Asterisk Documentation

Asterisk Development Team <asteriskteam@digium.com>

Home

This is the home of the official wiki for The Asterisk Project.

This is not the first wiki that has existed for Asterisk, but there are some significant things that are different about this wiki than others. The most significant difference is that this wiki was created to be the official source of documentation for the Asterisk project, maintained by the same development team that manages the code itself. That means that we are committed to the content being correct and up to date. To make that happen, editing the content is not open to the general public. However, all Asterisk users are encouraged to participate by leaving comments on pages.

If you are an Asterisk expert and would like to get involved with the development and maintenance of content for the Asterisk wiki, contact Russell Bryant.

Thank you very much for your continued support of Asterisk!

Recently Updated

- Asterisk Developer Call - 06232011 - 1000 EST
  created by Bryan M. Johns
  Jun 23, 2011
- Application_GotoIf
  updated by Wiki Bot
  (view change)
  Jun 18, 2011
- Application_Goto
  updated by Wiki Bot
  (view change)
  Jun 18, 2011
- ManagerAction_Login
  updated by Wiki Bot
  (view change)
  Jun 18, 2011
- Application_MeetMeChannelAdmin
  updated by Wiki Bot
  (view change)
  Jun 18, 2011
- ConfBridge 1.10
  updated by Malcolm Davenport
  (view change)
  Jun 15, 2011
- Creating SIP Accounts
  commented by Tim Osman
  Jun 15, 2011
- Commit Messages
  updated by Russell Bryant
  (view change)
  Jun 14, 2011
- AGICommand_HANGUP
  updated by Wiki Bot
  (view change)
  Jun 14, 2011
- MySQL CEL Backend
  commented by Jose P. Espinal
  Jun 08, 2011
- Asterisk Packages
  updated by Malcolm Davenport

Navigate space

Asterisk 1.8 Documentation

New in 1.8

Overview

A listing of new capabilities in Asterisk 1.8

In Brief

Asterisk 1.8 introduces a number of new features since the previous 1.6.2 release. Highlights include:

- Secure RTP (SRTP)
- IPv6 Support for SIP
- Connected Party Identification Support - COLP and CONP.
- Calendaring Integration for CalDAV, iCal, Exchange or EWS calendars

- A new call logging system, Channel Event Logging (CEL)
- Distributed Device State, including Message Waiting Indicator using Jabber/XMPP PubSub
- Call Completion Supplementary Services (CCSS) Support, including Call Completion on Busy Subscriber (CCBS) and Call Completion on No Response (CCNR)
- Advice of Charge, including AOC-S, AOC-D, and AOC-E Support
- Multicast RTP
- ISDN Q.SIG Call Rerouting and Call Deflection
- Google Talk and Google Voice integration
- Audio Pitch Shifting (for fun and profit)

Detailed Listing

SIP Changes

- Added preferred_codec_only option in sip.conf. This feature limits the joint codecs sent in response to an INVITE to the single most preferred codec.
- Added SIP_CODEC_OUTBOUND dialplan variable which can be used to set the codec to be used for the outgoing call. It must be one of the codecs configured for the device.
- Added tlsprivatekey option to sip.conf. This allows a separate .pem file to be used for holding a private key. If tlsprivatekey is not specified, tlscertfile is searched for both public and private key.
- Added tlsclientmethod option to sip.conf. This allows the protocol for outbound client connections to be specified.
- The sendrpid parameter has been expanded to include the options 'rpid' and 'pai'. Setting sendrpid to 'rpid' will cause Remote-Party-ID header to be sent (equivalent to setting sendrpid=yes) and setting sendrpid to 'pai' will cause P-Asserted-Identity header to be sent.
- The 'ignoresdpversion' behavior has been made automatic when the SDP received is in response to a T.38 re-INVITE that Asterisk initiated. In this situation, since the call will fail if Asterisk does not process the incoming SDP, Asterisk will accept the SDP even if the SDP version number is not properly incremented, but will generate a warning in the log indicating that the SIP peer that sent the SDP should have the 'ignoresdpversion' option set.
- The 'nat' option has now been been changed to have yes, no, force_rport, and comedia as valid values. Setting it to yes forces RFC 3581 behavior and enables symmetric RTP support. Setting it to no only enables RFC 3581 behavior if the remote side requests it and disables symmetric RTP support. Setting it to force_rport forces RFC 3581 behavior and disables symmetric RTP support. Setting it to comedia enables RFC 3581 behavior if the remote side requests it and enables symmetric RTP support.
- Slave SIP channels now set HASH(SIP_CAUSE,<slave-channel-name>) on each response. This permits the master channel to know how each channel dialled in a multi-channel setup resolved in an individual way.
- Added 'externtcpport' and 'externtlsport' options to allow custom port configuration for the externip and externhost options when tcp or tls is used.
- Added support for message body (stored in content variable) to SIP NOTIFY message accessible via AMI and CLI.
- Added 'media_address' configuration option which can be used to explicitly specify the IP address to use in the SDP for media (audio, video, and text) streams.
- Added 'unsolicited_mailbox' configuration option which specifies the virtual mailbox that the new/old count should be stored on if an unsolicited MWI NOTIFY message is received.
- Added 'use_q850_reason' configuration option for generating and parsing if available Reason: Q.850;cause=<cause code> header. It is implemented in some gateways for better passing PRI/SS7 cause codes via SIP.
- When dialing SIP peers, a new component may be added to the end of the dialstring to indicate that a specific remote IP address or host should be used when dialing the particular peer. The dialstring format is SIP/peer/exten/host_or_IP.
- SRTP SDES support for encrypting calls to/from Asterisk over SIP. The ability to selectively force bridged channels to also be encrypted is also implemented. Branching in the dialplan can be done based on whether or not a channel has secure media and/or signaling.
- Added directmediapermit/directmediadeny to limit which peers can send direct media to each other
- Added the 'snom_aoc_enabled' option to turn on support for sending Advice of Charge messages to snom phones.
- Added support for G.719 media streams.
- Added support for 16khz signed linear media streams.
- SIP is now able to bind to and communicate with IPv6 addresses. In addition, RTP has been outfitted with the same abilities.
- Added support for setting the Max-Forwards: header in SIP requests. Setting is available in device configurations as well as in the dial plan.

- Addition of the 'subscribe_network_change' option for turning on and off res_stun_monitor module support in chan_sip.
- Addition of the 'auth_options_requests' option for turning on and off authentication for OPTIONS requests in chan_sip.

IAX2 Changes

- Added rtsavesysname option into iax.conf to allow the systname to be saved on realtime updates.
- Added the ability for chan_iax2 to inform the dialplan whether or not encryption is being used. This interoperates with the SIP SRTP implementation so that a secure SIP call can be bridged to a secure IAX call when the dialplan requires bridged channels to be "secure".
- Addition of the 'subscribe_network_change' option for turning on and off res_stun_monitor module support in chan_iax.

MGCP Changes

- Added ability to preset channel variables on indicated lines with the setvar configuration option. Also, clearvars=all resets the list of variables back to none.
- PacketCable NCS 1.0 support has been added for Docsis/Eurodocsis Networks. See configs/res_pktccops.conf for more information.

XMPP Google Talk/Jingle changes

- Added the externip option to gtalk.conf.
- Added the stunaddr option to gtalk.conf which allows for the automatic retrieval of the external ip from a stun server.

Applications

- Added 'p' option to PickupChan() to allow for picking up channel by the first match to a partial channel name.
- Added .m3u support for Mp3Player application.
- Added progress option to the app_dial D() option. When progress DTMF is present, those values are sent immediately upon receiving a PROGRESS message regardless if the call has been answered or not.
- Added functionality to the app_dial F() option to continue with execution at the current location when no parameters are provided.
- Added the 'a' option to app_dial to answer the calling channel before any announcements or macros are executed.
- Modified app_dial to set answertime when the called channel answers even if the called channel hangs up during playback of an announcement.
- Modified app_dial 'r' option to support an additional parameter to play an indication tone from indications.conf
- Added c() option to app_chanspy. This option allows custom DTMF to be set to cycle through the next available channel. By default this is still '*'.
- Added x() option to app_chanspy. This option allows DTMF to be set to exit the application.
- The Voicemail application has been improved to automatically ignore messages that only contain silence.
- If you set maxmsg to 0 in voicemail.conf, Voicemail will consider the associated mailbox(es) to be greetings-only.
- The ChanSpy application now has the 'S' option, which makes the application automatically exit once it hits a point where no more channels are available to spy on.
- The ChanSpy application also now has the 'E' option, which spies on a single channel and exits when that channel hangs up.
- The MeetMe application now turns on the DENOISE() function by default, for each participant. In our tests, this has significantly decreased background noise (especially noisy data centers).
- Voicemail now permits storage of secrets in a separate file, located in the spool directory of each individual user. The control for this is located in the "passwordlocation" option in voicemail.conf. Please see the sample configuration for more information.
- The ChanIsAvail application now exposes the returned cause code using a separate variable, AVAILCAUSECODE, instead of overwriting the device state in AVAILSTATUS.
- Added 'd' option to app_followme. This option disables the "Please hold"

announcement.
- Added 'y' option to app_record. This option enables a mode where any DTMF digit received will terminate recording.
- Voicemail now supports per mailbox settings for folders when using IMAP storage. Previously the folder could only be set per context, but has now been extended using the imapfolder option.
- Voicemail now supports per mailbox settings for nextaftercmd and minsecs.
- Voicemail now allows the pager date format to be specified separately from the email date format.
- New applications JabberJoin, JabberLeave, and JabberSendGroup have been added to allow joining, leaving, and sending text to group chats.
- MeetMe has a new option 'G' to play an announcement before joining a conference.
- Page has a new option 'A(x)' which will playback an announcement simultaneously to all paged phones (and optionally excluding the caller's one using the new option 'n') before the call is bridged.
- The 'f' option to Dial has been augmented to take an optional argument. If no argument is provided, the 'f' option works as it always has. If an argument is provided, then the connected party information of all outgoing channels created during the Dial will be set to the argument passed to the 'f' option.
- Dial now inherits the GOSUB_RETVAL from the peer, when the U() option runs a Gosub on the peer.
- The OSP lookup application adds in/outbound network ID, optional security, number portability, QoS reporting, destination IP port, custom info and service type features.
- Added new application VMSayName that will play the recorded name of the voicemail user if it exists, otherwise will play the mailbox number.
- Added custom device states to ConfBridge bridges. Use 'confbridge:<name>' to retrieve state for a particular bridge, where <name> is the conference name
- app_directory now allows exiting at any time using the operator or pound key.
- Voicemail now supports setting a locale per-mailbox.
- Two new applications are provided for declining counting phrases in multiple languages. See the application notes for SayCountedNoun and SayCountedAdj for more information.
- Voicemail now runs the externnotify script when pollmailboxes is activated and notices a change.
- Voicemail now includes rdnis within msgXXXX.txt file.
- Added 'D' command to ExternalIVR full details in http://wiki.asterisk.org

Dialplan Functions

- SRVQUERY and SRVRESULT functions added. This can be used to query and iterate over SRV records associated with a specific service. From the CLI, type 'core show function SRVQUERY' and 'core show function SRVRESULT' for more details on how these may be used.
- PITCH_SHIFT dialplan function added. This function can be used to modify the pitch of a channel's tx and rx audio streams.
- Added new dialplan functions CONNECTEDLINE and REDIRECTING which permits setting various connected line and redirecting party information.
- CALLERID and CONNECTEDLINE dialplan functions have been extended to support ISDN subaddressing.
- The CHANNEL() function now supports the "name" and "checkhangup" options.
- For DAHDI channels, the CHANNEL() dialplan function now allows the dialplan to request changes in the configuration of the active echo canceller on the channel (if any), for the current call only. The syntax is:

exten => s,n,Set(CHANNEL(echocan_mode)=off)

The possible values are:

on - normal mode (the echo canceller is actually reinitialized)
off - disabled
fax - FAX/data mode (NLP disabled if possible, otherwise completely disabled)
voice - voice mode (returns from FAX mode, reverting the changes that were made when FAX mode was requested)

- Added new dialplan function MASTER_CHANNEL(), which permits retrieving

and setting variables on the channel which created the current channel.
Administrators should take care to avoid naming conflicts, when multiple
channels are dialled at once, especially when used with the Local channel
construct (which all could set variables on the master channel). Usage
of the HASH() dialplan function, with the key set to the name of the slave
channel, is one approach that will avoid conflicts.

- Added new dialplan function MUTEAUDIO() for muting inbound and/or outbound
  audio in a channel.
- func_odbc now allows multiple row results to be retrieved without using
  mode=multirow. If rowlimit is set, then additional rows may be retrieved
  from the same query by using the name of the function which retrieved the
  first row as an argument to ODBC_FETCH().
- Added JABBER_RECEIVE, which permits receiving XMPP messages from the
  dialplan. This function returns the content of the received message.
- Added REPLACE, which searches a given variable name for a set of characters,
  then either replaces them with a single character or deletes them.
- Added PASSTHRU, which literally passes the same argument back as its return
  value. The intent is to be able to use a literal string argument to
  functions that currently require a variable name as an argument.
- HASH-associated variables now can be inherited across channel creation, by
  prefixing the name of the hash at assignment with the appropriate number of
  underscores, just like variables.
- GROUP_MATCH_COUNT has been improved to allow regex matching on category
- CHANNEL(secure_bridge_signaling) and CHANNEL(secure_bridge_media) to set/get
  whether or not channels that are bridged to the current channel will be
  required to have secure signaling and/or media.
- CHANNEL(secure_signaling) and CHANNEL(secure_media) to get whether or not
  the current channel has secure signaling and/or media.
- For DAHDI/ISDN channels, the CHANNEL() dialplan function now supports the
  "no_media_path" option.
  Returns "0" if there is a B channel associated with the call.
  Returns "1" if no B channel is associated with the call. The call is either
  on hold or is a call waiting call.
- Added option to dialplan function CDR(), the 'f' option
  allows for high resolution times for billsec and duration fields.
- FILE() now supports line-mode and writing.
- Added FIELDNUM(), which returns the 1-based offset of a field in a list.
- FRAME_TRACE(), for tracking internal ast_frames on a channel.

Dialplan Variables

- Added DYNAMIC_FEATURENAME which holds the last triggered dynamic feature.
- Added DYNAMIC_PEERNAME which holds the unique channel name on the other side
  and is set when a dynamic feature is triggered.
- Added PARKINGLOT which can be used with parkeddynamic feature.conf option
  to dynamically create a new parking lot matching the value this varible is
  set to.
- Added PARKINGDYNAMIC which represents the template parkinglot defined in
  features.conf that should be the base for dynamic parkinglots.
- Added PARKINGDYNCONTEXT which tells what context a newly created dynamic
  parkinglot should have.
- Added PARKINGDYNPOS which holds what parking positions a dynamic parkinglot
  should have.

Queue changes

- Added "ready" option to QUEUE_MEMBER counting to count free agents whose wrap-up
  timeout has expired.
- Added 'R' option to app_queue. This option stops moh and indicates ringing
  to the caller when an Agent's phone is ringing. This can be used to indicate
  to the caller that their call is about to be picked up, which is nice when
  one has been on hold for an extened period of time.
- A new config option, penaltymemberslimit, has been added to queues.conf.
  When set this option will disregard penalty settings when a queue has too
  few members.
- A new option, 'I' has been added to both app_queue and app_dial.
  By setting this option, Asterisk will not update the caller with
  connected line changes or redirecting party changes when they occur.
- A 'relative-peroidic-announce' option has been added to queues.conf. When
  enabled, this option will cause periodic announce times to be calculated
  from the end of announcements rather than from the beginning.
- The autopause option in queues.conf can be passed a new value, "all." The

result is that if a member becomes auto-paused, he will be paused in all queues for which he is a member, not just the queue that failed to reach the member.

- Added dialplan function QUEUE_EXISTS to check if a queue exists
- The queue logger now allows events to optionally propagate to a file, even when realtime logging is turned on. Additionally, realtime logging supports sending the event arguments to 5 individual fields, although it will fallback to the previous data definition, if the new table layout is not found.

mISDN channel driver (chan_misdn) changes

- Added display_connected parameter to misdn.conf to put a display string in the CONNECT message containing the connected name and/or number if the presentation setting permits it.
- Added display_setup parameter to misdn.conf to put a display string in the SETUP message containing the caller name and/or number if the presentation setting permits it.
- Made misdn.conf parameters localdialplan and cpndialplan take a -1 to indicate the dialplan settings are to be obtained from the asterisk channel.
- Made misdn.conf parameter callerid accept the "name" <number> format used by the rest of the system.
- Made use the nationalprefix and internationalprefix misdn.conf parameters to prefix any received number from the ISDN link if that number has the corresponding Type-Of-Number. NOTE: This includes comparing the incoming call's dialed number against the MSN list.
- Added the following new parameters: unknownprefix, netspecificprefix, subscriberprefix, and abbreviatedprefix in misdn.conf to prefix any received number from the ISDN link if that number has the corresponding Type-Of-Number.
- Added new dialplan application misdn_command which permits controlling the CCBS/CCNR functionality.
- Added new dialplan function mISDN_CC which permits retrieval of various values from an active call completion record.
- For PTP, you should manually send the COLR of the redirected-to party for an incomming redirected call if the incoming call could experience further redirects. Just set the REDIRECTING(to-num,i) = ${EXTEN} and set the REDIRECTING(to-pres) to the COLR. A call has been redirected if the REDIRECTING(from-num) is not empty.
- For outgoing PTP redirected calls, you now need to use the inhibit(i) option on all of the REDIRECTING statements before dialing the redirected-to party. You still have to set the REDIRECTING(to-xxx,i) and the REDIRECTING(from-xxx,i) values. The PTP call will update the redirecting-to presentation (COLR) when it becomes available.
- Added outgoing_colp parameter to misdn.conf to filter outgoing COLP information.

thirdparty mISDN enhancements

mISDN has been modified by Digium, Inc. to greatly expand facility message support to allow:

- Enhanced COLP support for call diversion and transfer.
- CCBS/CCNR support.

The latest modified mISDN v1.1.x based version is available at:
http://svn.digium.com/svn/thirdparty/mISDN/trunk
http://svn.digium.com/svn/thirdparty/mISDNuser/trunk

Tagged versions of the modified mISDN code are available under:
http://svn.digium.com/svn/thirdparty/mISDN/tags
http://svn.digium.com/svn/thirdparty/mISDNuser/tags

libpri channel driver (chan_dahdi) DAHDI changes

- The channel variable PRIREDIRECTREASON is now just a status variable and it is also deprecated. Use the REDIRECTING(reason) dialplan function to read and alter the reason.
- For Q.SIG and ETSI PRI/BRI-PTP, you should manually send the COLR of the redirected-to party for an incomming redirected call if the incoming call could experience further redirects. Just set the REDIRECTING(to-num,i) = CALLERID(dnid) and set the REDIRECTING(to-pres) to the COLR. A call has been redirected if the REDIRECTING(count) is not zero.
- For outgoing Q.SIG and ETSI PRI/BRI-PTP redirected calls, you need to use the inhibit(i) option on all of the REDIRECTING statements before dialing the redirected-to party. You still have to set the REDIRECTING(to-xxx,i) and the REDIRECTING(from-xxx,i) values. The call will update the redirecting-to presentation (COLR) when it becomes available.
- Added the ability to ignore calls that are not in a Multiple Subscriber Number (MSN) list for PTMP CPE interfaces.
- Added dynamic range compression support for dahdi channels. It is configured via the rxdrc and txdrc parameters in chan_dahdi.conf.
- Added support for ISDN calling and called subaddress with partial support for connected line subaddress.
- Added support for BRI PTMP NT mode. (Requires latest LibPRI.)
- Added handling of received HOLD/RETRIEVE messages and the optional ability to transfer a held call on disconnect similar to an analog phone.
- Added CallRerouting/CallDeflection support for Q.SIG, ETSI PTP, ETSI PTMP. Will reroute/deflect an outgoing call when receive the message. Can use the DAHDISendCallreroutingFacility to send the message for the supported switches.
- Added standard location to add options to chan_dahdi dialing: Dial(DAHDI/g1[/extension[/options]]) Current options: K(<keypad_digits>) R Reverse charging indication
- Added Reverse Charging Indication (Collect calls) send/receive option. Send reverse charging in SETUP message with the chan_dahdi R dialing option. Dial(DAHDI/g1/extension/R) Access received reverse charge in SETUP message by: ${CHANNEL(reversecharge)} (requires latest LibPRI)
- Added ability to send/receive keypad digits in the SETUP message. Send keypad digits in SETUP message with the chan_dahdi K(<keypad_digits>) dialing option. Dial(DAHDI/g1/[~mdavenport:extension]/K(<keypad_digits>)) Access any received keypad digits in SETUP message by: ${CHANNEL(keypad_digits)} (requires latest LibPRI)
- Added ability to send and receive ETSI Explicit Call Transfer (ECT) messages to eliminate tromboned calls. A tromboned call goes out an interface and comes back into the same interface. Tromboned calls happen because of call routing, call deflection, call forwarding, and call transfer.
- Added the ability to send and receive ETSI Advice-Of-Charge messages.
- Added the ability to support call waiting calls. (The SETUP has no B channel assigned.)
- Added Malicious Call ID (MCID) event to the AMI call event class.
- Added Message Waiting Indication (MWI) support for ISDN PTMP endpoints (phones).

