MaxCallBR: 384 kbps
Dynamic: Y
Callerid: "Linksys #2" <555>
RegExpire: 2516 seconds
SIP-AuthInsecure: no
SIP-NatSupport: RFC3581
ACL: N
SIP-CanReinvite: Y
SIP-PromiscRedir: N
SIP-UserPhone: N
SIP-VideoSupport: N
SIP-DTMFmode: rfc2833
SIPLastMsg: 0
ToHost:
Address-IP: 192.168.5.71
Address-Port: 5061
Default-addr-IP: 0.0.0.0
Default-addr-port: 5056
```

```
Default-Username: linksys2
RegExtension: 6100
Codecs: 0x4 (ulaw)
CodecOrder: ulaw
Status: Unmonitored
SIP-Useragent: Linksys/SPA962-5.1.5
Reg-Contact : sip:linksys2@192.168.5.71:5061
```

## SetCDRUserField

SetCDRUserField — Sets the CDR UserField

Sets the UserField setting for the CDR record on the specified channel.

### Parameters

Channel	[required] The channel on which to set the CDR UserField.
UserField	[required] The value to assign to the UserField in the CDR record.
ActionID	[optional] An identifier that can be used to identify the response to this action.

### Privilege

call, all

### Example

```
Action: SetCDRUserField
Channel: SIP/test-10225140
UserField: abcdefg
```

```
Response: Success
Message: CDR Userfield Set
```

## SetVar

SetVar — Sets channel variable

Sets a global or channel variable.

### Parameters

Channel	[optional] Channel on which to set the variable. If not set, the variable will be set as a global variable.
---------	---

Variable	[required] Variable name.
Value	[required] Value.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

call, all

## Example

```
Action: SetVar
Channel: SIP/linksys2-10225140
Variable: MyOwnChannelVariable
Value: 42
```

```
Response: Success
Message: Variable Set
```

```
Action: SetVar
Variable: MyOwnGlobalVariable
Value: 25
```

```
Response: Success
Message: Variable Set
```

## Status

Status — Lists channel status

Lists the status of one or more channels, showing details about their current state.

## Parameters

Channel	[optional] Limits the status output to the specified channel.
ActionID	[optional] An action identifier that you can use to identify the response to this manager transaction.

## Privilege

call, all

## Example



**Action: Status****Channel:** SIP/test-10225140**ActionID:** 101010101010101

Response: Success

ActionID: 101010101010101

Message: Channel status will follow

Event: Status

Privilege: Call

Channel: SIP/test-10225140

CallerID: "Bob Jones" &lt;501&gt;

CallerIDNum: 501

CallerIDName: "Bob Jones"

Account:

State: Up

Context: lab

Extension: 201

Priority: 1

Seconds: 865

Link: Local/200@lab-4d13,1

Uniqueid: 1177550165.0

ActionID: 101010101010101

Event: StatusComplete

ActionID: 101010101010101

## StopMonitor

StopMonitor — Stops the recording of a channel

Stops a previously started monitoring (recording) on a channel.

## Parameters

Channel [required] The name of the channel to stop monitoring.

ActionID [optional] A unique identifier to help you identify responses to this command.

## Privilege

call, all

## Example

Action: StopMonitor

Channel: SIP/linksys2-10216e38

Response: Success

Message: Stopped monitoring channel

# UnpauseMonitor

UnpauseMonitor — Unpauses monitoring

Unpauses the monitoring (recording) of the specified channel.

## Parameters

Channel	[required] The name of the channel on which to unpause the monitoring.
ActionID	[optional] A unique identifier to help you identify responses to this command.

## Privilege

call, all

## Example

```
Action: UnpauseMonitor
Channel: SIP/linksys2-10212040
ActionID: 282828282828282

Response: Success
ActionID: 282828282828282
Message: Unpaused monitoring of the channel
```

# UpdateConfig

UpdateConfig — Updates a config file

Dynamically updates an Asterisk configuration file.

## Parameters

SrcFilename	[required] The filename of the configuration file from which to read the current information.
DstFilename	[required] The filename of the configuration file to be written.
Reload	[optional] Specifies whether or not a reload should take place after the configuration update, or the name of a specific module that should be reloaded.
Action-XXXXXX	[required] An action to take. Can be one of NewCat, RenameCat, DelCat, Update, Delete, or Append.

Cat-XXXXXX	[required] The name of the category to operate on.
Var-XXXXXX	[optional] The name of the variable to operate on.
Value-XXXXXX	[optional] The value of the variable to operate on.
Match-XXXXXX	[optional] If set, an extra parameter that must be matched on the line.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

config, all

## Example