Asterisk Manager Interface

- The Hangup action now accepts a Cause header which may be used to set the channel's hangup cause.
- sslprivatekey option added to manager.conf and http.conf. Adds the ability to specify a separate .pem file to hold a private key. By default sslcert is used to hold both the public and private key.
- Options in manager.conf and http.conf with the 'ssl' prefix have been replaced for options containing the 'tls' prefix. For example, 'sslenable' is now 'tlsenable'. This has been done in effort to keep ssl and tls options consistent across all .conf files. All affected sample.conf files have been modified to reflect this change. Previous options such as 'sslenable' still work, but options with the 'tls' prefix are preferred.
- Added a MuteAudio AMI action for muting inbound and/or outbound audio in a channel. (res_mutestream.so)
- The configuration file manager.conf now supports a channelvars option, which specifies a list of channel variables to include in each channel-oriented event.
- The redirect command now has new parameters ExtraContext, ExtraExtension, and ExtraPriority to allow redirecting the second channel to a different location than the first.

- Added new event "JabberStatus" in the Jabber module to monitor buddies status.
- Added a "MixMonitorMute" AMI action for muting inbound and/or outbound audio in a MixMonitor recording.
- The 'iax2 show peers' output is now similar to the expected output of 'sip show peers'.
- Added Advice-Of-Charge events (AOC-S, AOC-D, and AOC-E) in the new aoc event class.
- Added Advice-Of-Charge manager action, AOCMessage, for generating AOC-D and AOC-E messages on a channel.
- A DBGetComplete event now follows a DBGetResponse, to make the DBGet action conform more closely to similar events.
- Added a new eventfilter option per user to allow whitelisting and blacklisting of events.
- Added optional parkinglot variable for park command.

Channel Event Logging

- A new interface, CEL, is introduced here. CEL logs single events, much like the AMI, but it differs from the AMI in that it logs to db backends much like CDR does; is based on the event subsystem introduced by Russell, and can share in all its benefits; allows multiple backends to operate like CDR; is specialized to event data that would be of concern to billing sytems, like CDR. Backends for logging and accounting calls have been produced, but a new CDR backend is still in development.

CDR

- 'linkedid' and 'peeraccount' are new CDR fields available to CDR aficionados. linkedid is based on uniqueID, but spreads to other channels as transfers, dials, etc are performed. Thus the pieces of CDR can be grouped into multilegged sets.
- Multiple files and formats can now be specified in cdr_custom.conf.
- cdr_syslog has been added which allows CDRs to be written directly to syslog. See configs/cdr_syslog.conf.sample for more information.
- A 'sequence' field has been added to CDRs which can be combined with linkedid or uniqueid to uniquely identify a CDR.
- Handling of billsec and duration field has changed. If your table definition specifies those fields as float,double or similar they will now be logged with microsecond accuracy instead of a whole integer.

Calendaring for Asterisk

- A new set of modules were added supporing calendar integration with Asterisk. Dialplan functions for reading from and writing to calendars are included, as well as the ability to execute dialplan logic upon calendar event notifications. iCalendar, CalDAV, and Exchange Server calendars (via res_calendar_exchange for Exchange Server 2003 with no write or attendee support, and res_calendar_ews for Exchange Server 2007+ with full write and attendee support) are supported (Exchange 2003 support does not support forms-based authentication).

Call Completion Supplementary Services for Asterisk

- Call completion support has been added for SIP, DAHDI/ISDN, and DAHDI/analog. DAHDI/ISDN supports call completion for the following switch types: EuroIsdn(ETSI) for PTP and PTMP modes, and Qsig. See http://wiki.asterisk.org for details.

Multicast RTP Support

- A new RTP engine and channel driver have been added which supports Multicast RTP. The channel driver can be used with the Page application to perform multicast RTP paging. The dial string format is: MulticastRTP/<type>/<destination>/<control address> Type can be either basic or linksys. Destination is the IP address and port for the RTP packets. Control address is specific to the linksys type and is used for sending the control packets unique to them.

Security Events Framework

- Asterisk has a new C API for reporting security events. The module res_security_log
  sends these events to the "security" logger level. Currently, AMI is the only
  Asterisk component that reports security events. However, SIP support will be
  coming soon. For more information on the security events framework, see the
  "Security Events" chapter of the included documentation - doc/AST.pdf.

Fax

- A technology independent fax frontend (res_fax) has been added to Asterisk.
- A spandsp based fax backend (res_fax_spandsp) has been added.
- The app_fax module has been deprecated in favor of the res_fax module and
  the new res_fax_spandsp backend.
- The SendFAX and ReceiveFAX applications now send their log messages to a
  'fax' logger level, instead of to the generic logger levels. To see these
  messages, the system's logger.conf file will need to direct the 'fax' logger
  level to one or more destinations; the logger.conf.sample file includes an
  example of how to do this. Note that if the 'fax' logger level is **not**
  directed to at least one destination, log messages generated by these
  applications will be lost, and that if the 'fax' logger level is directed to
  the console, the 'core set verbose' and 'core set debug' CLI commands will
  have no effect on whether the messages appear on the console or not.

Miscellaneous

- The transmit_silence_during_record option in asterisk.conf.sample has been removed.
  Now, in order to enable transmitting silence during record the transmit_silence
  option should be used. transmit_silence_during_record remains a valid option, but
  defaults to the behavior of the transmit_silence option.
- Addition of the Unit Test Framework API for managing registration and execution
  of unit tests with the purpose of verifying the operation of C functions.
- SendText is now implemented in chan_gtalk and chan_jingle. It will simply send
  XMPP text messages to the remote JID.
- Modules.conf has a new option - "require" - that marks a module as critical for
  the execution of Asterisk.
  If one of the required modules fail to load, Asterisk will exit with a return
  code set to 2.
- An 'X' option has been added to the asterisk application which enables #exec support.
  This allows #exec to be used in asterisk.conf.
- jabber.conf supports a new option auth_policy that toggles auto user registration.
- A new lockconfdir option has been added to asterisk.conf to protect the
  configuration directory (/etc/asterisk by default) during reloads.
- The parkeddynamic option has been added to features.conf to enable the creation
  of dynamic parkinglots.
- chan_dahdi now supports reporting alarms over AMI either by channel or span via
  the reportalarms config option.
- chan_dahdi supports dialing configuring and dialing by device file name.
  DAHDI/span-name!local!1 will use /dev/dahdi/span-name/local/1 . Likewise
  it may appear in chan_dahdi.conf as 'channel => span-name!local!1'.
- A new options for chan_dahdi.conf: 'ignore_failed_channels'. Boolean.
  False by default. If set, chan_dahdi will ignore failed 'channel' entries.
  Handy for the above name-based syntax as it does not depend on
  initialization order.
- The Realtime dialplan switch now caches entries for 1 second. This provides a
  significant increase in performance (about 3X) for installations using this switchtype.
- Distributed devicestate now supports the use of the XMPP protocol, in addition to
  AIS. For more information, please see http://wiki.asterisk.org
- The addition of G.719 pass-through support.
- Added support for 16khz Speex audio. This can be enabled by using 'allow=speex16'
  during device configuration.
- The UNISTIM channel driver (chan_unistim) has been updated to support devices that
  have less than 3 lines on the LCD.
- Realtime now supports database failover. See the sample extconfig.conf for details.
- The addition of improved translation path building for wideband codecs. Sample
  rate changes during translation are now avoided unless absolutely necessary.
- The addition of the res_stun_monitor module for monitoring and reacting to network
  changes while behind a NAT.

CLI Changes

- The 'core set debug' and 'core set verbose' commands, in previous versions, could optionally accept a filename, to apply the setting only to the code generated from that source file when Asterisk was built. However, there are some modules in Asterisk that are composed of multiple source files, so this did not result in the behavior that users expected. In this version, 'core set debug' and 'core set verbose' can optionally accept **module** names instead (with or without the .so extension), which applies the setting to the entire module specified, regardless of which source files it was built from.
- New 'manager show settings' command showing the current settings loaded from manager.conf.
- Added 'all' keyword to the CLI command "channel request hangup" so that you can send the channel hangup request to all channels.
- Added a "core reload" CLI command that executes a global reload of Asterisk.

Getting Started

A Beginners Guide to Asterisk. Herein, you will find content related to installing Asterisk and basic usage concepts.

Precursors, Background and Business

Discovering Asterisk

This section of the documentation attempts to explain at a high level what Asterisk is and does. It also attempts to provide primers on the key technical disciplines that are required to successfully create and manage Asterisk solutions. Much of the material in this section is optional and may be redundant for those with a background in communications application development. For the other 99.9875% of the population, this is good stuff. Read on...

Asterisk Concepts

Asterisk is a very large application that does many things. It can be somewhat difficult to understand, especially if you are new to communications technologies. In the next few chapters we will do our best to explain what Asterisk is, what it is not, and how it came to be this way. This section doesn't cover the technology so much as the concept. If you're already familiar with the function of a telephony engine, feel free to jump ahead to the next section.

Asterisk as a Swiss Army Knife of Telephony

What Is Asterisk?

People often tend to think of Asterisk as an "open source PBX" because that was the focus of the original development effort. But calling Asterisk a PBX is both selling it short (it is much more) and overstating it (it can be much less). It is true that Asterisk started out as a phone system for a small business (see the "Brief History" section for the juicy details) but in the decade since it was originally released it has grown into a universal tool for building communications applications. Today Asterisk powers not only IP PBX systems but also VoIP gateways, call center systems, conference bridges, voicemail servers and all kinds of other applications that involve real-time communications.

Asterisk is not a PBX but is the engine that powers PBXs. Asterisk is not an IVR but is the engine that powers IVRs. Asterisk is not a call center ACD but is the engine that powers ACD/queueing systems.

Asterisk is to communications applications what the Apache web server is to web applications. Apache is a web server. Asterisk is a communication server. Apache handles all the low-level details of sending and receiving data using the HTTP protocol. Asterisk handles all the low level details of sending and receiving data using lots of different communication protocols. When you install Apache, you have a web server but its up to you to create the web applications. When you install Asterisk, you have a communications server but its up to you to create the communications applications.

Web applications are built out of HTML pages, CSS style sheets, server-side processing scripts, images, databases, web services, etc. Asterisk communications applications are built out Dialplan scripts, configuration files, audio recordings, databases, web services, etc. For a web application to work, you need the web server connected to the Internet. For a communications application to work, you need the communications server connected to communication services (VoIP or PSTN). For people to be able to access your web site you need to register a domain name and set up DNS entries that point "www.yourdomain.com" to your server. For people to access your communications system you need phone numbers or VoIP URIs that send calls to your server.

In both cases the server is the plumbing that makes your application work. The server handles the low-level complexities and allows you, the application developer, to concentrate on the application logic and presentation. You don't have to be an expert on HTTP to create powerful web applications, and you don't have to be an expert on SIP or Q.931 to create powerful communications applications.

Here's a simple example. The following HTML script, installed on a working web server, prints "Hello World" in large type:

```
<html>
  <head>
    <title>Hello World Demo</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

The following Dialplan script answers the phone, waits for one second, plays back "hello world" then hangs up.

```
exten => 100,1,Answer()
exten => 100,n,Wait(1)
exten => 100,n,Playback(hello-world)
exten => 100,n,Hangup()
```

In both cases the server components are handling all of the low level details of the underlying protocols. Your application doesn't have to worry about the byte alignment, the packet size, the codec or any of the thousands of other critical details that make the application work. This is the

power of an engine.

Who Uses Asterisk?

Asterisk is created by communication system developers, for communication system developers. As an open source project, Asterisk is a collaboration between many different individuals and companies, all of which need a flexible communications engine to power their applications.

A Brief History of the Asterisk Project

Way, way back in 1999 a young man named Mark Spencer was finishing his Computer Engineering degree at Auburn University when he hit on an interesting business concept. 1999 was the high point in the .com revolution (aka bubble), and thousands of businesses world-wide were discovering that they could save money by using the open source Linux operating system in place of proprietary operating systems. The lure of a free operating system with open access to the source code was too much to pass up. Unfortunately there was little in the way of commercial support available for Linux at that time. Mark decided to fill this gap by creating a company called "Linux Support Services". LSS offered a support hotline that IT professionals could (for a fee) call to get help with Linux.

The idea took off. Within a few months, Mark had a small office staffed with Linux experts. Within a few more months the growth of the business expanded demanded a "real" phone system that could distribute calls evenly across the support team, so Mark called up several local phone system vendors and asked for quotes. Much to his surprise, the responses all came back well above $50,000 -- far more than Mark had budgeted for the project. Far more than LSS could afford.

Rather than give in and take out a small business loan, Mark made a fateful decision. He decided to write his own phone system. Why not? A phone system is really just a computer running phone software, right? Fortunately for us, Mark had no idea how big a project he had take on. If he had known what a massive undertaking it was to build a phone system from the ground up might have gritted his teeth, borrowed the money and spent the next decade doing Linux support. But he didn't know what he didn't know, and so he started to code. And he coded. And he coded.

Mark had done his engineering co-op at Adtran, a communications and networking device manufacturer in Huntsville, AL. There he had cut his teeth on telecommunications system development, solving difficult problems generating a prodigious amount of complex code in short time. This experience proved invaluable as he began to frame out the system which grew into Asterisk. In only a few months Mark crafted the original Asterisk core code. As soon as he had a working prototype he published the source code on the Internet, making it available under the GPL license (the same license used for Linux).

Within a few months the idea of an "open source PBX" caught on. There had been a few other open source communications projects, but none had captured the imagination of the global population of communications geeks like Asterisk. As Mark labored on the core system, hundreds (now thousands) of developers from all over the world began to submit new features and functions.

Beginning Asterisk

Installing Asterisk

Now that you know a bit about Asterisk and how it is used, it's time to get you up and running with your own Asterisk installation. There are various ways to get started with Asterisk on your own system:

- Install an Asterisk-based Linux distribution such as AsteriskNOW. This takes care of installing Linux, Asterisk, and some web-based interfaces all at the same time, and is the easiest way to get started if you're new to Linux and/or Asterisk.
- If you're already familiar with Linux or Unix, you can simply install packages for Asterisk and its related tools using the package manager in your operating system. We'll cover this in more detail below in Section 200.3. Alternate Install Methods.
- For the utmost in control of your installation, you can compile and install Asterisk (and its related tools) from source code. We'll explain how to do this in Section 201. Installing Asterisk From Source.

Installing AsteriskNOW

Installing AsteriskNOW is easy! Simply visit http://www.asterisknow.org/ and download the latest version. The file you'll download will have a .iso file extension. Then burn the .iso image to a CD, and boot your system from the CD.

🔴 Installing AsteriskNOW Will Overwrite Data

Please be aware that installing AsteriskNOW will overwrite any existing data on your hard drive. Anything that is important should first be backed up to a different system.

When you boot from the AsteriskNOW CD, you'll see an introductory screen. Simply press enter to continue the installation.

Screen shot of ISOLINUX screen

Install-Time Options

As the installer continues, you'll be prompted to enter several pieces of information.

- Hard disk layout. It is recommended to select "Remove all partitions on selected drives and create default layout." and move to the next screen. This will erase all data on the system.
- Timezone settings. Select the location that is nearest to you and move to the next screen.
- Root password. The root user is the administrative user on Linux systems. Most system configuration requires root access. If this password is lost, it is difficult to recover. It is recommended that your password contain a mix of lowercase and UPPERCASE letters, numbers, and/or symbols.

After the final option, installation will begin. This will take approximately 15-30 minutes. Once installation has completed, the system will reboot into your AsteriskNOW installation.

By default, AsteriskNOW will use DHCP to obtain an IP address on your network. You can use the ifconfig command under Linux to view your current IP address, or system-config-network to change your network settings.

Alternate Install Methods

If you already have a Linux system that you can dedicate to Asterisk, simply use the package manager in your operating system to install Asterisk, DAHDI, and libpri. Most modern Linux distributions such as Debian, Ubuntu, and Fedora have these packages in their repositories.

Packages for Red Hat Enterprise Linux and CentOS are also available at http://packages.asterisk.org/ (see Asterisk Packages for instructions on use).

Validating Your AsteriskNOW Installation

Before continuing on, let's check a few things to make sure your system is in good working order. First, let's make sure the DAHDI drivers are loaded. After logging in as the **root** user you can use the **lsmod** under Linux to list all of the loaded kernel modules, and the **grep** command to filter the input and only show the modules that have dahdi in their name.

```
[root@server asterisk-1.6.X.Y]# lsmod | grep dahdi
```

If the command returns nothing, then DAHDI has not been started. Start DAHDI by running:

```
[root@server asterisk-1.6.X.Y]# service dadhi start
```

If you have DAHDI running, the output of lsmod | grep dahdi should look something like the output below. (The exact details may be different, depending on which DAHDI modules have been built, and so forth.)

```
[root@server ~]# lsmod | grep dahdi

dahdi_dummy           4288    0

dahdi_transcode       7928    1 wctc4xxp

dahdi_voicebus        40464   2 wctdm24xxp,wcte12xp

dahdi                 196544 12
dahdi_dummy,wctdm24xxp,wcte11xp,wct1xxp,wcte12xp,wct4xxp

crc_ccitt             2096    1 dahdi
```

Now that DAHDI is running, you can run **dahdi_hardware** to list any DAHDI-compatible devices in your system. You can also run the **dahdi_tool** utility to show the various DAHDI-compatible devices, and their current state.

To check if Asterisk is running, you can use the Asterisk initscript.

```
[root@server ~]# service asterisk status
asterisk is stopped
```

To start Asterisk, we'll use the initscript again, this time giving it the start action:

```
[root@server ~]# service asterisk start
Starting asterisk:
```

When Asterisk starts, it runs as a background service (or daemon), so you typically won't see any response on the command line. We can check the status of Asterisk and see that it's running by using the command below. (The process identifier, or pid, will obviously be different on your system.)

```
[root@server ~]# service asterisk status
asterisk (pid 32117) is running...
```

And there you have it... you have an Asterisk system up and running! You should now continue on in Section 202. Getting Started with Asterisk.

Asterisk Configuration Files

Intro to Asterisk Configuration Files

In this section, we'll introduce you to the Asterisk configuration files, and show you how to use some advanced features.

Config File Format

Asterisk is a very flexible telephony engine. With this flexibility, however, comes a bit of complexity. Asterisk has quite a few configuration files which control almost every aspect of how it operates. The format of these configuration files, however, is quite simple. The Asterisk configuration files are plain text files, and can be edited with any text editor.

Sections and Settings

The configuration files are broken into various section, with the section name surrounded by square brackets. Section names should not contain spaces, and are case sensitive. Inside of each section, you can assign values to various settings. In general, settings in one section are independent of values in another section. Some settings take values such as true or false, while other settings have more specific settings. The syntax for assigning a value to a setting is to write the setting name, an equals sign, and the value, like this:

```
[section-name]
setting=true

[another_section]
setting=false
setting2=true
```

Objects

Some Asterisk configuration files also create objects. The syntax for objects is slightly different than for settings. To create an object, you specify the type of object, an arrow formed by the equals sign and a greater-than sign (=>), and the settings for that object.

```
[section-name]
some_object => settings
```

✓ **Confused by Object Syntax?**
In order to make life easier for newcomers to the Asterisk configuration files, the developers have made it so that you can also create objects with an equal sign. Thus, the two lines below are functionally equivalent.

```
some_object => settings
some_object=settings
```

It is common to see both versions of the syntax, especially in online Asterisk documentation and examples. This book, however, will denote objects by using the arrow instead of the equals sign.

```
[section-name]
label1=value1
label2=value2
object1 => name1

label1=value0
label3=value3
object2 => name2
```

In this example, **object1** inherits both **label1** and **label2**. It is important to note that **object2** also inherits **label2**, along with **label1** (with the new overridden value **value0**) and **label3**.

In short, objects inherit all the settings defined above them in the current section, and later settings override earlier settings.

Comments

We can (and often do) add comments to the Asterisk configuration files. Comments help make the configuration files easier to read, and can also be used to temporarily disable certain settings.

Comments on a Single Line

Single-line comments begin with the semicolon (;) character. The Asterisk configuration parser treats everything following the semicolon as a comment. To expand on our previous example:

```
[section-name]
setting=true

[another_section]
setting=false ; this is a comment
; this entire line is a comment
;awesome=true
; the semicolon on the line above makes it a
; comment, disabling the setting
```

Block Comments

Asterisk also allows us to create block comments. A block comment is a comment that begins on one line, and continues for several lines. Block comments begin with the character sequence

```
;--
```

and continue across multiple lines until the character sequence

```
--;
```

is encountered. The block comment ends immediately after --; is encountered.

```
[section-name]
setting=true
;-- this is a block comment that begins on this line
and continues across multiple lines, until we
get to here --;
```

Using The include and exec Constructs

There are two other constructs we can use within our configuration files. They are **#include** and **#exec**.

The **#include** construct tells Asterisk to read in the contents of another configuration file, and act as though the contents were at this location in this configuration file. The syntax is **#include filename**, where **filename** is the name of the file you'd like to include. This construct is most often used to break a large configuration file into smaller pieces, so that it's more manageable.

The **#exec** takes this one step further. It allows you to execute an external program, and place the output of that program into the current configuration file. The syntax is **#exec program**, where **program** is the name of the program you'd like to execute.

Let's look at example of both constructs in action.

```
[section-name]
setting=true
#include otherconfig.conf      ; include another configuration file
#exec otherprogram             ; include output of otherprogram
```

Adding to an existing section

If you want to add settings to an existing section of a configuration file (either later in the file, or when using the **#include** and **#exec** constructs), add a plus sign in parentheses after the section heading, as shown below:

```
[section-name]
setting1=value1

[section-name](+)
setting2=value2
```

This example shows that the **setting2** setting was added to the existing section of the configuration file.

Templates

Another construct we can use within most Asterisk configuration files is the use of templates. A template is a section of a configuration file that is only used as a base (or template, as the name suggests) to create other sections from.

Template Syntax

To define a section as a template, place an exclamation mark in parentheses after the section heading, as shown in the example below.

```
[template-name](!)
setting=value
```

Using Templates

To use a template when creating another section, simply put the template name in parentheses after the section heading name, as shown in the example below. If you want to inherit from multiple templates, use commas to separate the template names).

```
[template-name](!)
setting=value

[template-2](!)
setting2=value2

[section-name](template-name,template-2)
setting3=value3
```

The newly-created section will inherit all the values and objects defined in the template(s), as well as any new settings or objects defined in the newly-created section. The settings and objects defined in the newly-created section override settings or objects of the same name from the templates. Consider this example:

```
[test-one](!)
permit=192.168.0.2
host=alpha.example.com
deny=192.168.0.1

[test-two](!)
permit=192.168.1.2
host=bravo.example.com
deny=192.168.1.1

[test-three](test-one,test-two)
permit=192.168.3.1
host=charlie.example.com
```

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

```
[test-three]
permit=192.168.0.2
host=alpha.example.com
deny=192.168.0.1
permit=192.168.1.2
host=bravo.example.com
deny=192.168.1.1
permit=192.168.3.1
host=charlie.example.com
```

Basic PBX Functionality

In this section, we're going to guide you through the basic setup of a very primitive PBX. After you finish, you'll have a basic PBX with two phones that can dial each other. In later modules, we'll go into more detail on each of these steps, but in the meantime, this will give you a basic system on which you can learn and experiement.

The Most Basic PBX

While it won't be anything to brag about, this basic PBX that you will build from Asterisk will help you learn the fundamentals of configuring Asterisk. For this exercise, we're going to assume that you have access to two phones which speak the SIP voice-over-IP protocol. There are a wide variety of SIP phones available in many different shapes and sizes, and if your budget doesn't allow for you to buy phones, feel free to use a free soft phone. Soft phones are simply computer programs which run on your computer and emulate a real phone, and communicate with other devices across your network, just like a real voice-over-IP phone would.

Creating SIP Accounts

In order for our two phones to communicate with each other, we need to configure an account for each phone in the channel driver which corresponds to the protocol they'll be using. Since both the phones are using the SIP protocol, we'll configure accounts in the SIP channel driver configuration file, called **sip.conf**. (This file resides in the Asterisk configuration directory, which is typically **/etc/asterisk**.) Let's name your phones **Alice** and **Bob**, so that we can easily differentiate between them.

Open **sip.conf** with your favorite text editor, and spend a minute or two looking at the file. (Don't let it overwhelm you — the sample **sip.conf** has a **lot** of data in it, and can be overwhelming at first glance.) Notice that there are a couple of sections at the top of the configuration, such as [general] and [authentication], which control the overall functionality of the channel driver. Below those sections, there are sections which correspond to SIP accounts on the system. Scroll to the bottom of the file, and add a section for Alice and Bob. You'll need to choose your own unique password for each account, and change the **permit** line to match the settings for your local network.

```
[demo-alice]
type=friend
host=dynamic
secret=verysecretpassword ; put a strong, unique password here
instead
context=users
deny=0.0.0.0/0
permit=192.168.5.0/255.255.255.0 ; replace with your network
settings

[demo-bob]
type=friend
host=dynamic
secret=othersecretpassword ; put a strong, unique password here
instead
context=users
deny=0.0.0.0/0
permit=192.168.5.0/255.255.255.0 ; replace with your network
settings
```

> **Be Serious About Account Security**
>
> We can't stress enough how important it is for you to pick a strong password for all accounts on Asterisk, and to only allow access from trusted networks. Unfortunately, we've found many instances of people exposing their Asterisk to the internet at large with easily-guessable passwords, or no passwords at all. You could be at risk of toll fraud, scams, and other malicious behavior.
>
> For more information on Asterisk security and how you can protect yourself, check out http://www.asterisk.org/security/webinar/.

After adding the two sections above to your **sip.conf** file, go to the Asterisk command-line interface and run the **sip reload** command to tell Asterisk to re-read the **sip.conf** configuration file.

```
server*CLI> sip reload

Reloading SIP

server*CLI>
```

> ℹ️ **Reloading Configuration Files**
>
> Don't forget to reload the appropriate Asterisk configuration files after you have made changes to them.

Registering Phones to Asterisk

The next step is to configure the phones themselves to communicate with Asterisk. The way we have configured the accounts in the SIP channel driver, Asterisk will expect the phones to **register** to it. Registration is simply a mechanism where a phone communicates "Hey, I'm Bob's phone... here's my username and password. Oh, and if you get any calls for me, I'm at this particular IP address."

Configuring your particular phone is obviously beyond the scope of this guide, but here are a list of common settings you're going to want to set in your phone, so that it can communicate with Asterisk:

- **Registrar/Registration Server** - The location of the server which the phone should register to. This should be set to the IP address of your Asterisk system.
- *SIP User Name/Account Name/Address - *The SIP username on the remote system. This should be set to demo-alice on one phone and demo-bob on the other. This username corresponds directly to the section name in square brackets in sip.conf.
- **SIP Authentication User/Auth User** - On Asterisk-based systems, this will be the same as the SIP user name above.
- **Proxy Server/Outbound Proxy Server** - This is the server with which your phone communicates to make outside calls. This should be set to the IP address of your Asterisk system.

You can tell whether or not your phone has registered successfully to Asterisk by checking the output of the **sip show peers** command at the Asterisk CLI. If the **Host** column says **(Unspecified)**, the phone has not yet registered. On the other hand, if the **Host** column contains an IP address and the **Dyn** column contains the letter **D**, you know that the phone has successfully registered.

```
server*CLI> sip show peers
Name/username           Host            Dyn NAT ACL Port
Status
demo-alice              (Unspecified)   D       A   5060
Unmonitored
demo-bob                192.168.5.105   D       A   5060
Unmonitored
2 sip peers [Monitored: 0 online, 0 offline Unmonitored: 2 online, 0
offline]
```

In the example above, you can see that Alice's phone has not registered, but Bob's phone has registered.

Creating Dialplan Extensions

The last things we need to do to enable Alice and Bob to call each other is to configure a couple of extensions in the dialplan.

ℹ️ What is an Extension?

When dealing with Asterisk, the term extension does not represent a physical device such as a phone. An extension is simply a set of actions in the dialplan which may or may not write a physical device. In addition to writing a phone, an extensions might be used for such things auto-attendant menus and conference bridges. In this guide we will be careful to use the words phone or device when referring to the physical phone, and extension when referencing the set of instructions in the Asterisk dialplan.

Let's take a quick look at the dialplan, and then add two extensions.

Open **extensions.conf**, and take a quick look at the file. Near the top of the file, you'll see some general-purpose sections named [general] and [globals]. Any sections in the dialplan beneath those two sections is known as a **context**. The sample **extensions.conf** file has a number of other contexts, with names like [demo] and [default].

We'll cover contexts more in Dialplan Fundamentals, but for now you should know that each phone or outside connection in Asterisk points at a single context. If the dialed extension does not exist in the specified context, Asterisk will reject the call.

Go to the bottom of your **extensions.conf** file, and add a new context named [users].

Naming Your Dialplan Contexts

There's nothing special about the name **users** for this context. It could have been named **strawberry_milkshake**, and it would have behaved exactly the same way. It is considered best practice, however, to name your contexts for the types of extensions that are contained in that context. Since this context contains extensions for the users of our PBX system, we'll call our context **users**.

Underneath that context name, we'll create an extesion numbered **6001** which attempts to ring Alice's phone for twenty seconds, and an extension **6002** which attempts to rings Bob's phone for twenty seconds.

```
[users]
exten=>6001,1,Dial(SIP/demo-alice,20)
exten=>6002,1,Dial(SIP/demo-bob,20)
```

After adding that section to **extensions.conf**, go to the Asterisk command-line interface and tell Asterisk to reload the dialplan by typing the command **dialplan reload**. You can verify that Asterisk successfully read the configuration file by typing **dialplan show users** at the CLI.

```
server*CLI> dialplan show users
[ Context 'users' created by 'pbx_config' ]
  '6001' =>          1. Dial(SIP/demo-alice,20)
[pbx_config]
  '6002' =>          1. Dial(SIP/demo-bob,20)
[pbx_config]

-= 2 extensions (2 priorities) in 1 context. =-
```

Now we're ready to make a test call!

Making a Phone Call

At this point, you should be able to pick up Alice's phone and dial extension **6002** to call Bob, and dial **6001** from Bob's phone to call Alice. As you make a few test calls, be sure to watch the Asterisk command-line interface (and ensure that your verbosity is set to a value three or higher) so that you can see the messages coming from Asterisk, which should be similar to the ones below:

```
server*CLI>       -- Executing [6002@users:1]
Dial("SIP/demo-alice-00000000", "SIP/demo-bob,20") in new stack
      -- Called demo-bob
      -- SIP/demo-bob-00000001 is ringing
      -- SIP/demo-bob-00000001 answered SIP/demo-alice-00000000
      -- Native bridging SIP/demo-alice-00000000 and
SIP/demo-bob-00000001
   == Spawn extension (users, 6002, 1) exited non-zero on
'SIP/demo-alice-00000000'
```

As you can see, Alice called extension **6002** in the [users] context, which in turn used the **Dial** application to call Bob's phone. Bob's phone rang, and then answered the call. Asterisk then bridged the two calls (one call from Alice to Asterisk, and the other from Asterisk to Bob), until Alice hung up the phone.

At this point, you have a very basic PBX. It has two extensions which can dial each other, but that's all. Before we move on, however, let's review a few basic troubleshooting steps that will help you be more successful as you learn about Asterisk.

> ✓ Basic PBX Troubleshooting
>
> The most important troubleshooting step is to set your verbosity level to three (or higher), and watch the command-line interface for errors or warnings as calls are placed.
>
> To ensure that your SIP phones are registered, type **sip show peers** at the Asterisk CLI.
>
> To see which context your SIP phones will send calls to, type **sip show users**.
>
> To ensure that you've created the extensions correctly in the **[users]** context in the dialplan, type **dialplan show users**.
>
> To see which extension will be executed when you dial extension **6002**, type **dialplan show 6002@users**.

Sound Prompt Searching based on Channel Language

Dialplan Fundamentals

The dialplan is essential to the operation of any successful Asterisk system. In this module, we'll help you learn the fundamental components of the Asterisk dialplan, and how to combine them to begin scripting your own dialplan. We'll also add voice mail and a dial-by-name directory features to your dialplan.

Contexts, Extensions, and Priorities

The dialplan is organized into various sections, called contexts. Contexts are the basic organizational unit within the dialplan, and as such, they keep different sections of the dialplan independent from each other. We'll use contexts to enforce security boundaries between the various parts of our dialplan, as well as to provide different classes of service to groups of users.

The syntax for a context is exactly the same as any other section heading in the configuration files, as explained in Section 206.2.1. Sections and Settings. Simply place the context name in square brackets. For example, here is the context we defined in the previous module:

```
[users]
```

Within each context, we can define one or more **extensions**. As explained in the previous module, an extension is simply a named set of actions. Asterisk will perform each action, in sequence, when that extension number is dialed. The syntax for an extension is:

```
exten => number,priority,application([parameter[,parameter2...]])
```

As an example, let's review extension **6001** from the previous module. It looks like:

```
exten => 6001,1,Dial(SIP/demo-alice,20)
```

In this case, the extension number is **6001**, the priority number is **1**, the application is **Dial()**, and the two parameters to the application are **SIP/demo-alice** and **20**.
Within each extension, there must be one or more *priorities*. A priority is simply a sequence number. The first priority on an extension is executed first. When it finishes, the second priority is executed, and so forth.

> ✅ Priority numbers
> Priority numbers must begin with 1, and must increment sequentially. If Asterisk can't find the next priority number, it will terminate the call. We call this *auto-fallthrough*. Consider the example below:
>
> ```
> exten => 6123,1,do something
> exten => 6123,2,do something else
> exten => 6123,4,do something different
> ```
>
> In this case, Asterisk would execute priorites one and two, but would then terminate the call, because it couldn't find priority number three.

Priority number can also be simplied by using the letter **n** in place of the priority numbers greater than one. The letter **n** stands for **next**, and when Asterisk sees priority n it replaces it in memory with the previous priority number plus one. Note that you must still explicitly declare priority number one.

```
exten => 6123,1,do something
exten => 6123,n,do something else
exten => 6123,n,do something different
```

You can also assign a label (or alias) to a particular priority number by placing the label in parentheses directly after the priority number, as shown below. Labels make it easier to jump back to a particular location within the extension at a later time.

```
exten => 6123,1,do something
exten => 6123,n(repeat),do something else
exten => 6123,n,do something different
```

Here, we've assigned a label named **repeat** to the second priority.

Included in the Asterisk 1.6.2 branch (and later) there is a way to avoid having to repeat the extension name/number or pattern using the **same =>** prefix.

```
exten => _1NXXNXXXXXX,1,do something
same => n(repeat),do something else
same => n,do something different
```

Applications

Each priority in the dialplan calls an application. An application does some work on the channel, such as answering a call or playing back a sound prompt. There are a wide variety of dialplan applications available for your use. For a complete list of the dialplan applications available to your installation of Asterisk, type **core show applications** at the Asterisk CLI.

Most applications take one or more parameters, which provide additional information to the application or change its behavior. Parameters should be separated by commas.

> ✅ Syntax for Parameters
> You'll often find examples of Asterisk dialplan code online and in print which use the pipe character or vertical bar character (|) between parameters, as shown in this example:
>
> ```
> exten => 6123,1,application(one|two|three)
> ```
>
> This is a deprecated syntax, and will no longer work in newer versions of Asterisk. Simply replace the pipe character with a comma, like this:
>
> ```
> exten => 6123,1,application(one,two,three)
> ```

Answer, Playback, and Hangup Applications

As its name suggests, the **Answer()** application answers an incoming call. The **Answer()** application takes a delay (in milliseconds) as its first parameter. Adding a short delay is often useful for ensuring that the remote endpoing has time to begin processing audio before you play a sound prompt. Otherwise, you may not hear the very beginning of the prompt.

Knowing When to Answer a Call

When you're first learning your way around the Asterisk dialplan, it may be a bit confusing knowing when to use the **Answer()** application, and when not to.

If Asterisk is simply going to pass the call off to another device using the **Dial()** application, you probably don't want to call the answer the call first. If, on the other hand, you want Asterisk to play sound prompts or gather input from the caller, it's probably a good idea to call the **Answer()** application before doing anything else.

The **Playback()** application loads a sound prompt from disk and plays it to the caller, ignoring any touch tone input from the caller. The first parameter to the dialplan application is the filename of the sound prompt you wish to play, without a file extension. If the channel has not already been answered, **Playback()** will answer the call before playing back the sound prompt, unless you pass **noanswer** as the second parameter.

To avoid the first few milliseconds of a prompt from being cut off you can play a second of silence. For example, if the prompt you wanted to play was hello-world which would look like this in the dialplan:

```
exten => 1234,1,Playback(hello-world)
```

You could avoid the first few seconds of the prompt from being cut off by playing the silence/1 file:

```
exten => 1234,1,Playback(silence/1)
exten => 1234,n,Playback(hello-world)
```

Alternatively this could all be done on the same line by separating the filenames with an ampersand (&):

```
exten => 1234,1,Playback(silence/1&hello-world)
```

Exploring Sound Prompts

Asterisk comes with a wide variety of pre-recorded sound prompts. When you install Asterisk, you can choose to install both core and extra sound packages in several different file formats. Prompts are also available in several languages. To explore the sound files on your system, simply find the sounds directory (this will be **/var/lib/asterisk/sounds** on most systems) and look at the filenames. You'll find useful prompts ("Please enter the extension of the person you are looking for..."), as well as as a number of off-the-wall prompts (such as "Weasels have eaten our

phone system", "The office has been overrun with iguanas", and "Try to spend your time on hold not thinking about a blue-eyed polar bear") as well.

> ✅ **Sound Prompt Formats**
> Sound prompts come in a variety of file formats, such as **.wav** and **.ulaw** files. When asked to play a sound prompt from disk, Asterisk plays the sound prompt with the file format that can most easily be converted to the CODEC of the current call. For example, if the inbound call is using the **alaw** CODEC and the sound prompt is available in **.gsm** and **.ulaw** format, Asterisk will play the **.ulaw** file because it requires fewer CPU cycles to transcode to the **alaw** CODEC.
> You can type the command **core show translation** at the Asterisk CLI to see the transcoding times for various CODECs. The times reported (in Asterisk 1.6.0 and later releases) are the number of microseconds it takes Asterisk to transcode one second worth of audio. These times are calculated when Asterisk loads the codec modules, and often vary slightly from machine to machine. To perform a current calculation of translation times, you can type the command **core show translation recalc 60**.

How Asterisk Searches for Sound Prompts Based on Channel Language

Each channel in Asterisk can be assigned a language by the channel driver. The channel's language code is split, piece by piece (separated by underscores), and used to build paths to look for sound prompts. Asterisk then uses the first file that is found.

This means that if we set the language to en_GB_female_BT, for example, Asterisk would search for files in:

.../sounds/en/GB/female/BT

.../sounds/en/GB/female

.../sounds/en/GB

.../sounds/en

.../sounds

This scheme makes it easy to add new sound prompts for various language variants, while falling back to a more general prompt if there is no prompt recorded in the more specific variant.

The **Hangup()** application hangs up the current call. While not strictly necessary due to auto-fallthrough (see the note on Priority numbers above), in general we recommend you add the **Hangup()** application as the last priority in any extension.

Now let's put **Answer()**, **Playback()**, and **Hangup()** together to play a sample sound file. Place this extension in your **[docs:users]** context:

```
exten => 6000,1,Answer(500)
exten => 6000,n,Playback(hello-world)
exten => 6000,n,Hangup()
```

Dial Application

Now that you've learned the basics of using dialplan applications, let's take a closer look at the **Dial()** application that we used earlier in extensions **6001** and **6002**. **Dial()** attempts to ring an external device, and if the call is answered it bridges the two channels together and does any necessary protocol or CODEC conversion. It also handles call progress responses (busy, no-answer, ringing).

> ✅ **Dial() and the Dialplan**
> Please note that if the Dial() application successfully bridges two channels together, that the call does not progress in the dialplan. The call will only continue on to the next priority if the Dial() application is unable to bridge the calling channel to the dialed device.

The **Dial()** application takes four parameters:

1. **Devices**
   - A list of the device(s) you want to call. Devices are specified as technology or channel driver, a forward slash, and the device or account name. For example, **SIP/demo-alice** would use the SIP channel driver to call the device specified in the **demo-alice** section of **sip.conf**. Devices using the IAX2 channel driver take the form of **IAX2/demo-george**, and DAHDI channels take the form of **DAHDI/1**.
   - When calling through a device (such as a gateway) or service provider to reach another number, the syntax is **technology/device/number** such as **SIP/my_provider/5551212** or **DAHDI/4/5551212**.
   - To dial multiple devices at once, simply concatenate the devices together whith the ampersand character (**&**). The first device to answer will get bridged with the caller, and the other endpoints will stop ringing.

     ```
     exten => 6003,1,Dial(SIP/demo-alice&SIP/demo-bob,30)
     ```

2. **Timeout**
   - The number of seconds to allow the device(s) to ring before giving up and moving on to the next priority in the extension.
3. **Options**
   - There are dozens of options that you can set on the outbound call, including call screening, distinctive ringing and more. Type **core show application dial** at the Asterisk CLI for a complete list of all available options. If you want to specify multiple options, simply concatenate them together. For example, if you want to use both the *m*and **H** options, you would set **mH** as the options parameter.
4. **URL**
   - The fourth parameter is a URL that will be sent to the endpoint. Few endpoints do anything with the URL, but there are a few (softphones mostly) that do act on the URL.

Adding Voice Mail to Dialplan Extensions

Adding voicemail to the extensions is quite simple. The Asterisk voicemail module provides two key applications for dealing with voice mail. The first, named **VoiceMail()**, allows a caller to leave a voice mail message in the specified mailbox. The second, called **VoiceMailMain()**, allows the mailbox owner to retrieve their messages and change their greetings.

VoiceMail Application

The **VoiceMail()** applications takes two parameters:

1. **Mailbox**
   - This parameter specifies the mailbox in which the voice mail message should be left. It should be a mailbox number and a voice mail context concatenated with an at-sign (**@**), like **6001@default**. (Voice mail boxes are divided out into various voice mail context, similar to the way that extensions are broken up into dialplan contexts.) If the voice mail context is omitted, it will default to the **default** voice mail context.
2. **Options**
   - One or more options for controlling the mailbox greetings. The most popular options include the u option to play the unavailable message, the **b** option to play the busy message, and the **s** option to skip the system-generated instructions.

VoiceMailMain Application

The **VoiceMailMain()** application allows the owner of a voice mail box to retrieve their messages, as well as set mailbox options such as greetings and their PIN number. The **VoiceMailMain()** application takes two parameters:

1. **Mailbox** - This parameter specifies the mailbox to log into. It should be a mailbox number and a voice mail context, concatenated with an at-sign (@), like 6001@default. If the voice mail context is omitted, it will default to the default voice mail context. If the mailbox number is omitted, the system will prompt the caller for the mailbox number.
2. **Options** - One or more options for controlling the voicemail system. The most popular option is the s option, which skips asking for the PIN number

> ⛔ **Direct Access to Voice mail**
> Please exercise extreme caution when using the s option! With this option set, anyone which has access to this extension can retrieve voicemail messages without entering the mailbox passcode.

Configuring Voice Mail Boxes

Now that we've covered the two main voice mail applications, let's look at the voicemail configuration. Voice mail options and mailboxes are configured in the **voicemail.conf** configuration file. This file has three major sections:

The [general] section

Near the top of **voicemail.conf**, you'll find the **[general]** section. This section of the configuration file controls the general aspects of the voicemail system, such as the maximum number of messages per mailbox, the maximum length of a voicemail message, and so forth. Feel free to look at the sample **voicemail.conf** file for more details about the various settings.

The [zonemessages] section

The **[zonemessages]** section is used to define various timezones around the world. Each mailbox can be assigned to a particular time zone, so that times and dates are announced relative to their local time. The time zones specified in this section also control the way in which times and dates are announced, such as reading the time of day in 24-hour format.

Voice Mail Contexts

After the **[general]** and **[zonemessages]** sections, any other bracketed section is a voice mail context. Within each context, you can define one or more mailbox. To define a mailbox, we set a mailbox number, a PIN, the mailbox owner's name, the primary email address, a secondary email address, and a list of mailbox options (separated by the pipe character), as shown below:

```
mailbox=>pin,full name,email address,short email address,mailbox
options
```

By way of explanation, the short email address is an email address that will receive shorter email notifications suitable for mobile devices such as cell phones and pagers. It will never receive attachments.

To add voice mail capabilities to extensions **6001** and **6002**, add these three lines to the bottom of **voicemail.conf**.

```
[vm-demo]
6001 => 8762,Alice
Jones,alice@example.com,alice2@example.com,attach=no|tz=central|maxmsg
=> 9271,Bob
Smith,bob@example.com,bob2@example.com,attach=yes|tz=eastern
```

Now that we've defined the mailboxes, we can go into the Asterisk CLI and type **voicemail reload** to get Asterisk to reload the **voicemail.conf** file. We can also verify that the new mailboxes have been created by typing **voicemail show users**.

```
server*CLI> voicemail reload
Reloading voicemail configuration...
server*CLI> voicemail show users
Context     Mbox  User                          Zone      NewMsg
default     general New User                              0
default     1234  Example Mailbox                         0
other       1234  Company2 User                           0
vm-demo     6001  Alice Jones               central       0
vm-demo     6002  Bob Smith                 eastern       0
5 voicemail users configured.
```

Now that we have mailboxes defined, let's add a priority to extensions **6001** and **6002** which will allow callers to leave voice mail in their respective mailboxes. We'll also add an extension **6500** to allow Alice and Bob to check their voicemail messages. Please modify your **[users]** context in **extensions.conf** to look like the following:

```
[users]
exten => 6000,1,Answer(500)
exten => 6000,n,Playback(hello-world)
exten => 6000,n,Hangup()

exten => 6001,1,Dial(SIP/demo-alice,20)
exten => 6001,n,VoiceMail(6001@vm-demo,u)

exten => 6002,1,Dial(SIP/demo-bob,20)
exten => 6002,n,VoiceMail(6002@vm-demo,u)

exten => 6500,1,Answer(500)
exten => 6500,n,VoiceMailMain(@vm-demo)
```

Reload the dialplan by typing **dialplan reload** at the Asterisk CLI. You can then test the voice mail system by dialing from one phone to the other and waiting twenty seconds. You should then

be connected to the voicemail system, where you can leave a message. You should also be able to dial extension **6500** to retrieve the voicemail message. When prompted, enter the mailbox number and PIN number of the mailbox.

While in the **VoiceMainMain()** application, you can also record the mailbox owner's name, unavailable greeting, and busy greeting by pressing 0 at the voicemail menu. Please record at least the name greeting for both Alice and Bob before continuing on to the next section.

Go into lots of detail about the voicemail interface? How to move between messages, move between folders, forward messages, etc?

Directory Application

The next application we'll cover is named **Directory()**, because it presents the callers with a dial-by-name directory. It asks the caller to enter the first few digits of the person's name, and then attempts to find matching names in the specified voice mail context in **voicemail.conf**. If the matching mailboxes have a recorded name greeting, Asterisk will play that greeting. Otherwise, Asterisk will spell out the person's name letter by letter.

```
Directory([voicemail_context,[dialplan_context,[options]]])
```

The **Directory()** application takes three parameters:

voicemail_context

This is the context within **voicemail.conf** in which to search for a matching directory entry. If not specified , the **[docs:default]** context will be searched.

dialplan_context

When the caller finds the directory entry they are looking for, Asterisk will dial the extension matching their mailbox in this context.

options

A set of options for controlling the dial-by-name directory. Common options include **f** for searching based on first name instead of last name and **e** to read the extension number as well as the name.

> ✅ **Directory() Options**
> To see the complete list of options for the Directory() application, type **core show application Directory** at the Asterisk CLI.

Let's add a dial-by-name directory to our dialplan. Simply add this line to your **[docs:users]** context in **extensions.conf**:

```
exten => 6501,1,Directory(vm-demo,users,ef)
```

Now you should be able to dial extension **6501** to test your dial-by-name directory.

Auto-attendant and IVR Menus

In this section, we'll cover the how to build voice menus, often referred to as auto-attedants and IVR menus. IVR stands for *Interactive Voice Response*, and is used to describe a system where a caller navigates through a system by using the touch-tone keys on their phone keypad.

When the caller presses a key on their phone keypad, the phone emits two tones, known as DTMF tones. DTMF stands for *Dual Tone Multi-Frequency*. Asterisk recognizes the DTMF tones and responds accordingly. For more information on DTMF tones, see [Section 440.3. DTMF Dialing].

Let's dive in and learn how to build IVR menus in the Asterisk dialplan!

Background and WaitExten Applications

The **Background()** application plays a sound prompt, but listens for DTMF input. Asterisk then tries to find an extension in the current dialplan context that matches the DTMF input. If it finds a matching extension, Asterisk will send the call to that extension.

The Background() application takes the name of the sound prompt as the first parameter just like the Playback() application, so remember not to include the file extension.

> ✅ **Multiple Prompts**
> If you have multiple prompts you'd like to play during the Background() application, simply concatenate them together with the ampersand (&) character, like this:
>
> ```
> exten => 6123,1,Background(prompt1&prompt2&prompt3)
> ```

One problems you may encounter with the **Background()** application is that you may want Asterisk to wait a few more seconds after playing the sound prompt. In order to do this, you can call the **WaitExten()** application. You'll usually see the **WaitExten()** application called immediately after the **Background()** application. The first parameter to the **WaitExten()** application is the number of seconds to wait for the caller to enter an extension. If you don't supply the first parameter, Asterisk will use the built-in response timeout (which can be modified with the **TIMEOUT()** dialplan function).

```
[auto_attendant]
exten => start,1,Verbose(2,Incoming call from ${CALLERID(all)})
    same => n,Playback(silence/1)
    same => n,Background(prompt1&prompt2&prompt3)
    same => n,WaitExten(10)
    same => n,Goto(timeout-handler,1)

exten => timeout-handler,1)
    same => n,Dial(${GLOBAL(OPERATOR)},30)
    same => n,Voicemail(operator@default,${IF($[${DIALSTATUS} =
BUSY]?b:u)})
    same => n,Hangup()
```

Goto Application and Priority Labels

Before we create a simple auto-attendant menu, let's cover a couple of other useful dialplan applications. The **Goto()** application allows us to jump from one position in the dialplan to another. The parameters to the **Goto()** application are slightly more complicated than with the other applications we've looked at so far, but don't let that scare you off.

The **Goto()** application can be called with either one, two, or three parameters. If you call the **Goto()** application with a single parameter, Asterisk will jump to the specified priority (or its label) within the current extension. If you specify two parameters, Asterisk will read the first as an extension within the current context to jump to, and the second parameter as the priority (or label) within that extension. If you pass three parameters to the application, Asterisk will assume they are the context, extension, and priority (respectively) to jump to.

```
[StartingContext]
exten => 100,1,Goto(monkeys)
    same => n,NoOp(We skip this)
    same => n(monkeys),Playback(tt-monkeys)
    same => n,Hangup()

exten => 200,1,Goto(start,1)  ; play tt-weasels then tt-monkeys

exten => 300,1,Goto(start,monkeys) ; only play tt-monkeys

exten => 400,1,Goto(JumpingContext,start,1)  ; play hello-world

exten => start,1,NoOp()
    same => n,Playback(tt-weasels)
    same => n(monkeys),Playback(tt-monkeys)

[JumpingContext]
exten => start,1,NoOp()
    same => n,Playback(hello-world)
    same => n,Hangup()
```

SayDigits, SayNumber, SayAlpha, and SayPhonetic Applications

While not exactly related to auto-attendant menus, we'll introduce some applications to read back various pieces of information back to the caller. The **SayDigits()** and **SayNumber()** applications read the specified number back to caller. To use the **SayDigits()** and **SayNumber()** application simply pass it the number you'd like it to say as the first parameter.

The **SayDigits()** application reads the specified number one digit at a time. For example, if you called **SayDigits(123)**, Asterisk would read back "one two three". On the other hand, the **SayNumber()** application reads back the number as if it were a whole number. For example, if you called **SayNumber(123)** Asterisk would read back "one hundred twenty three".

The **SayAlpha()** and **SayPhonetic()** applications are used to spell an alphanumeric string back to the caller. The **SayAlpha()** reads the specified string one letter at a time. For example, **SayAlpha(hello)** would read spell the word "hello" one letter at a time. The **SayPhonetic()** spells back a string one letter at a time, using the international phonetic alphabet. For example, **SayPhonetic(hello)** would read back "Hotel Echo Lima Lima Oscar".

We'll use these four applications to read back various data to the caller througout this guide. In the meantime, please feel free to add some sample extensions to your dialplan to try out these applications. Here are some examples:

```
exten => 6592,1,SayDigits(123)
exten => 6593,1,SayNumber(123)
exten => 6594,1,SayAlpha(hello)
exten => 6595,1,SayPhonetic(hello)
```

Creating a Simple IVR Menu

Let's go ahead and apply what we've learned about the various dialplan applications by building a very simple auto-attendant menu. It is common practice to create an auto-attendant or IVR menu in a new context, so that it remains independant of the other extensions in the dialplan. Please add the following to your dialplan (the **extensions.conf** file) to create a new **demo-menu** context. In this new context, we'll create a simple menu that prompts you to enter one or two, and then it will read back what you're entered.

> ℹ️ **Sample Sound Prompts**
> Please note that the example below (and many of the other examples in this guide) use sound prompts that are part of the *extra* sounds packages. If you didn't install the extra sounds earlier, now might be a good time to do that.

```
[demo-menu]
exten => s,1,Answer(500)
    same => n(loop),Background(press-1&or&press-2)
    same => n,WaitExten()

exten => 1,1,Playback(you-entered)
    same => n,SayNumber(1)
    same => n,Goto(s,loop)

exten => 2,1,Playback(you-entered)
    same => n,SayNumber(2)
    same => n,Goto(s,loop)
```

Before we can use the demo menu above, we need to add an extension to the **[docs:users]** context to redirect the caller to our menu. Add this line to the **[docs:users]** context in your dialplan:

```
exten => 6598,1,Goto(demo-menu,s,1)
```

Reload your dialplan, and then try dialing extension **6598** to test your auto-attendant menu.

Handling Special Extensions

We have the basics of an auto-attendant created, but now let's make it a bit more robust. We need to be able to handle special situations, such as when the caller enters an invalid extension, or doesn't enter an extension at all. Asterisk has a set of special extensions for dealing with

situations like there. They all are named with a single letter, so we recommend you don't create any other extensions named with a single letter. The most common special extensions include:

i: the invalid entry extension

If Asterisk can't find an extension in the current context that matches the digits dialed during the **Background()** or **WaitExten()** applications, it will send the call to the **i** extension. You can then handle the call however you see fit.

t: the reponse timeout extension

When the caller waits too long before entering a response to the **Background()** or **WaitExten()** applications, and there are no more priorities in the current extension, the call is sent to the t extension.

s: the start extension

When an analog call comes into Asterisk, the call is sent to the **s** extension. The s extension is also used in macros.

Please note that the **s** extension is **not** a catch-all extension. It's simply the location that analog calls and macros begin. In our example above, it simply makes a convenient extension to use that can't be easily dialed from the **Background()** and **WaitExten()** applications.

h: the hangup extension

When a call is hung up, Asterisk executes the **h** extension in the current context. This is typically used for some sort of clean-up after a call has been completed.

o: the operator extension

If a caller presses the zero key on their phone keypad while recording a voice mail message, and the **o** extension exists, the caller will be redirected to the o extension. This is typically used so that the caller can press zero to reach an operator.

a: the assistant extension

This extension is similar to the **o** extension, only it gets triggered when the caller presses the asterisk (*) key while recording a voice mail message. This is typically used to reach an assistant.

Let's add a few more lines to our **[docs:demo-menu]** context, to handle invalid entries and timeouts. Modify your **[docs:demo-menu]** context so that it matches the one below:

```
[demo-menu]
exten => s,1,Answer(500)
    same => n(loop),Background(press-1&or&press-2)
    same => n,WaitExten()

exten => 1,1,Playback(you-entered)
    same => n,SayNumber(1)
    same => n,Goto(s,loop)

exten => 2,1,Playback(you-entered)
    same => n,SayNumber(2)
    same => n,Goto(s,loop)

exten => i,1,Playback(option-is-invalid)
    same => n,Goto(s,loop)

exten => t,1,Playback(are-you-still-there)
    same => n,Goto(s,loop)
```

Now dial your auto-attendant menu again (by dialing extension **6598**), and try entering an invalid option (such as **3**) at the auto-attendant menu. If you watch the Asterisk command-line interface while you dial and your verbosity level is three or higher, you should see something similar to the following:

```
-- Executing [6598@users:1] Goto("SIP/demo-alice-00000008",
"demo-menu,s,1") in new stack
-- Goto (demo-menu,s,1)
-- Executing [s@demo-menu:1] Answer("SIP/demo-alice-00000008",
"500") in new stack
-- Executing [s@demo-menu:2] BackGround("SIP/demo-alice-00000008",
"press-1&or&press-2") in new stack
-- <SIP/demo-alice-00000008> Playing 'press-1.gsm' (language 'en')
-- <SIP/demo-alice-00000008> Playing 'or.gsm' (language 'en')
-- <SIP/demo-alice-00000008> Playing 'press-2.gsm' (language 'en')
-- Invalid extension '3' in context 'demo-menu' on
SIP/demo-alice-00000008
-- Executing [i@demo-menu:1] Playback("SIP/demo-alice-00000008",
"option-is-invalid") in new stack
-- <SIP/demo-alice-00000008> Playing 'option-is-invalid.gsm'
(language 'en')
-- Executing [i@demo-menu:2] Goto("SIP/demo-alice-00000008",
"s,loop") in new stack
-- Goto (demo-menu,s,2)
-- Executing [s@demo-menu:2] BackGround("SIP/demo-alice-00000008",
"press-1&or&press-2") in new stack
-- <SIP/demo-alice-00000008> Playing 'press-1.gsm' (language 'en')
-- <SIP/demo-alice-00000008> Playing 'or.gsm' (language 'en')
-- <SIP/demo-alice-00000008> Playing 'press-2.gsm' (language 'en')
```

If you don't enter anything at the auto-attendant menu and instead wait approximately ten seconds, you should hear (and see) Asterisk go to the **t** extension as well.

Record Application

For creating your own auto-attendant or IVR menus, you're probably going to want to record your own custom prompts. An easy way to do this is with the **Record()** application. The **Record()** application plays a beep, and then begins recording audio until you press the hash key (**#**) on your keypad. It then saves the audio to the filename specified as the first parameter to the application and continues on to the next priority in the extension. If you hang up the call before pressing the hash key, the audio will not be recorded. For example, the following extension records a sound prompt called **custom-menu** in the **gsm** format in the **en/** sub-directory, and then plays it back to you.

```
exten => 6597,1,Answer(500)
    same => n,Record(en/custom-menu.gsm)
    same => n,Wait(1)
    same => n,Playback(custom-menu)
    same => n,Hangup()
```

✅ **Recording Formats**
When specifiying a file extension when using the **Record()** application, you must choose a file extension which represents one of the supported file formats in Asterisk. For the complete list of file formats supported in your Asterisk installation, type **core show file formats** at the Asterisk command-line interface.

You've now learned the basics of how to create a simple auto-attendant menu. Now let's build a more practical menu for callers to be able to reach Alice or Bob or the dial-by-name directory.

Procedure 216.1. Building a Practical Auto-Attendant Menu

1. Add an extension 6599 to the [docs:users] context which sends the calls to a new context we'll build called [docs:day-menu]. Your extension should look something like:

```
exten=>6599,1,Goto(day-menu,s,1)
```

2. Add a new context called **[docs:day-menu]**, with the following contents:

```
[day-menu]
exten => s,1,Answer(500)
same => n(loop),Background(custom-menu)
same => n,WaitExten()

exten => 1,1,Goto(users,6001,1)

exten => 2,1,Goto(users,6002,1)

exten => 9,1,Directory(vm-demo,users,fe)
exten => *,1,VoiceMailMain(@vm-demo)

exten => i,1,Playback(option-is-invalid)
same => n,Goto(s,loop)

exten => t,1,Playback(are-you-still-there)
same => n,Goto(s,loop)
```

1. Dial extension **6597** to record your auto-attendant sound prompt. Your sound prompt should say something like "Thank you for calling! Press one for Alice, press two for Bob, or press 9 for a company directory". Press the hash key (**#**) on your keypad when you're finished recording, and Asterisk will play it back to you. If you don't like it, simply dial extension **6597** again to re-record it.
2. Dial extension **6599** to test your auto-attendant menu.

In just a few lines of code, you've created your own auto-attendant menu. Feel free to experiment with your auto-attendant menu before moving on to the next section.

Dialplan Architecture

In this section, we'll begin adding structure to our dialplan. We'll begin by talking about variables and how to use them, as well as how to manipulate them. Then we'll cover more advanced topics, such as pattern matching and using include statements to build classes of functionality.

Variables

Variables are used in most programming and scripting languages. In Asterisk, we can use variables to simplify our dialplan and begin to add logic to the system. A variable is simply a container that has both a name and a value. For example, we can have a variable named **COUNT** which has a value of three. Later on, we'll show you how to route calls based on the

value of a variable. Before we do that, however, let's learn a bit more about variables. The names of variables are case-sensitive, so **COUNT** is different than **Count** and **count**. Any channel variables created by Asterisk will have names that are completely upper-case, but for your own channels you can name them however you would like.

In Asterisk, we have two different types of variables: *channel variables* and *global variables*.

Channel Variables Basics

Channel variables are variables that are set for the current channel (one leg of a bridged phone call). They exist for the lifetime of the channel, and then go away when that channel is hung up. Channel variables on one particular channel are completely independent of channel variables on any other channels; in other words, two channels could each have variables called COUNT with different values.

To assign a value to a channel variable, we use the **Set()** application. Here's an example of setting a variable called **COUNT** to a value of **3**.

```
exten=>6123,1,Set(COUNT=3)
```

To retrieve the value of a variable, we use a special syntax. We put a dollar sign and curly braces around the variable name, like **${COUNT}**

When Asterisk sees the dollar sign and curly braces around a variable name, it substitutes in the value of the variable. Let's look at an example with the **SayNumber()** application.