```
Action: UpdateConfig
SrcFilename: sip.conf
DstFilename: test.conf
Action-000000: update
Cat-000000: linksys
Var-000000: mailbox
Value-000000: 101@lab
```

Response: Success

## Notes

Note that the first set of parameters should be numbered 000000, the second 000001, and so on. This allows you to update many different configuration values at the same time. It should also be noted that the Asterisk GUI uses this its primary mechanism for updating the configuration of Asterisk.

## UserEvent

UserEvent — Sends an arbitrary event

Sends an arbitrary event to the Asterisk Manager Interface.

## Parameters

UserEvent	[required] The name of the arbitrary event to send.
Header	[optional] The name and value of an arbitrary parameter to your event. You may add as many additional headers (along with their values) to your event.
ActionID	[optional] An identifier which can be used to identify the response to this action.

## Privilege

user, all

## Example

```
Action: UserEvent
Blah: one
SomethingElse: two
ActionID: 63346

Event: UserEvent
Privilege: user,all
UserEvent:
Action: UserEvent
Blah: one
SomethingElse: two
ActionID: 63346
```

## WaitEvent

WaitEvent — Waits for an event to occur

After calling this action, Asterisk will send you a Success response as soon as another event is queued by the Asterisk Manager Interface. Once WaitEvent has been called on an HTTP manager session, events will be generated and queued.

## Parameters

Timeout	[optional] Maximum time to wait for events.
ActionID	[optional] An identifier that can be used to identify the response to this action.

## Privilege

none

## Example

```
Action: WaitEvent
Timeout: 30

Action: Ping

Response: Success
Message: Waiting for Event...

Event: WaitEventComplete

Response: Pong
```

# ZapDNDOff

ZapDNDOff — Sets a Zap channel's do not disturb status to off

Toggles the do not disturb state on the specified Zap channel to off.

## Parameters

<code>ZapChannel</code>	[required] The number of the Zap channel on which to turn off the do not disturb status.
<code>ActionID</code>	[optional] An identifier that can be used to identify the response to this action.

## Privilege

none

## Example

```
Action: ZapDNDOff
ZapChannel: 1
ActionID: 01234567899876543210
```

```
Response: Success
ActionID: 01234567899876543210
Message: DND Disabled
```

# ZapDNDon

ZapDNDon — Sets a Zap channel's do not disturb status to on

Toggles the do not disturb state on the specified Zap channel to on.

## Parameters

<code>ZapChannel</code>	[required] The number of the Zap channel on which to turn on the Do Not Disturb status.
<code>ActionID</code>	[optional] An identifier that can be used to identify the response to this action.

## Privilege

none

## Example

```
Action: ZapDNDOn
ZapChannel: 1
ActionID: 98765432100123456789
```

```
Response: Success
ActionID: 98765432100123456789
Message: DND Enabled
```

## ZapDialOffhook

ZapDialOffhook — Dials over Zap channel while off-hook

Dials the specified number on the Zap channel while the phone is off-hook.

### Parameters

ZapChannel	[required] The Zap channel on which to dial the number.
Number	[required] The number to dial.
ActionID	[optional] A unique identifier to help you identify responses to this command.

### Privilege

none

### Example

```
Action: ZapDialOffhook
ZapChannel: 1
Number: 543215432154321
ActionID: 5676
```

```
Response: Success
ActionID: 5676
Message: ZapDialOffhook
```

## ZapHangup

ZapHangup — Hangs up Zap channel

Hangs up the specified Zap channel.

### Parameters

ZapChannel	[required] The Zap channel to hang up.
------------	--

ActionID [optional] A unique identifier to help you identify responses to this command.

## Privilege

none

## Example

```
Action: ZapHangup
ZapChannel: 1-1
ActionID: 98237892
```

```
Response: Success
ActionID: 98237892
Message: ZapHangup
```

## ZapRestart

ZapRestart — Fully restarts Zaptel channels

Completely restarts the Zaptel channels, terminating any calls in progress.

## Privilege

none

## Example

```
Action: ZapRestart
```

```
Response: Success
Message: ZapRestart: Success
```

## ZapShowChannels

ZapShowChannels — Shows status Zapata channels

Shows the status of all the Zap channels.

## Parameters

ActionID [optional] An action identifier that can be used to identify the response from Asterisk.

## Privilege

none

## Example

```
Action: ZapShowChannels
ActionID: 9999999999
```

```
Response: Success
ActionID: 9999999999
Message: Zapata channel status will follow
```

```
Event: ZapShowChannels
Channel: 1
Signalling: FXO Kewlstart
Context: incoming
DND: Disabled
Alarm: No Alarm
ActionID: 9999999999
```

```
Event: ZapShowChannels
Channel: 4
Signalling: FXS Kewlstart
Context: incoming
DND: Disabled
Alarm: No Alarm
ActionID: 9999999999
```

```
Event: ZapShowChannelsComplete
ActionID: 9999999999
```

## ZapTransfer

ZapTransfer — Transfers Zap channel

Transfers a Zap channel.

## Privilege

none

## Example

```
Action: ZapTransfer
ZapChannel: 1
```



**ActionID: 4242**

Response: Success

Message: ZapTransfer

ActionID: 4242