```
exten=>6123,1,Set(COUNT=3)
exten=>6123,n,SayNumber(${COUNT})
```

In the second line of this example, Asterisk replaces the **${COUNT}** text with the value of the **COUNT** variable, so that it ends up calling **SayNumber(3)**.

Global Variables Basics

Global variables are variables that don't live on one particular channel — they pertain to all calls on the system. They have global scope. There are two ways to set a global variable. The first is to declare the variable in the **[globals]** section of **extensions.conf**, like this:

```
[globals]
MYGLOBALVAR=somevalue
```

You can also set global variables from dialplan logic using the **GLOBAL()** dialplan function along with the **Set()** application. Simply use the syntax:

```
exten=>6124,1,Set(GLOBAL(MYGLOBALVAR)=somevalue)
```

To retrieve the value of a global channel variable, use the same syntax as you would if you were retrieving the value of a channel variable.

Manipulating Variables Basics

It's often useful to do string manipulation on a variable. Let's say, for example, that we have a variable named **NUMBER** which represents a number we'd like to call, and we want to strip off the first digit before dialing the number. Asterisk provides a special syntax for doing just that, which looks like **${variable[:skip[docs::length]]}**.

The optional **skip** field tells Asterisk how many digits to strip off the front of the value. For example, if **NUMBER** were set to a value of **98765**, then **${NUMBER:2}** would tell Asterisk to remove the first two digits and return **765**.

If the skip field is negative, Asterisk will instead return the specified number of digits from the end of the number. As an example, if **NUMBER** were set to a value of **98765**, then **${NUMBER:-2}** would tell Asterisk to return the last two digits of the variable, or **65**.

If the optional **length** field is set, Asterisk will return at most the specified number of digits. As an example, if **NUMBER** were set to a value of **98765**, then **${NUMBER:0:3}** would tell Asterisk not to skip any characters in the beginning, but to then return only the three characters from that point, or **987**. By that same token, **${NUMBER:1:3}** would return **876**.

Variable Inheritance Basics

When building your Asterisk dialplan, it may be useful to have one channel inherit variables from another channel. For example, imagine that Alice's call has a channel variable containing an account code, and you'd like to pass that variable on to Bob's channel when Alice's call gets bridged to Bob. We call this variable inheritance. There are two levels of variable inheritance in Asterisk: *single inheritance* and *multiple inheritance*.

Multiple Inheritance

Multiple inheritance means that a channel variable will be inherited by created (spawned) channels, and it will continue to be inherited by any other channels created by the spawned channels. To set multiple inheritance on a channel, preface the variable name with two underscores when giving it a value with the **Set()** application, as shown below.

```
exten=>6123,1,Set(__ACCOUNT=5551212)
```

Single Inheritance

Single inheritance means that a channel variable will be inherited by created (spawned) channels, but not propogate from there to any other swawned channels. To follow our example above, if Alice sets a channel variable with single inheritance and calls Bob, Bob's channel will inherit that channel variable, but the channel variable won't get inherited by any channels that

might get spawned by Bob's channel (if the call gets transferred, for example). To set single inheritance on a channel, preface the variable name with an underscore when giving it a value with the **Set()** application, as shown below.

```
exten=>6123,1,Set(_ACCOUNT=5551212)
```

Using the CONTEXT, EXTEN, PRIORITY, UNIQUEID, and CHANNEL Variables

Now that you've learned a bit about variables, let's look at a few of the variables that Asterisk automatically creates.

Asterisk creates channel variables named **CONTEXT**, **EXTEN**, and **PRIORITY** which contain the current context, extension, and priority. We'll use them in pattern matching (below), as well as when we talk about macros in [Section 308.10. Macros]. Until then, let's show a trivial example of using **${EXTEN}** to read back the current extension number.

```
exten=>6123,1,SayNumber(${EXTEN})
```

If you were to add this extension to the **[docs:users]** context of your dialplan and reload the dialplan, you could call extension **6123** and hear Asterisk read back the extension number to you.

Another channel variable that Asterisk automatically creates is the **UNIQUEID** variable. Each channel within Asterisk receives a unique identifier, and that identifier is stored in the **UNIQUEID** variable. The **UNIQUEID** is in the form of **1267568856.11**, where **1267568856** is the Unix epoch, and **11** shows that this is the eleventh call on the Asterisk system since it was last restarted.

Last but not least, we should mention the **CHANNEL** variable. In addition to a unique identifier, each channel is also given a channel name and that channel name is set in the **CHANNEL** variable. A SIP call, for example, might have a channel name that looks like **SIP/george-0000003b**, for example.

The Verbose and NoOp Applications

Asterisk has a convenient dialplan applications for printing information to the command-line interface, called **Verbose()**. The **Verbose()** application takes two parameters: the first parameter is the minimum verbosity level at which to print the message, and the second parameter is the message to print. This extension would print the current channel identifier and unique identifier for the current call, if the verbosity level is two or higher.

```
exten=>6123,1,Verbose(2,The channel name is ${CHANNEL})
exten=>6123,n,Verbose(2,The unique id is ${UNIQUEID})
```

The **NoOp()** application stands for "No Operation". In other words, it does nothing. Because of the way Asterisk prints everything to the console if your verbosity level is three or higher, however, the **NoOp()** application is often used to print debugging information to the console like

the **Verbose()** does. While you'll probably come across examples of the **NoOp()** application in other examples, we recommend you use the **Verbose()** application instead.

The Read Application

The **Read()** application allows you to play a sound prompt to the caller and retrieve DTMF input from the caller, and save that input in a variable. The first parameter to the **Read()** application is the name of the variable to create, and the second is the sound prompt or prompts to play. (If you want multiple prompts, simply concatenate them together with ampersands, just like you would with the **Background()** application.) There are some additional parameters that you can pass to the **Read()** application to control the number of digits, timeouts, and so forth. You can get a complete list by running the core show application read command at the Asterisk CLI. If no timeout is specified, **Read()** will finish when the caller presses the hash key (**#**) on their keypad.

```
exten=>6123,1,Read(Digits,enter-ext-of-person)
exten=>6123,n,Playback(you-entered)
exten=>6123,n,SayNumber(${Digits})
```

In this example, the **Read()** application plays a sound prompt which says "Please enter the extension of the person you are looking for", and saves the captured digits in a variable called **Digits**. It then plays a sound prompt which says "You entered" and then reads back the value of the **Digits** variable.

Pattern Matching

The next concept we'll cover is called *pattern matching*. Pattern matching allows us to create extension patterns in our dialplan that match more than one possible dialed number. Pattern matching saves us from having to create an extension in the dialplan for every possible number that might be dialed.

When Alice dials a number on her phone, Asterisk first looks for an extension (in the context specified by the channel driver configuration) that matches exactly what Alice dialed. If there's no exact match, Asterisk then looks for a pattern match that matches. After we show the syntax and some basic examples of pattern matching, we'll explain how Asterisk finds the best match if there are two or more patterns which match the dialed number.

Pattern matches always begin with an underscore. This is how Asterisk recognizes that the extension is a pattern and not just an extension with a funny name. Within the pattern, we use various letter and characters to represent sets or ranges of numbers. Here are the most common letters:

X

The letter **X** represents a single digit from 0 to 9.

Z

The letter **Z** represents any digit from 1 to 9.

N

The letter **N** represents a single digit from 2 to 9.

Now let's look at a sample pattern. If you wanted to match all four-digit numbers that had the first two digits as six and four, you would create an extension that looks like:

```
exten => _64XX,1,SayDigits(${EXTEN})
```

In this example, each **X** represents a single digit, with any value from zero to nine. We're essentially saying "The first digit must be a six, the second digit must be a four, the third digit can be anything from zero to nine, and the fourth digit can be anything from zero to nine".

If we want to be more specific about a range of numbers, we can put those numbers or number ranges in square brackets. For example, what if we wanted the second digit to be either a three or a four? One way would be to create two patterns (**_64XX** and **_63XX**), but a more compact method would be to do **_6[docs:34]XX**. This specifies that the first digit must be a six, the second digit can be either a three or a four, and that the last two digits can be anything from zero to nine.

You can also use ranges within square brackets. For example, **[docs:1-468]** would match a single digit from one through four or six or eight. It does not match any number from one to four hundred sixty-eight!

Within Asterisk patterns, we can also use a couple of other characters to represent ranges of numbers. The period character (**.**) within a pattern matches on one or more remaining digits in the pattern. It typically appears at the end of a pattern match, especially when you want to match extensions of an indeterminate length. As an example, the pattern **_9876.** would match any number that began with **9876** and had at least one more character or digit.

The exclamation mark (**!**) character is similar to the period and also matches one more more remaining characters, but is used in overlap dialing. For example, **_9876!** would match any number that began with **9876**, and would respond that the number was complete as soon as there was an unambiguous match.

> ⛔ **Be Careful With Wildcards in Pattern Matches**
> Please be extremely cautious when using the period and exclamation mark characters in your pattern matches. They match more than just digits, they also match on characters as well, and if you're not careful to filter the input from your callers, a malicious caller might try to use these wildcards to bypass security boundaries on your system.
>
> For a more complete explanation of this topic and how you can protect yourself, please refer to the **README-SERIOUSLY.bestpractices.txt** file in the Asterisk source code.

Now let's show what happens when there is more than one pattern that matches the dialed number. How does Asterisk know which pattern to choose as the best match?

Asterisk uses a simple set of rules to determine the best match. They are:

1. Examine the first digit eliminate any patterns which don't match the first digit of the dialed number
2. Sort the remaining patterns based on the most constrained match for the current digit. By most constrained, we mean the pattern that has the fewest possible matches for this digit. As an example, the **N** character has 8 possible matches (two through nine), while an **X** has ten possible matches.
3. In the case of a match, sort the patterns in ASCII sort order. For example, _**[docs:234]X** and _**[docs:345]X** have three possible matches in the first digit, but **234** comes before **345** in ASCII sort order.
4. Move on to the next digit (moving digit by digit from left to right), and eliminate any patterns which don't match the current digit of the dialed number. Then continue back at step number two.
5. After you've examined all the digits, return the match that has been sorted to the top of the list.

Let's look at an example to better understand how this works. Let's assume Alice dials extension 6401, and she has the following patterns in her dialplan:

```
exten => _64XX,1,SayAlpha(A)
exten => _640X,1,SayAlpha(B)
exten => _64NX,1,SayAlpha(C)
exten => _6XX1,1,SayAlpha(D)
```

Can you tell (without reading ahead) which one would match?

Let's walk step by step through the rules explained above, and see what happens when Alice dials **6401**.

Rule 1

We look at the first digit, and all the patterns match.

Rule 2

Each of the patterns have the same number of possible matches on this digit (one match — the number six).

Rule 3

We sort the patterns in ASCII sort order.

Rule 4

We move on to the second digit. There are no patterns that can be eliminated based on the second digit, so we go back to rule two for this digit.

Rule 2

The three patterns with a **4** in the second digit are more constrained than the **X**, so they get sorted to the top.

Rule 3

The top three patterns get sorted in ASCII sort order, since they are tied in the number of possible matches.

Rule 4

We move on to the third digit. The third pattern (the one that would call **SayAlpha(C)**) is eliminated, because the third digit of this pattern (the **N**) doesn't match the third dialed digit (the **0**). The other patterns match, so we now go back to rule two.

Rule 2

The second pattern (the one that would call **SayAlpha(B)**) is the most constrained, as it only has a single possibility, so it gets sorted to the top.

Rule 3

There are no ties at the top of the sorting table, so we can move on to rule four.

Rule 4

We move on to the fourth digit. Since all the remaining patterns match, the second pattern remains at the top of the sorting table. You might be asking yourself... "What about the fourth pattern? Isn't it more constrained?" Remember that it was less constrained in an earlier digit, so it would only match if none of the other patterns above it in the sorting table matched on this digit.

Step 5

Since we have run out of digits, we know that Asterisk will match on the second pattern, as it is the one at the top of the sorting table.

To verify that Asterisk actually does sort the extensions in the manner that we've described, add the following extensions to the **[docs:users]** context of your own dialplan.

```
exten => _64XX,1,SayAlpha(A)
exten => _640X,1,SayAlpha(B)
exten => _64NX,1,SayAlpha(C)
exten => _6XX1,1,SayAlpha(D)
```

Reload the dialplan, and then type **dialplan show 6104@users** at the Asterisk CLI. Asterisk will show you what would match if you were to dial extension **6104** in the **[docs:users]** context.

```
server*CLI> dialplan show 6401@users
[ Context 'users' created by 'pbx_config' ]
  '_640X' =>        1. SayAlpha(B)
[pbx_config]
  '_64XX' =>        1. SayAlpha(A)
[pbx_config]
  '_6XX1' =>        1. SayAlpha(D)
[pbx_config]

-= 3 extensions (3 priorities) in 1 context. =-
```

You can then dial extension **6104** to try it out on your own.

Be Careful with Pattern Matching

Please be aware that because of the way auto-fallthrough works, if Asterisk can't find the next priority number for the current extension or pattern match, it will also look for that same priority in a less specific pattern match. Consider the following example:

```
exten => 6410,1,SayDigits(987)
exten => _641X,1,SayDigits(12345)
exten => _641X,n,SayDigits(54321)
```

If you were to dial extension 6410, you'd hear "nine eight seven five four three two one".

We strongly recommend you make the **Hangup()** application be the last priority of any extension to avoid this problem, unless you purposely want to fall through to a less specific match.

Include Statements

Include statements allow us to split up the functionality in our dialplan into smaller chunks, and then have Asterisk search multiple contexts for a dialed extension. Most commonly, this functionality is used to provide security boundaries between different classes of callers.

It is important to remember that when calls come into the Asterisk dialplan, they get directed to a particular context by the channel driver. Asterisk then begins looking for the dialed extension in the context specified by the channel driver. By using include statements, we can include other contexts in the search for the dialed extension.

Asterisk supports two different types of include statements: regular includes and time-based includes.

Include Statements Basics

To set the stage for our explanation of include statements, let's say that we want to organize our dialplan and create a new context called **[docs:features]**. We'll leave our extensions **6001** and

6002 for Alice and Bob in the **[docs:users]** context, and place extensions such as **6500** in the new **[docs:features]** context. When calls come into the users context and doesn't find a matching extension, the include statement tells Asterisk to also look in the new **[docs:features]** context.

The syntax for an include statement is very simple. You simply write **include =>** and then the name of the context you'd like to include from the existing context. If we reorganize our dialplan to add a **[docs:features]** context, it might look something like this:

```
[users]
include => features

exten => 6001,1,Dial(SIP/demo-alice,20)
    same => n,VoiceMail(6001@vm-demo,u)

exten => 6002,1,Dial(SIP/demo-bob,20)
    same => n,VoiceMail(6002@vm-demo,u)

[features]
exten => 6000,1,Answer(500)
    same => n,Playback(hello-world)
    same => n,Hangup()

exten => 6500,1,Answer(500)
    same => n,VoiceMailMain(@vm-demo)
```

✅ **Location of Include Statements**
Please note that in the example above, we placed the include statement before extensions **6001** and **6002**. It could have just as well come after. Asterisk will always try to find a matching extension in the current context first, and only follow the include statement to a new context if there isn't anything that matches in the current context.

Using Include Statements to Create Classes of Service

Now that we've shown the basic syntax of include statements, let's put some include statements to good use. Include statements are often used to build chains of functionality or classes of service. In this example, we're going to build several different contexts, each with its own type of outbound calling. We'll then use include statements to chain these contexts together.

In these examples, we're going to assuming that a seven-digit number that does not begin with a zero or a one is a local (non-toll) call. Ten-digit numbers (where neither the first or fourth digits begin with zero or one) are also treated as local calls. A one, followed by ten digits (where neither the first or fourth digits begin with zero or one) is considered a long-distance (toll) call. Again, feel free to modify these examples to fit your own particular circumstances.

First, let's create a new context for local calls.

```
[local]
; seven-digit local numbers
exten => _NXXXXXX,1,Dial(SIP/provider/${EXTEN})

; ten-digit local numbers
exten => _NXXNXXXXXX,1,Dial(SIP/provider/${EXTEN})

; emergency services (911), and other three-digit services
exten => NXX,1,Dial(SIP/provider/${EXTEN})

; if you don't find a match in this context, look in [users]
include => users
```

Remember that the variable **${EXTEN}** will get replaced with the dialed extension. For example, if Bob dials **5551212** in the **local** context, Asterisk will execute the Dial application with **SIP/provider/5551212** as the first parameter. (This syntax means "Dial out to the account named provider using the SIP channel driver, and dial the number **5551212**.)

Next, we'll build a long-distance context, and link it back to the **local** context with an include statement. This way, if you dial a local number and your phone's channel driver sends the call to the **longdistance** context, Asterisk will search the **local** context if it doesn't find a matching pattern in the **longdistance** context.

```
[longdistance]
; 1+ ten digit long-distance numbers
exten => _1NXXNXXXXXX,1,Dial(SIP/provider/${EXTEN})


; if you don't find a match in this context, look in [local]
include => local
```

Last but not least, let's add an **[docs:international]** context. In North America, you dial 011 to signify that you're going to dial an international number.

```
[international]
; 1+ ten digit long-distance numbers
exten => _011.,1,Dial(SIP/provider/${EXTEN})


; if you don't find a match in this context, look in [longdistance]
include => longdistance
```

And there we have it -- a simple chain of contexts going from most privileged (international calls) down to lease privileged (local calling).

At this point, you may be asking yourself, "What's the big deal? Why did we need to break them up into contexts, if they're all going out the same outbound connection?" That's a great question! The primary reason for breaking the different classes of calls into separate contexts is so that we can enforce some security boundaries.

Do you remember what we said earlier, that the channel drivers point inbound calls at a particular context? In this case, if we point a phone at the **[docs:local]** context, it could only make local and internal calls. On the other hand, if we were to point it at the **[docs:international]** context, it could make international and long-distance and local and internal calls. Essentially, we've created different classes of service by chaining contexts together with include statements, and using the channel driver configuration files to point different phones at different contexts along the chain.

Many people find it instructive to look at a visual diagram at this point, so let's draw ourselves a map of the contexts we've created so far.

Insert graphic showing chain of includes from international through long-distance to local and to users and features

In this graphic, we've illustrated the various contexts and how they work together. We've also shown that Alice's phone is pointed at the **[docs:international]** context, while Bob's phone is only pointed at the **[docs:local]** context.

Please take the next few minutes and implement a series of chained contexts into your own dialplan, similar to what we've explained above. You can then change the configuration for Alice and Bob (in **sip.conf**, since they're SIP phones) to point to different contexts, and see what

happens when you attempt to make various types of calls from each phone.

Installing Asterisk From Source

One popular option for installing Asterisk is to download the source code and compile it yourself. While this isn't as easy as using package management or using an Asterisk-based Linux distribution, it does let you decide how Asterisk gets built, and which Asterisk modules are built.

In this section, you'll learn how to download and compile the Asterisk source code, and get Asterisk installed.

What to Download?

On a typical system, you'll want to download three components:

- Asterisk
- DAHDI
- libpri

The **libpri** library allows Asterisk to communicate with ISDN connections. (We'll cover more about ISDN connections in Section 450.8, "Intro to ISDN PRI and BRI Connections".) While not always necessary, we recommend you install it on new systems.

The **DAHDI** library allows Asterisk to communicate with analog and digital telephones and telephone lines, including connections to the Public Switched Telephone Network, or PSTN. It should also be installed on new systems, even if you don't immediately plan on using analog or digital connections to your Asterisk system.

DAHDI

DAHDI stands for Digium Asterisk Hardware Device Interface, and is a set of drivers and utilities for a number of analog and digital telephony cards, such as those manufactured by Digium. The DAHDI drivers are independent of Asterisk, and can be used by other applications. DAHDI was previously called Zaptel, as it evolved from the Zapata Telephony Project.

The DAHDI code can be downloaded as individual pieces (**dahdi-linux** for the DAHDI drivers, and **dahdi-tools** for the DAHDI utilities. They can also be downloaded as a complete package called **dahdi-linux-complete**, which contains both the Linux drivers and the utilities.

> ✅ Why is DAHDI split into different pieces?
>
> DAHDI has been split into two pieces (the Linux drivers and the tools) as third parties have begun porting the DAHDI drivers to other operating systems, such as FreeBSD. Eventually, we may have dahdi-linux, dahdi-freebsd, and so on.

The current version of libpri, DAHDI, and Asterisk can be downloaded from http://downloads.digium.com/pub/telephony/.

System Requirements

In order to compile and install Asterisk, you'll need to install a C compiler and a number of system libraries on your system.

- Compiler
- System Libraries

Compiler

The compiler is a program that takes source code (the code written in the C programming language in the case of Asterisk) and turns it into a program that can be executed. While any C compiler should be able to compile the Asterisk code, we strongly recommend that you use the **GCC** compiler. Not only is it the most popular free C compiler on Linux and Unix systems, but it's also the compiler that the Asterisk developers are using.

If the GCC compiler isn't already installed on your machine, simply use appropriate package management system on your machine to install it. You'll also want to install the C++ portion of GCC as well, as certain Asterisk modules will use it.

System Libraries

In addition to the C compiler, you'll also need a set of system libraries. These libraries are used by Asterisk and must be installed before you can compile Asterisk. On most operating systems, you'll need to install both the library and it's corresponding development package.

> ✅ Development libraries
>
> For most operating systems, the development packages will have -dev or -devel on the end of the name. For example, on a Red Hat Linux system, you'd want to install both the "openssl" and "openssl-devel" packages.

A list of libraries you'll need to install include:

- OpenSSL
- ncurses
- newt
- libxml2
- Kernel headers (for building DAHDI drivers)

We recommend you use the package management system of your operating system to install these libraries before compiling and installing libpri, DAHDI, and Asterisk.

Untarring the Source

When you download the source for libpri, DAHDI, and Asterisk you'll typically end up with files with a .tar.gz or .tgz file extension. These files are affectionately known as tarballs. The name comes from the tar Unix utility, which stands for tape archive. A tarball is a collection of other files combined into a single file for easy copying, and then often compressed with a utility such as GZip.

To extract the source code from the tarballs, we'll use the tar command. The commands below assume that you've downloaded the tarballs for libpri, DAHDI, and Asterisk to the /usr/local/src directory on a Linux machine. (You'll probably need to be logged in as the root user to be able to write to that directory.) We're also going to assume that you'll replace the letters X, Y, and Z with the actual version numbers from the tarballs you downloaded. Also please note that the command prompt may be slightly different on your system than what we show here. Don't worry, the commands should work just the same.

First, we'll change to the directory where we downloaded the source code:

```
[root@server ~]# cd /usr/local/src
```

Next, let's extract the source code from each tarball using the tar command. The -zxvf parameters to the tar command tell it what we want to do with the file. The z option tells the system to unzip the file before continuing, the x option tells it to extract the files from the tarball, the v option tells it to be verbose (write out the name of every file as it's being extracted, and the f option tells the tar command that we're extracting the file from a tarball file, and not from a tape.

```
[root@server src]# tar -zxvf libpri-1.X.Y.tar.gz

[root@server src]# tar -zxvf dahdi-linux-complete-2.X.Y+2.X.Y.tar.gz

[root@server src]# tar -zxvf asterisk-1.6.X.Y.tar.gz
```

You should now notice that a new sub-directory was created for each of the tarballs, each containing the extracted files from the corresponding tarball. We can now compile and install each of the components.

Building and Installing LibPRI

First, let's compile and install libpri. Again, we'll assume that you'll replace the letters X, Y, and Z with the actual version numbers from the tarballs you downloaded.

```
[root@server src]# cd libpri-1.X.Y
```

This command changes directories to the **libpri** source directory.

```
[root@server libpri-1.X.Y]# make
```

This command compiles the **libpri** source code into a system library.

```
[root@server libpri-1.X.Y]# make install
```

This command installs the **libpri** library into the proper system library directory

Building and Installing DAHDI

Now that we have libpri installed, let's install DAHDI. On Linux, we will use the DAHDI-linux-complete tarball, which contains both the DAHDI Linux drivers as well as the DAHDI tools. Again, we're assuming that you've untarred the tarball in the /usr/local/src directory, and that you'll replace X and Y with the appropriate version numbers.

```
[root@server src]# cd dahdi-linux-complete-2.X.Y+2.X.Y

[root@server dahdi-linux-complete-2.X.Y+2.X.Y]# make

[root@server dahdi-linux-complete-2.X.Y+2.X.Y]# make install

[root@server dahdi-linux-complete-2.X.Y+2.X.Y]# make config
```

Checking Asterisk Requirements

Now it's time to compile and install Asterisk. Let's change to the directory which contains the Asterisk source code.

```
[root@server dahdi-linux-complete-2.X.Y+2.X.Y]# cd
/usr/local/src/asterisk-1.6.X.Y
```

Next, we'll run a command called **./configure**, which will perform a number of checks on the operating system, and get the Asterisk code ready to compile on this particular server.

```
[root@server asterisk-1.6.X.Y]# ./configure
```

This will run for a couple of minutes, and warn you of any missing system libraries or other dependencies. Upon completion, you should see a message that looks similar to the one shown below. (Obviously, your host CPU type may be different than the below.)

```
                .$$$$$$$$$$$$$$$=..
             .$7$7..          .7$$7:.
           .$7$7..              .7$$7:.
         .$$:.                     ,$7.7
       .$7.      7$$$$           .$$77
    ..$$.         $$$$$          .$$$7
   ..7$   .?.     $$$$$    .?.     7$$$.
   $.$.   .$$$7. $$$$7 .7$$$.      .$$$.
 .777.   .$$$$$77$$$77$$$$$7.      $$$,
 $$$~      .7$$$$$$$$$$$$$7.      .$$$.
.$$7          .7$$$$$$$7:          ?$$$.
$$$             ?7$$$$$$$$$I        .$$$7
$$$           .7$$$$$$$$$$$$$$      :$$$.
$$$           $$$$$7$$$$$$$$$$$$    .$$$.
$$$           $$$   7$$$7  .$$$    .$$$.
$$$$          $$$$7            .$$$.
7$$$7          7$$$$           7$$$
 $$$$$                        $$$
  $$$$7.                      $$   (TM)
   $$$$$$$.           .7$$$$$$  $$
    $$$$$$$$$$$$7$$$$$$$$$.$$$$$$
      $$$$$$$$$$$$$$$$.

configure: Package configured for: 
configure: OS type  : linux-gnu
configure: Host CPU : x86_64
configure: build-cpu:vendor:os: x86_64 : unknown : linux-gnu :
configure: host-cpu:vendor:os: x86_64 : unknown : linux-gnu :
```

✅ **Cached Data**
The **./configure** command caches certain data to speed things up if it's invoked multiple times. To clear all the cached data, you can use the following command to completely clear out any cached data from the Asterisk build system.

```
[root@server asterisk-1.6.X.Y]# make distclean
```

Using Menuselect to Select Asterisk Options

The next step in the build process is to tell Asterisk which modules[docs:1] to compile and install, as well as set various compiler options. These settings are all controlled via a menu-driven system called **menuselect**. To access the menuselect system, type:

```
[root@server asterisk-1.6.X.Y]# make menuselect
```

> ⚠ **Terminal Window**
> Your terminal window size must be at least eighty characters wide and twenty-one lines high, or menuselect will not work. Instead, you'll get an error message stating
>
> ```
> Terminal must be at least 80 x 21.
> ```

> ⚠ **Asterisk 1.8+**
>
> ```
> Terminal must be at least 80 x 27.
> ```

The menuselect menu should look like the screen-shot below. On the left-hand side, you have a list of categories, such as **Applications**, **Channel Drivers**, and **PBX Modules**. On the right-hand side, you'll see a list of modules that correspond with the select category. At the bottom of the screen you'll see two buttons. You can use the **Tab** key to cycle between the various sections, and press the **Enter** key to select or unselect a particular module. If you see [docs:**] next to a module name, it signifies that the module has been selected. If you see *XXX** next to a module name, it signifies that the select module cannot be built, as one of its dependencies is missing. In that case, you can look at the bottom of the screen for the line labeled **Depends upon**: for a description of the missing dependency.

When you're first learning your way around Asterisk on a test system, you'll probably want to stick with the default settings in menuselect. If you're building a production system, however, you may not wish to build all of the various modules, and instead only build the modules that your system is using.

```
┌─────────┤ Asterisk Module and Build Option Selection ├─────────┐
│                                                                │
│  Applications           [*] app_adsiprog              ↑        │
│  Call Detail Recording  [*] app_alarmreceiver         ▪        │
│  Channel Drivers        [*] app_amd                   ▒        │
│  Codec Translators      [*] app_authenticate          ▒        │
│  Format Interpreters    [*] app_cdr                   ▒        │
│  Dialplan Functions     [*] app_chanisavail           ▒        │
│  PBX Modules            [*] app_channelredirect       ▒        │
│  Resource Modules       [*] app_chanspy               ▒        │
│  Test Modules           [*] app_controlplayback       ↓        │
│                                                                │
│  Asterisk ADSI Programming Application                         │
│                                                                │
│      Depends on: res_adsi(M)     ┌──────────┐  ┌──────────┐   │
│         Can use: N/A             │Save & Exit│  │   Exit   │   │
│   Conflicts with: N/A            └──────────┘  └──────────┘   │
│                                                                │
└────────────────────────────────────────────────────────────────┘
  <ENTER> toggles selection | <F12> saves & exits | <ESC> exits without save
```

✅ **Easier Debugging of Asterisk Crashes**

If you're finding that Asterisk is crashing on you, there's a setting in menuselect that will help provide additional information to the Asterisk developers. Go into menuselect, select the the Compiler Flags section (you'll need to scroll down in the left-hand list), and select the DONT_OPTIMIZE setting. Then rebuild Asterisk as shown below. While the Asterisk application will be slightly larger, it will provide additional debugging symbols in the event of a crash.

We should also inform people that the sound prompts are selected in menuselect as well

When you are finished selecting the modules and options you'd like in **menuselect**, press **F12** to save and exit, or highlight the **Save and Exit** button and press enter.

Building and Installing Asterisk

Now we can compile and install Asterisk. To compile Asterisk, simply type make at the Linux command line.

```
[root@server asterisk-1.6.X.Y]# make
```

The compiling step will take several minutes, and you'll see the various file names scroll by as they are being compiled. Once Asterisk has finished compiling, you'll see a message that looks like:

```
+--------- Asterisk Build Complete ---------+
+ Asterisk has successfully been built, and +
+ can be installed by running:              +
+                                           +
+                make install               +
+-------------------------------------------+
+--------- Asterisk Build Complete ---------+
```

As the message above suggests, our next step is to install the compiled Asterisk program and modules. To do this, use the **make install** command.

```
[root@server asterisk-1.6.X.Y]# make install
```

When finished, Asterisk will display the following warning:

```
+---- Asterisk Installation Complete -------+
+                                           +
+     YOU MUST READ THE SECURITY DOCUMENT   +
+                                           +
+ Asterisk has successfully been installed. +
+ If you would like to install the sample   +
+ configuration files (overwriting any      +
+ existing config files), run:              +
+                                           +
+                make samples               +
+                                           +
+-------------------------------------------+
+---- Asterisk Installation Complete -------+
```

🛑 Security Precautions
As the message above suggests, we very strongly recommend that you read the security documentation before continuing with your Asterisk installation. Failure to read and follow the security documentation can leave your system vulnerable to a number of security issues, including toll fraud.

If you installed Asterisk from a tarball (as shown above), the security information is located in a PDF file named asterisk.pdfin the tex/ sub-directory of the source code. If that file doesn't exist, please install the rubber application on your system, and then type:

```
[root@server asterisk-1.6.X.Y]# make pdf
```

Installing Sample Files

To install a set of sample configuration files for Asterisk, type:

```
[root@server asterisk-1.6.X.Y]# make samples
```

Any existing sample files which have been modified will be given a **.old** file extension. For example, if you had an existing file named **extensions.conf**, it would be renamed to **extensions.conf.old** and the sample dialplan would be installed as **extensions.conf**.

Installing Initialization Scripts

Now that you have Asterisk compiled and installed, the last step is to install the initialization script, or initscript. This script starts Asterisk when your server starts, and can be used to stop or restart Asterisk as well. To install the initscript, use the **make config** command.

```
[root@server asterisk-1.6.X.Y]# make config
```

As your Asterisk system runs, it will generate logfiles. It is recommended to install the logrotation script in order to compress and rotate those files, to save disk space and to make searching them or cataloguing them easier. To do this, use the **make install-logrotate** command.

```
[root@server asterisk-1.6.X.Y]# make install-logrotate
```

Validating Your Installation

Before continuing on, let's check a few things to make sure your system is in good working order. First, let's make sure the DAHDI drivers are loaded. You can use the **lsmod** under Linux to list all of the loaded kernel modules, and the **grep** command to filter the input and only show the modules that have **dahdi** in their name.

```
[root@server asterisk-1.6.X.Y]# lsmod | grep dahdi
```

If the command returns nothing, then DAHDI has not been started. Start DAHDI by running:

```
[root@server asterisk-1.6.X.Y]# /etc/init.d/dadhi start
```

Many Linux distributions have different methods for starting initscripts. On most Red Hat based distributions (such as Red Hat Enterprise Linux, Fedora, and CentOS) you can run:

```
[root@server asterisk-1.6.X.Y]# service dahdi start
```

Distributions based on Debian (such as Ubuntu) have a similar command, though it's not commonly used:

```
[root@server asterisk-1.6.X.Y]# invoke-rc.d dahdi start
```

If you have DAHDI running, the output of **lsmod | grep dahdi** should look something like the output below. (The exact details may be different, depending on which DAHDI modules have been built, and so forth.)

```
[root@server asterisk-1.6.X.Y]# lsmod | grep dahdi
dahdi_dummy        4288 0
dahdi_transcode  7928 1 wctc4xxp
dahdi_voicebus   40464 2 wctdm24xxp,wcte12xp
dahdi           196544 12
dahdi_dummy,wctdm24xxp,wcte11xp,wct1xxp,wcte12xp,wct4xxp
crc_ccitt          2096 1 dahdi
```

Now that DAHDI is running, you can run **dahdi_hardware** to list any DAHDI-compatible devices in your system. You can also run the **dahdi_tool** utility to show the various DAHDI-compatible devices, and their current state.

To check if Asterisk is running, you can use the Asterisk initscript.

```
[root@server asterisk-1.6.X.Y]# /etc/init.d/asterisk status
asterisk is stopped
```

To start Asterisk, we'll use the initscript again, this time giving it the start action:

```
[root@server asterisk-1.6.X.Y]# /etc/init.d/asterisk start
Starting asterisk:
```

When Asterisk starts, it runs as a background service (or daemon), so you typically won't see any response on the command line. We can check the status of Asterisk and see that it's running using the command below. (The process identifier, or pid, will obviously be different on your

system.)

```
[root@server asterisk-1.6.X.Y]# /etc/init.d/asterisk status
asterisk (pid 32117) is running...
```

And there you have it! You've compiled and installed Asterisk, DAHDI, and libpri from source code.

Getting Started with Asterisk

In this section, we'll show you how to get started with Asterisk, and how to get around on the Asterisk command-line interface (commonly abbreviated as CLI). We'll also show you how to troubleshoot common problems that you might encounter when first learning Asterisk

Connecting to the CLI

First, let's show you how to connect to the Asterisk command-line interface. As you should recall from the installation, Asterisk typically runs in the background as a service or daemon. If the Asterisk service is already running, type the command below to connect to its command-line interface.

```
[root@server ~]# asterisk -r
```

The -r parameter tells the system that you want to re-connect to the Asterisk service. If the reconnection is successful, you'll see something like this:

```
[root@server ~]# asterisk -r
Asterisk version, Copyright (C) 1999 - 2010 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show
warranty' for details.
This is free software, with components licensed under the GNU
General Public
License version 2 and other licenses; you are welcome to
redistribute it under
certain conditions. Type 'core show license' for details.
=======================================================================
to Asterisk version currently running on server (pid = 11187)
server*CLI>
```

Notice the ***CLI>** text? That's your Asterisk command-line prompt. All of the Asterisk CLI commands take the form of **module action parameters....** For example, type **core show uptime** to see how long Asterisk has been running.

```
server*CLI> core show uptime
System uptime: 1 hour, 34 minutes, 17 seconds
Last reload: 1 hour, 34 minutes, 17 seconds
```

You can use the built-in help to get more information about the various commands. Simply type **core show help** at the Asterisk prompt for a full list of commands, or **core show help command** for help on a particular command.

If you'd like to exit the Asterisk console and return to your shell, just use the **quit** command from the CLI. Such as:

```
server*CLI> quit
```

> ✅ **Executing Command Outside Of CLI**
> You can execute an Asterisk command from outside the CLI:
>
> ```
> $ asterisk -rx "reload"
> ```
>
> ```
> $ asterisk -rx "help" | grep -i "sip"
> ```

Stopping and Restarting Asterisk

There are four common commands related to stopping the Asterisk service. They are:

1. **core stop now** - This command stops the Asterisk service immediately, ending any calls in progress.
2. **core stop gracefully** - This command prevents new calls from starting up in Asterisk, but allows calls in progress to continue. When all the calls have finished, Asterisk stops.
3. **core stop when convenient** - This command waits until Asterisk has no calls in progress, and then it stops the service. It does not prevent new calls from entering the system.

There are three related commands for restarting Asterisk as well.

1. **core restart now** - This command restarts the Asterisk service immediately, ending any calls in progress.
2. **core restart gracefully** - This command prevents new calls from starting in Asterisk, but allows calls in progress to continue. When all the calls have finished, Asterisk restarts.
3. **core restart when convenient** - This command waits until Asterisk has no calls in progress, and then it restarts the service. It does not prevent new calls from entering the system.

There is also a command if you change your mind.

- **core abort shutdown** - This command aborts a shutdown or restart which was previously initiated with the gracefully or when convenient options.

Changing the Verbose and Debug Levels

Asterisk has two different classes of messages that appear in the command-line interface. The first class is called **verbose** messages. Verbose messages give information about the calls on the system, as well as notices, warnings, and errors. Verbose messages are intended for Asterisk administrators to be able to better manage their systems.

Asterisk allows you to control the verbosity level of the command-line interface. At a verbosity level of zero, you'll receive minimal information about calls on your system. As you increase the verbosity level, you'll see more and more information about the calls. For example, if you set the verbosity level to three or higher, you'll see each step a call takes as it makes its way through the dialplan. There are very few messages that only appear at verbosity levels higher than three.

To change the verbosity level, use the CLI command **core set verbose**, as shown below:

```
server*CLI> core set verbose 3
Verbosity was 0 and is now 3
```

You can also increase (but not decrease) the verbosity level when you connect to the Asterisk CLI from the Linux prompt, by using one or more **-v** parameters to the **asterisk** application. For example, this would connect to the Asterisk CLI and set the verbosity to three (if it wasn't already three or higher), because we added three -v parameters:

```
[root@server ~]# asterisk -vvvr
```

The second class of system messages is known as **debug** messages. These messages are intended for Asterisk developers, to give information about what's happening in the Asterisk program itself. They're often used by developers when trying to track down problems in the code, or to understand why Asterisk is behaving in a certain manner.

To change the debugging level, use the CLI command **core set debug**, as shown below:

```
server*CLI> core set debug 4
Core debug was 0 and is now 4
```

You can also increase (but not decrease) the debugging level when you connect to the Asterisk CLI from the Linux prompt. Simply add one or more **-d** parameters to the **asterisk** application.

```
[root@server ~]# asterisk \-ddddr
```

> ℹ **Verbose and Debug Levels**
> Please note that the verbose and debug levels are global settings, and apply to all of Asterisk, not just your command-line interface.
>
> We recommend that you set your verbosity level to three while learning Asterisk, so that you can get a feel for what is happening as calls are processed. On a busy production system, however, you'll want to set the verbosity level lower. We also recommend that you use debug messages sparingly, as they tend to be quite verbose and can affect call volume on busy systems.

Simple CLI Tricks

There are a couple of tricks that will help you on the Asterisk command-line interface. The most popular is tab completion. If you type the beginning of a command and press the Tab key, Asterisk will attempt to complete the name of the command for you, or show you the possible commands that start with the letters you have typed. For example, type co and then press the Tab key on your keyboard.

```
server*CLI> co[Tab]
config core
server*CLI> co
```

Now press the **r** key, and press tab again. This time Asterisk completes the word for you, as **core** is the only command that begins with **cor**. This trick also works with sub-commands. For example, type **core show** and press tab. (You may have to press tab twice, if you didn't put a space after the word **show**.) Asterisk will show you all the sub-commands that start with **core show**.

```
server*CLI> core show [Tab]
application     applications    calls           channel
channels       channeltype     channeltypes    codec
codecs         config          file            function
functions      help            hint            hints
image          license         profile         settings
switches       sysinfo         taskprocessors  threads
translation    uptime          version         warranty
server*CLI> core show
```

Another trick you can use on the CLI is to cycle through your previous commands. Asterisk stores a history of the commands you type and you can press the **up arrow** key to cycle through the history.

If you type an exclamation mark at the Asterisk CLI, you will get a Linux shell. When you exit the Linux shell (by typing **exit** or pressing **Ctrl+D**), you return to the Asterisk CLI. You can also type an exclamation mark and a Linux command, and the output of that command will be shown to you, and then you'll be returned to the Asterisk CLI.

```
server*CLI> !whoami
root
server*CLI>
```

As you can see, there's a wealth of information available from the Asterisk command-line interface, and we've only scratched the surface. In later sections, we'll go into more details about how to use the command-line interface for other purposes.

Troubleshooting

If you're able to get the command-line examples above working, feel free to skip this section. Otherwise, let's look at troubleshooting connections to the Asterisk CLI.

The most common problem that people encounter when learning the Asterisk command-line interface is that sometimes they're not able to connect to the Asterisk service running in the background. For example, let's say that Fred starts the Asterisk service, but then isn't able to connect to it with the CLI:

```
[root@server ~]# service asterisk start
Starting asterisk:                                       [  OK  ]
[root@server ~]# asterisk -r
Asterisk version, Copyright (C) 1999 - 2010 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
======================================================================
to connect to remote asterisk (does /var/run/asterisk/asterisk.ctl
exist?)
```

What does this mean? It most likely means that Asterisk did not remain running between the time that the service was started and the time Fred tried to connect to the CLI (even if it was only a matter of a few seconds.) This could be caused by a variety of things, but the most common is a broken configuration file.

To diagnose Asterisk start-up problems, we'll start Asterisk in a special mode, known as **console** mode. In this mode, Asterisk does not run as a background service or daemon, but instead runs directly in the console. To start Asterisk in console mode, pass the **-c** parameter to the **asterisk** application. In this case, we also want to turn up the verbosity, so we can see any error messages that might indicate why Asterisk is unable to start.

```
[root@server ~]# asterisk -vvvc
Asterisk version, Copyright (C) 1999 - 2010 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show
warranty' for details.
This is free software, with components licensed under the GNU
General Public
License version 2 and other licenses; you are welcome to
redistribute it under
certain conditions. Type 'core show license' for details.
=====================================================================
== Parsing '/etc/asterisk/asterisk.conf':   == Found
  == Parsing '/etc/asterisk/extconfig.conf':   == Found
  == Parsing '/etc/asterisk/logger.conf':   == Found
  == Parsing '/etc/asterisk/asterisk.conf':   == Found
  Asterisk Dynamic Loader Starting:
  == Parsing '/etc/asterisk/modules.conf':   == Found
...
```

Carefully look for any errors or warnings that are printed to the CLI, and you should have enough information to solve whatever problem is keeping Asterisk from starting up.

> ✅ **Running Asterisk in Console Mode**
> We don't recommend you use Asterisk in console mode on a production system, but simply use it for debugging, especially when debugging start-up problems. On production systems, run Asterisk as a background service.

Asterisk Architecture

From an architectural standpoint, Asterisk is made up of many different modules. This modularity gives you an almost unlimited amount of flexibility in the design of an Asterisk-based system. As an Asterisk administrator, you have the choice on which modules to load. Each module that you loads provides different capabilities to the system. For example, one module might allow your Asterisk system to communicate with analog phone lines, while another might add call reporting capabilities. In this section, we'll discuss the various types of modules and the capabilities they provide.

Asterisk Architecture, The Big Picture

Before we dive too far into the various types of modules, let's first take a step back and look at the overall architecture of Asterisk.
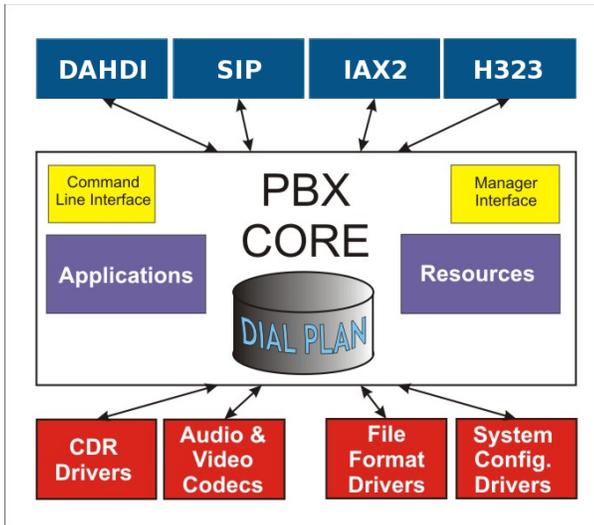
Asterisk Architecture

We need to add CEL and Bridge modules to this picture, and take CLI and Manager out for now

The heart of any Asterisk system is the **core**. The PBX core is the essential component that takes care of bridging calls. The core also takes care of other items like reading the configuration files and loading the other modules. We'll talk more about the core below, but for now just remember that all the other modules connect to it.

From a logistical standpoint, these modules are typically files with a **.so** file extension, which live in the Asterisk modules directory (which is typically **/usr/lib/asterisk/modules**). When Asterisk starts up, it loads these files and adds their functionality to the system.

> ✅ **A Plethora of Modules**
> Take just a minute and go look at the Asterisk modules directory on your system. You should find a wide variety of modules. A typical Asterisk system has over one hundred fifty different modules!

The core also contains the dialplan, which is the logic of any Asterisk system. The dialplan contains a list of instructions that Asterisk should follow to know how to handle incoming and outgoing calls on the system.

Asterisk modules which are part of the core have a file name that look like **pbx_xxxxx.so**.

Types of Asterisk Modules

There are many different types of modules, many of which are shown in the diagram above.

- **Channel Drivers**

At the top of the diagram, we show channel drivers. Channel drivers communicate with devices outside of Asterisk, and translate that particular signaling or protocol to the core.

- **Dialplan Applications**

Applications provide call functionality to the system. An application might answer a call, play a sound prompt, hang up a call, and so forth.

- **Dialplan Functions**

Functions are used to retrieve or set various settings on a call. A function might be used to set the Caller ID on an outbound call, for example.

- **Resources**

As the name suggests, resources provide resources to Asterisk. Common examples of resources include music on hold and call parking.

- **CODECs**

A CODEC (which is an acronym for COder/DECoder) is a module for encoding or decoding audio or video. Typically codecs are used to encode media so that it takes less bandwidth.

- **File Format Drivers**

File format drivers are used to save media to disk in a particular file format, and to convert those files back to media streams on the network.

- **Call Detail Record (CDR) Drivers**

CDR drivers write call logs to a disk or to a database.

- **Call Event Log (CEL) Drivers**

Call event logs are similar to call detail records, but record more detail about what happened inside of Asterisk during a particular call.

- **Bridge Drivers**

Bridge drivers are used by the bridging architecture in Asterisk, and provide various methods of bridging call media between participants in a call.

Now let's go into more detail on each of the module types.

Channel Driver Modules

All calls from the outside come through a channel driver before reaching the core, and all outbound calls go through a channel driver on their way to the external device.

The SIP channel driver, for example, communicates with external devices using the SIP protocol. It translates the SIP signaling into the core. This means that the core of Asterisk is signaling agnostic. Therefore, Asterisk isn't just a SIP PBX, it's a multi-protocol PBX.

For more information on the various channel drivers, see [Section 400. Channel Drivers and External Connectivity].

All channel drivers have a file name that look like **chan_xxxxx.so**, such as **chan_sip.so** or **chan_dahdi.so**.

Dialplan Application Modules

The application modules provide call functionality to the system. These applications are then scripted sequentially in the dialplan. For example, a call might come into Asterisk dialplan, which might use one application to answer the call, another to play back a sound prompt from disk, and a third application to allow the caller to leave voice mail in a particular mailbox.

For more information on dialplan applications, see Dialplan Fundamentals.

All application modules have file names that looks like **app_xxxxx.so**, such as **app_voicemail.so**.

Dialplan Function Modules

Dialplan functions are somewhat similar to dialplan applications, but instead of doing work on a particular channel or call, they simply retrieve or set a particular setting on a channel, or perform text manipulation. For example, a dialplan function might retrieve the Caller ID information from an incoming call, filter some text, or set a timeout for caller input.

For more information on dialplan functions, see [PBX Features].

All dialplan application modules have file names that looks like **func_xxxxx.so**, such as **func_callerid.so**.

Resource Modules

Resources provide functionality to Asterisk that may be called upon at any time during a call, even while another application is running on the channel. Resources are typically used of asynchronous events such as playing hold music when a call gets placed on hold, or performing call parking.

Resource modules have file names that looks like **res_xxxxx.so**, such as **res_musiconhold.so**.

Codec Modules

CODEC modules have file names that look like **codec_xxxxx.so**, such as **codec_alaw.so** and **codec_ulaw.so**.

CODECs represent mathematical algorithms for encoding (compressing) and decoding (decompression) media streams. Asterisk uses CODEC modules to both send and recieve media (audio and video). Asterisk also uses CODEC modules to convert (or transcode) media streams between different formats.

CODEC modules have file names that look like codec_xxxxx.so, such as codec_alaw.so and codec_ulaw.so.

Asterisk is provided with CODEC modules for the following media types:

- ADPCM, 32kbit/s
- G.711 alaw, 64kbit/s
- G.711 ulaw, 64kbit/s
- G.722, 64kbit/s
- G.726, 32kbit/s

- GSM, 13kbit/s
- LPC-10, 2.4kbit/s

If the Speex (www.speex.org) development libraries are detected on your system when Asterisk is built, a CODEC module for Speex will also be installed.

If the iLBC (www.ilbcfreeware.org) development libraries are detected on your system when Asterisk is built, a CODEC module for iLBC will also be installed.

Support for the patent-encumbered G.729A or G.723.1 CODECs is provided by Digium on a commercial basis through both software and hardware products. For more information about purchasing licenses or hardware to use the G.729A or G.723.1 CODECs with Asterisk, please see Digium's website.

Support for Polycom's patent-encumbered but free G.722.1 Siren7 and G.722.1C Siren14 CODECs can be enabled in Asterisk by downloading the binary CODEC modules from Digium's website.

For more detailed information on CODECs, see [CODECs].

File Format Drivers

Add a list of the file formats that Asterisk supports, then point them at the module in section 400 that goes into more detail?
Asterisk uses file format modules to take media (such as audio and video) from the network and save them on disk, or retrieve said files from disk and convert them back to a media stream. While often related to CODECs, there may be more than one available on-disk format for a particular CODEC.

File format modules have file names that look like **format_xxxxx.so**, such as **format_wav.so** and **format_jpeg.so**.

Add a list of the file formats that Asterisk supports, then point them at the module in section 400 that goes into more detail?

Call Detail Record (CDR) Drivers

CDR modules are used to store call detail records in a variety of formats. Popular storage mechanisms include comma-separated value (CSV) files, as well as relational databases such as PostgreSQL. Call detail records typically contain one record per call, and give details such as who made the call, who answered the call, the amount of time spent on the call, and so forth.

For more information on call detail records, see [Section 370. Call Detail Records].

Call detail record modules have file names that look like **cdr_xxxxx.so**, such as **cdr_csv.so** and **cdr_pgsql.so**.

Call Event Log (CEL) Driver Modules

Call Event Logs record the various actions that happen on a call. As such, they are typically more detailed that call detail records. For example, a call event log might show that Alice called Bob,

that Bob's phone rang for twenty seconds, then Bob's mobile phone rang for fifteen seconds, the call then went to Bob's voice mail, where Alice left a twenty-five second voicemail and hung up the call. The system also allows for custom events to be logged as well.

For more information about Call Event Logging, see [Call Event Logging].

Call event logging modules have file names that look like **cel_xxxxx.so**, such as **cel_custom.so** and **cel_adaptive_odbc.so**.

Bridging Modules

Beginning in Asterisk 1.6.2, Asterisk introduced a new method for bridging calls together. It relies on various bridging modules to control how the media streams should be mixed for the participants on a call. The new bridging methods are designed to be more flexible and more efficient than earlier methods.

Bridging modules have file names that look like **bridge_xxxxx.so**, such as **bridge_simple.so** and **bridge_multiplexed.so**.

Call Flow and Bridging Model

Now that you know about the various modules that Asterisk uses, let's talk about the ways that calls flow through an Asterisk system. To explain this clearly, let's say that Alice wants to talk to Bob, and they both have SIP phones connected to their Asterisk system. Let's see what happens!

Should we add a graphic to help explain the call flow model?

1. Alice dials extension 6002, which is Bob's extension on the Asterisk system.
2. A SIP message goes from Alice's phone to the SIP channel driver in Asterisk
3. The SIP channel driver authenticates the call. If Alice's phone does not provide the proper credentials, Asterisk rejects the call.
4. At this point, we have Alice's phone communicating with Asterisk.
5. Now the call goes from the SIP channel driver into the core of Asterisk. Asterisk looks for a set of instructions to follow for extension 6002 in the dialplan.
6. Extension 6002 in the dialplan tells Asterisk to call Bob's phone
7. Asterisk makes a call out through the SIP channel driver to Bob's phone.
8. Bob answers his phone.
9. Now we have two independent calls on the Asterisk system: one from Alice, and to Bob. Asterisk now bridges the audio between these two calls (known as **channels** in Asterisk parlance).
10. When one channel hangs up, Asterisk signals the other channel to hang up.

And there we have it! We've shown how calls flow from external devices, through the channel drivers to the core of Asterisk, and back out through the channel drivers to external devices.

Asterisk on (Open)Solaris

Asterisk on Solaris 10 and OpenSolaris

On this page

- onSolaris:

math library)
several dependencies)
- SUNWflexlex (GNU flex)
- SUNWggrp (GNU grep)
- SUNWgsed (GNU sed)
- SUNWdoxygen (optional; needed for "make progdocs")
- SUNWopenldap (optional; needed for res_config_ldap; see below)
- SUNWgnu-coreutils (optional; provides GNU install; see below)

Caution: installing SUNW gnu packages will change the default application run when the user types 'sed' and 'grep' from /usr/bin/sed to /usr/gnu/bin/sed. Just be aware of this change, as there are differences between the Sun and GNU versions of these utilities.

LDAP dependencies

Because OpenSolaris ships by default with Sun's LDAP libraries, you must install the SUNWopenldap package to provide OpenLDAP libraries. Because of namespace conflicts, the standard LDAP detection will not work.

There are two possible solutions:

1. Port res_config_ldap to use only the RFC-specified API. This should allow it to link against Sun's LDAP libraries.
   - The problem is centered around the use of the OpenLDAP-specific ldap_initialize() call.
2. Change the detection routines in configure to use OpenSolaris' layout of OpenLDAP.
   - This seems doubtful simply because the filesystem layout of SUNWopenldap is so non-standard.

Despite the above two possibilities, there is a workaround to make Asterisk compile with res_config_ldap.

- Modify the "configure" script, changing all instances of "-lldap" to "-lldap-2.4".
  - At the time of this writing there are only 4 instances. This alone will make configure properly detect LDAP availability. But it will not compile.
- When running make, specify the use of the OpenLDAP headers like this:

```
"make LDAP_INCLUDE=-I/usr/include/openldap"
```

Makefile layouts

This has been fixed in Asterisk 1.8 and is no longer an issue.

In Asterisk 1.6 the Makefile overrides any usage of --prefix. I suspect the assumptions are from

is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 United States License.

back before configure provided the ability to set the installation prefix. Regardless, if you are building on OpenSolaris, be aware of this behavior of the Makefile!

If you want to alter the install locations you will need to hand-edit the Makefile. Search for the string "SunOS" to find the following section:

```
# Define standard directories for various platforms
# These apply if they are not redefined in asterisk.conf
ifeq ($(OSARCH),SunOS)
  ASTETCDIR=/etc/asterisk
  ASTLIBDIR=/opt/asterisk/lib
  ASTVARLIBDIR=/var/opt/asterisk
  ASTDBDIR=$(ASTVARLIBDIR)
  ASTKEYDIR=$(ASTVARLIBDIR)
  ASTSPOOLDIR=/var/spool/asterisk
  ASTLOGDIR=/var/log/asterisk
  ASTHEADERDIR=/opt/asterisk/include/asterisk
  ASTBINDIR=/opt/asterisk/bin
  ASTSBINDIR=/opt/asterisk/sbin
  ASTVARRUNDIR=/var/run/asterisk
  ASTMANDIR=/opt/asterisk/man
else
```

Note that, despite the comment, these definitions have build-time and run-time implications. Make sure you make these changes BEFORE you build!

FAX support with SpanDSP

I have been able to get this to work reliably, including T.38 FAX over SIP. If you are running Asterisk 1.6 note Ticket 16342 if you do not install SpanDSP to the default locations (/usr/include and /usr/lib).

There is one build issue with SpanDSP that I need to document (FIXME)

Gotchas

Runtime issues

- WAV and WAV49 files are not written correctly (see Ticket 16610)
- 32-bit binaries on Solaris are limited to 255 file descriptors by default. (see http://developers.sun.com/solaris/articles/stdio_256.html)

Build issues

- bootstrap.sh does not correctly detect OpenSolaris build tools (see Ticket 16341)
- Console documentation is not properly loaded at startup (see Ticket 16688)
- Solaris sed does not properly create AEL parser files (see Ticket 16696; workaround is to install GNU sed with SUNWgsed)
- Asterisk's provided install script, install-sh, is not properly referenced in the makeopts file that is generated during the build. One workaround is to install GNU install from the SUNWgnu-coreutils package. (See Ticket 16781)

Finally, Solaris memory allocation seems far more sensitive than Linux. This has resulted in the discovery of several previously unknown bugs related to uninitialized variables that Linux

handled silently. Note that this means, until these bugs are found and fixed, you may get segfaults.

At the time of this writing I have had a server up and running reasonably stable. However, there are large sections of Asterisk's codebase I do not use and likely contain more of these uninitialized variable problems and associated potential segfaults.

Asterisk Command Reference

This page is the top level page for all of the Asterisk applications, functions, manager actions, and AGI commands that are kept in the XML based documentation that is included with Asterisk.

AGI Commands

AGICommand_ANSWER

ANSWER

Synopsis

Answer channel

Description

Answers channel if not already in answer state. Returns -1 on channel failure, or 0 if successful.

Syntax

```
ANSWER
```

Arguments

See Also

AGICommand_HANGUP

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_ASYNCAGI BREAK

ASYNCAGI BREAK

Synopsis

Interrupts Async AGI

Description

Interrupts expected flow of Async AGI commands and returns control to previous source

(typically, the PBX dialplan).

```
ASYNCAGI BREAK
```

Arguments

See Also

AGICommand_HANGUP

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_CHANNEL STATUS

CHANNEL STATUS

Synopsis

Returns status of the connected channel.

Description

Returns the status of the specified *channelname*. If no channel name is given then returns the status of the current channel.

Return values:

Channel is down and available.

Channel is down, but reserved.

Channel is off hook.

Digits (or equivalent) have been dialed.

Line is ringing.

Remote end is ringing.

Line is up.

Line is busy.

Syntax

```
CHANNEL STATUS [CHANNELNAME]
```

Arguments

- CHANNELNAME

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_CONTROL STREAM FILE

CONTROL STREAM FILE

Synopsis

Sends audio file on channel and allows the listener to control the stream.

Description

Send the given file, allowing playback to be controlled by the given digits, if any. Use double quotes for the digits if you wish none to be permitted. Returns 0 if playback completes without a digit being pressed, or the ASCII numerical value of the digit if one was pressed, or -1 on error or if the channel was disconnected.

Syntax

```
CONTROL STREAM FILE FILENAME ESCAPE_DIGITS [SKIPMS] [FFCHAR]
[REWCHR] [PAUSECHR]
```

Arguments

- FILENAME - The file extension must not be included in the filename.
- ESCAPE_DIGITS
- SKIPMS
- FFCHAR - Defaults to *
- REWCHR - Defaults to #
- PAUSECHR

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_DATABASE DEL

DATABASE DEL

Synopsis

Removes database key/value

Description

Deletes an entry in the Asterisk database for a given *family* and *key*.

Returns `1` if successful, `0` otherwise.

Syntax

```
DATABASE DEL FAMILY KEY
```

Arguments

- `FAMILY`
- `KEY`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_DATABASE DELTREE

DATABASE DELTREE

Synopsis

Removes database keytree/value

Description

Deletes a *family* or specific *keytree* within a *family* in the Asterisk database.

Returns `1` if successful, `0` otherwise.

Syntax

```
DATABASE DELTREE FAMILY [KEYTREE]
```

Arguments

- `FAMILY`
- `KEYTREE`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_DATABASE GET

DATABASE GET

Synopsis

Gets database value

Retrieves an entry in the Asterisk database for a given *family* and *key*.

Returns `0` if *key* is not set. Returns `1` if *key* is set and returns the variable in parenthesis.

Example return code: 200 result=1 (testvariable)

Syntax

```
DATABASE GET FAMILY KEY
```

Arguments

- FAMILY
- KEY

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_DATABASE PUT

DATABASE PUT

Synopsis

Adds/updates database value

Description

Adds or updates an entry in the Asterisk database for a given *family*, *key*, and *value*.

Returns `1` if successful, `0` otherwise.

Syntax

```
DATABASE PUT FAMILY KEY VALUE
```

Arguments

- FAMILY
- KEY
- VALUE

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_EXEC

EXEC

Synopsis

Executes a given Application

Description

Executes *application* with given *options*.

Returns whatever the *application* returns, or -2 on failure to find *application*.

Syntax

```
EXEC APPLICATION OPTIONS
```

Arguments

- APPLICATION
- OPTIONS

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_GET DATA

GET DATA

Synopsis

Prompts for DTMF on a channel

Description

Stream the given *file*, and receive DTMF data.

Returns the digits received from the channel at the other end.

Syntax

```
GET DATA FILE [TIMEOUT] [MAXDIGITS]
```

Arguments

- FILE
- TIMEOUT

- MAXDIGITS

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_GET FULL VARIABLE

GET FULL VARIABLE

Synopsis

Evaluates a channel expression

Description

Returns 0 if *variablename* is not set or channel does not exist. Returns 1 if *variablename* is set and returns the variable in parenthesis. Understands complex variable names and builtin variables, unlike GET VARIABLE.

Example return code: 200 result=1 (testvariable)

Syntax

```
GET FULL VARIABLE VARIABLENAME [CHANNEL_NAME]
```

Arguments

- VARIABLENAME
- CHANNEL_NAME

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_GET OPTION

GET OPTION

Synopsis

Stream file, prompt for DTMF, with timeout.

Description

Behaves similar to STREAM FILE but used with a timeout option.

Syntax

```
GET OPTION FILENAME ESCAPE_DIGITS [TIMEOUT]
```

Arguments

- FILENAME
- ESCAPE_DIGITS
- TIMEOUT

See Also

AGICommand_STREAM FILE

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_GET VARIABLE

GET VARIABLE

Synopsis

Gets a channel variable.

Description

Returns `0` if *variablename* is not set. Returns `1` if *variablename* is set and returns the variable in parentheses.

Example return code: 200 result=1 (testvariable)

Syntax

```
GET VARIABLE VARIABLENAME
```

Arguments

- VARIABLENAME

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_GOSUB

GOSUB

Synopsis

Cause the channel to execute the specified dialplan subroutine.

Description

Cause the channel to execute the specified dialplan subroutine, returning to the dialplan with execution of a Return().

Syntax

```
GOSUB CONTEXT EXTENSION PRIORITY [OPTIONAL-ARGUMENT]
```

Arguments

- CONTEXT
- EXTENSION
- PRIORITY
- OPTIONAL-ARGUMENT

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_HANGUP

HANGUP

Synopsis

Hangup a channel.

Description

Hangs up the specified channel. If no channel name is given, hangs up the current channel

Syntax

```
HANGUP [CHANNELNAME]
```

Arguments

- CHANNELNAME

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r323234.

AGICommand_NOOP

NOOP

Synopsis

Does nothing.

Syntax

```
NOOP
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_RECEIVE CHAR

RECEIVE CHAR

Synopsis

Receives one character from channels supporting it.

Description

Receives a character of text on a channel. Most channels do not support the reception of text. Returns the decimal value of the character if one is received, or `0` if the channel does not support text reception. Returns `-1` only on error/hangup.

Syntax

```
RECEIVE CHAR TIMEOUT
```

Arguments

- `TIMEOUT` - The maximum time to wait for input in milliseconds, or `0` for infinite. Most channels

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_RECEIVE TEXT

RECEIVE TEXT

Synopsis

Receives text from channels supporting it.

Receives a string of text on a channel. Most channels do not support the reception of text. Returns `-1` for failure or `1` for success, and the string in parenthesis.

Syntax

```
RECEIVE TEXT TIMEOUT
```

Arguments

- `TIMEOUT` - The timeout to be the maximum time to wait for input in milliseconds, or `0` for infinite.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_RECORD FILE

RECORD FILE

Synopsis

Records to a given file.

Description

Record to a file until a given dtmf digit in the sequence is received. Returns `-1` on hangup or error. The format will specify what kind of file will be recorded. The *timeout* is the maximum record time in milliseconds, or `-1` for no *timeout*. *offset samples* is optional, and, if provided, will seek to the offset without exceeding the end of the file. *silence* is the number of seconds of silence allowed before the function returns despite the lack of dtmf digits or reaching *timeout*. *silence* value must be preceded by `s=` and is also optional.

Syntax

```
RECORD FILE FILENAME FORMAT ESCAPE_DIGITS TIMEOUT [OFFSET_SAMPLES]
[BEEP] [S=SILENCE]
```

Arguments

- `FILENAME`
- `FORMAT`
- `ESCAPE_DIGITS`
- `TIMEOUT`
- `OFFSET_SAMPLES`
- `BEEP`
- `S=SILENCE`

See Also

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SAY ALPHA

SAY ALPHA

Synopsis

Says a given character string.

Description

Say a given character string, returning early if any of the given DTMF digits are received on the channel. Returns `0` if playback completes without a digit being pressed, or the ASCII numerical value of the digit if one was pressed or `-1` on error/hangup.

Syntax

```
SAY ALPHA NUMBER ESCAPE_DIGITS
```

Arguments

- `NUMBER`
- `ESCAPE_DIGITS`

See Also

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SAY DATE

SAY DATE

Synopsis

Says a given date.

Description

Say a given date, returning early if any of the given DTMF digits are received on the channel. Returns `0` if playback completes without a digit being pressed, or the ASCII numerical value of the digit if one was pressed or `-1` on error/hangup.

Syntax

```
SAY DATE DATE ESCAPE_DIGITS
```

Arguments

- `DATE` - Is number of seconds elapsed since 00:00:00 on January 1, 1970. Coordinated Universal Time (UTC).
- `ESCAPE_DIGITS`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SAY DATETIME

SAY DATETIME

Synopsis

Says a given time as specified by the format given.

Description

Say a given time, returning early if any of the given DTMF digits are received on the channel. Returns `0` if playback completes without a digit being pressed, or the ASCII numerical value of the digit if one was pressed or `-1` on error/hangup.

Syntax

```
SAY DATETIME TIME ESCAPE_DIGITS [FORMAT] [TIMEZONE]
```

Arguments

- `TIME` - Is number of seconds elapsed since 00:00:00 on January 1, 1970, Coordinated Universal Time (UTC)
- `ESCAPE_DIGITS`
- `FORMAT` - Is the format the time should be said in. See `voicemail.conf` (defaults to `ABdY 'digits/at' IMp`).
- `TIMEZONE` - Acceptable values can be found in `/usr/share/zoneinfo` Defaults to machine default.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SAY DIGITS

SAY DIGITS

Synopsis

Says a given digit string.

Description

Say a given digit string, returning early if any of the given DTMF digits are received on the channel. Returns `0` if playback completes without a digit being pressed, or the ASCII numerical value of the digit if one was pressed or `-1` on error/hangup.

```
SAY DIGITS NUMBER ESCAPE_DIGITS
```

Arguments

- NUMBER
- ESCAPE_DIGITS

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SAY NUMBER

SAY NUMBER

Synopsis

Says a given number.

Description

Say a given number, returning early if any of the given DTMF digits are received on the channel. Returns 0 if playback completes without a digit being pressed, or the ASCII numerical value of the digit if one was pressed or -1 on error/hangup.

Syntax

```
SAY NUMBER NUMBER ESCAPE_DIGITS [GENDER]
```

Arguments

- NUMBER
- ESCAPE_DIGITS
- GENDER

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SAY PHONETIC

SAY PHONETIC

Synopsis

Says a given character string with phonetics.

Description

Say a given character string with phonetics, returning early if any of the given DTMF digits are received on the channel. Returns `0` if playback completes without a digit pressed, the ASCII numerical value of the digit if one was pressed, or `-1` on error/hangup.

Syntax

```
SAY PHONETIC STRING ESCAPE_DIGITS
```

Arguments

- `STRING`
- `ESCAPE_DIGITS`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SAY TIME

SAY TIME

Synopsis

Says a given time.

Description

Say a given time, returning early if any of the given DTMF digits are received on the channel. Returns `0` if playback completes without a digit being pressed, or the ASCII numerical value of the digit if one was pressed or `-1` on error/hangup.

Syntax

```
SAY TIME TIME ESCAPE_DIGITS
```

Arguments

- `TIME` - Is number of seconds elapsed since 00:00:00 on January 1, 1970. Coordinated Universal Time (UTC).
- `ESCAPE_DIGITS`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SEND IMAGE

SEND IMAGE

Sends images to channels supporting it.

Description

Sends the given image on a channel. Most channels do not support the transmission of images. Returns `0` if image is sent, or if the channel does not support image transmission. Returns `-1` only on error/hangup. Image names should not include extensions.

Syntax

```
SEND IMAGE IMAGE
```

Arguments

- `IMAGE`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SEND TEXT

SEND TEXT

Synopsis

Sends text to channels supporting it.

Description

Sends the given text on a channel. Most channels do not support the transmission of text. Returns `0` if text is sent, or if the channel does not support text transmission. Returns `-1` only on error/hangup.

Syntax

```
SEND TEXT TEXT_TO_SEND
```

Arguments

- `TEXT_TO_SEND` - Text consisting of greater than one word should be placed in quotes since the command only accepts a single argument.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SET AUTOHANGUP

SET AUTOHANGUP

Synopsis

Autohangup channel in some time.

Description

Cause the channel to automatically hangup at *time* seconds in the future. Of course it can be hungup before then as well. Setting to 0 will cause the autohangup feature to be disabled on this channel.

Syntax

```
SET AUTOHANGUP TIME
```

Arguments

- TIME

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SET CALLERID

SET CALLERID

Synopsis

Sets callerid for the current channel.

Description

Changes the callerid of the current channel.

Syntax

```
SET CALLERID NUMBER
```

Arguments

- NUMBER

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SET CONTEXT

SET CONTEXT

Synopsis

Sets channel context.

Description

Sets the context for continuation upon exiting the application.

Syntax

```
SET CONTEXT DESIRED_CONTEXT
```

Arguments

- DESIRED_CONTEXT

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SET EXTENSION

SET EXTENSION

Synopsis

Changes channel extension.

Description

Changes the extension for continuation upon exiting the application.

Syntax

```
SET EXTENSION NEW_EXTENSION
```

Arguments

- NEW_EXTENSION

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SET MUSIC

Synopsis

Enable/Disable Music on hold generator

Description

Enables/Disables the music on hold generator. If *class* is not specified, then the `default` music on hold class will be used.

Always returns `0`.

Syntax

```
SET MUSIC UNNAMED_PARAMETER CLASS
```

Arguments

- UNNAMED_PARAMETER
    - Unnamed Option
    - Unnamed Option
- CLASS

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SET PRIORITY

SET PRIORITY

Synopsis

Set channel dialplan priority.

Description

Changes the priority for continuation upon exiting the application. The priority must be a valid priority or label.

Syntax

```
SET PRIORITY PRIORITY
```

Arguments

- PRIORITY

See Also

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SET VARIABLE

SET VARIABLE

Synopsis

Sets a channel variable.

Description

Sets a variable to the current channel.

Syntax

```
SET VARIABLE VARIABLENAME VALUE
```

Arguments

- VARIABLENAME
- VALUE

See Also

AGICommand_SPEECH ACTIVATE GRAMMAR

SPEECH ACTIVATE GRAMMAR

Synopsis

Activates a grammar.

Description

Activates the specified grammar on the speech object.

Syntax

```
SPEECH ACTIVATE GRAMMAR GRAMMAR_NAME
```

Arguments

- GRAMMAR_NAME

See Also

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SPEECH CREATE

SPEECH CREATE

Synopsis

Creates a speech object.

Description

Create a speech object to be used by the other Speech AGI commands.

Syntax

```
SPEECH CREATE ENGINE
```

Arguments

- ENGINE

See Also

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SPEECH DEACTIVATE GRAMMAR

SPEECH DEACTIVATE GRAMMAR

Synopsis

Deactivates a grammar.

Description

Deactivates the specified grammar on the speech object.

Syntax

```
SPEECH DEACTIVATE GRAMMAR GRAMMAR_NAME
```

Arguments

- GRAMMAR_NAME

See Also

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SPEECH DESTROY

SPEECH DESTROY

Synopsis

Destroys a speech object.

Description

Destroy the speech object created by SPEECH CREATE.

Syntax

```
SPEECH DESTROY
```

Arguments

See Also

AGICommand_SPEECH CREATE

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SPEECH LOAD GRAMMAR

SPEECH LOAD GRAMMAR

Synopsis

Loads a grammar.

Description

Loads the specified grammar as the specified name.

Syntax

```
SPEECH LOAD GRAMMAR GRAMMAR_NAME PATH_TO_GRAMMAR
```

Arguments

- GRAMMAR_NAME
- PATH_TO_GRAMMAR

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SPEECH RECOGNIZE

SPEECH RECOGNIZE

Synopsis

Recognizes speech.

Description

Plays back given *prompt* while listening for speech and dtmf.

Syntax

```
SPEECH RECOGNIZE PROMPT TIMEOUT [OFFSET]
```

Arguments

- `PROMPT`
- `TIMEOUT`
- `OFFSET`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SPEECH SET

SPEECH SET

Synopsis

Sets a speech engine setting.

Description

Set an engine-specific setting.

Syntax

```
SPEECH SET NAME VALUE
```

Arguments

- `SPEECH SET`
- `VALUE`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_SPEECH UNLOAD GRAMMAR

SPEECH UNLOAD GRAMMAR

Synopsis

Unloads a grammar.

Description

Unloads the specified grammar.

Syntax

```
SPEECH UNLOAD GRAMMAR GRAMMAR_NAME
```

Arguments

- GRAMMAR_NAME

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_STREAM FILE

STREAM FILE

Synopsis

Sends audio file on channel.

Description

Send the given file, allowing playback to be interrupted by the given digits, if any. Returns `0` if playback completes without a digit being pressed, or the ASCII numerical value of the digit if one was pressed, or `-1` on error or if the channel was disconnected.

Syntax

```
STREAM FILE FILENAME ESCAPE_DIGITS [SAMPLE_OFFSET]
```

Arguments

- FILENAME - File name to play. The file extension must not be included in the *filename*.
- ESCAPE_DIGITS - Use double quotes for the digits if you wish none to be permitted.
- SAMPLE_OFFSET - If sample offset is provided then the audio will seek to sample offset before play starts.

AGICommand_CONTROL STREAM FILE

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_TDD MODE

TDD MODE

Synopsis

Toggles TDD mode (for the deaf).

Description

Enable/Disable TDD transmission/reception on a channel. Returns `1` if successful, or `0` if channel is not TDD-capable.

Syntax

```
TDD MODE BOOLEAN
```

Arguments

- BOOLEAN
    - on
    - off

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_VERBOSE

VERBOSE

Synopsis

Logs a message to the asterisk verbose log.

Description

Sends *message* to the console via verbose message system. *level* is the verbose level (1-4). Always returns `1`

Syntax

```
VERBOSE MESSAGE LEVEL
```

Arguments

- `MESSAGE`
- `LEVEL`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGICommand_WAIT FOR DIGIT

WAIT FOR DIGIT

Synopsis

Waits for a digit to be pressed.

Description

Waits up to *timeout* milliseconds for channel to receive a DTMF digit. Returns -1 on channel failure, 0 if no digit is received in the timeout, or the numerical value of the ascii of the digit if one is received. Use -1 for the *timeout* value if you desire the call to block indefinitely.

Syntax

```
WAIT FOR DIGIT TIMEOUT
```

Arguments

- `TIMEOUT`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

AGI Command Template Page

AGI COMMAND

Synopsys

…..

Description

…

Syntax

- `AGI COMMAND <arg>`

Arguments

- arg
- something
- options
  - a
    - option 'a' is asdfadf
  - b
    - option 'b' is asdfasdfadf
  - c
    - option 'c' is for cookie

Runs Dead

Yes / No

See Also

[Dialplan Function Template Page](#)
[Dialplan Application Template Page](#)
[AMI Action Template Page](#)

Import Version

This documentation was imported from Asterisk version `VERSION STRING HERE`.

AMI Actions

AMI Action Template Page

ManagerAction

Synopsys

.....

Description

...

Syntax

```
Action: ManagerAction
RequiredHeader: Value
[OptionalHeader:] Value
```

Arguments

- RequiredHeader
  - This header is something that is required.
- OptionalHeader

- This is some optional header

See Also

[Dialplan Application Template Page](#)
[Dialplan Function Template Page](#)
[AGI Command Template Page](#)

Import Version

This documentation was imported from Asterisk version `VERSION STRING HERE`.

ManagerAction_AbsoluteTimeout

AbsoluteTimeout

Synopsis

Set absolute timeout.

Description

Hangup a channel after a certain time. Acknowledges set time with `Timeout Set` message.

Syntax

```
Action: AbsoluteTimeout
[ActionID:] <value>
Channel: <value>
Timeout: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Channel name to hangup.
- `Timeout` - Maximum duration of the call (sec).

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_AgentLogoff

AgentLogoff

Synopsis

Sets an agent as no longer logged in.

Description

Sets an agent as no longer logged in.

```
Action: AgentLogoff
[ActionID:] <value>
Agent: <value>
[Soft:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Agent` - Agent ID of the agent to log off.
- `Soft` - Set to `true` to not hangup existing calls.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Agents

Agents

Synopsis

Lists agents and their status.

Description

Will list info about all possible agents.

Syntax

```
Action: Agents
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_AGI

AGI

Synopsis

Add an AGI command to execute by Async AGI.

Add an AGI command to the execute queue of the channel in Async AGI.

Syntax

```
Action: AGI
[ActionID:] <value>
Channel: <value>
Command: <value>
[CommandID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Channel that is currently in Async AGI.
- `Command` - Application to execute.
- `CommandID` - This will be sent back in CommandID header of AsyncAGI exec event notification.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_AOCMessage

AOCMessage

Synopsis

Generate an Advice of Charge message on a channel.

Description

Generates an AOC-D or AOC-E message on a channel.

Syntax

```
Action: AOCMessage
[ActionID:] <value>
Channel: <value>
[ChannelPrefix:] <value>
MsgType: <value>
ChargeType: <value>
[UnitAmount(0):] <value>
[UnitType(0):] <value>
[CurrencyName:] <value>
[CurrencyAmount:] <value>
[CurrencyMultiplier:] <value>
[TotalType:] <value>
[AOCBillingId:] <value>
[ChargingAssociationId:] <value>
[ChargingAssociationNumber:] <value>
[ChargingAssociationPlan:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Channel name to generate the AOC message on.
- `ChannelPrefix` - Partial channel prefix. By using this option one can match the beginning part of a channel name without having to put the entire name in. For example if a channel name is SIP/snom-00000001 and this value is set to SIP/snom, then that channel matches and the message will be sent. Note however that only the first matched channel has the message sent on it.
- `MsgType` - Defines what type of AOC message to create, AOC-D or AOC-E
    - `D`
    - `E`
- `ChargeType` - Defines what kind of charge this message represents.
    - `NA`
    - `FREE`
    - `Currency`
    - `Unit`
- `UnitAmount(0)` - This represents the amount of units charged. The ETSI AOC standard specifies that this value along with the optional UnitType value are entries in a list. To accommodate this these values take an index value starting at 0 which can be used to generate this list of unit entries. For Example, If two unit entires were required this could be achieved by setting the paramter UnitAmount(0)=1234 and UnitAmount(1)=5678. Note that UnitAmount at index 0 is required when ChargeType=Unit, all other entries in the list are optional.
- `UnitType(0)` - Defines the type of unit. ETSI AOC standard specifies this as an integer value between 1 and 16, but this value is left open to accept any positive integer. Like the UnitAmount parameter, this value represents a list entry and has an index parameter that starts at 0.
- `CurrencyName` - Specifies the currency's name. Note that this value is truncated after 10 characters.
- `CurrencyAmount` - Specifies the charge unit amount as a positive integer. This value is required when ChargeType==Currency.
- `CurrencyMultiplier` - Specifies the currency multiplier. This value is required when ChargeType==Currency.
    - `OneThousandth`
    - `OneHundredth`
    - `OneTenth`
    - `One`
    - `Ten`
    - `Hundred`
    - `Thousand`
- `TotalType` - Defines what kind of AOC-D total is represented.
    - `Total`
    - `SubTotal`
- `AOCBillingId` - Represents a billing ID associated with an AOC-D or AOC-E message. Note that only the first 3 items of the enum are valid AOC-D billing IDs
    - `Normal`
    - `ReverseCharge`
    - `CreditCard`
    - `CallFwdUnconditional`
    - `CallFwdBusy`
    - `CallFwdNoReply`
    - `CallDeflection`

- CallTransfer
- `ChargingAssociationId` - Charging association identifier. This is optional for AOC-E and can be set to any value between -32768 and 32767
- `ChargingAssociationNumber` - Represents the charging association party number. This value is optional for AOC-E.
- `ChargingAssociationPlan` - Integer representing the charging plan associated with the ChargingAssociationNumber. The value is bits 7 through 1 of the Q.931 octet containing the type-of-number and numbering-plan-identification fields.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Atxfer

Atxfer

Synopsis

Attended transfer.

Description

Attended transfer.

Syntax

```
Action: Atxfer
[ActionID:] <value>
Channel: <value>
Exten: <value>
Context: <value>
Priority: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Transferer's channel.
- `Exten` - Extension to transfer to.
- `Context` - Context to transfer to.
- `Priority` - Priority to transfer to.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Bridge

Bridge

Synopsis

Bridge two channels already in the PBX.

Description

Bridge together two channels already in the PBX.

Syntax

```
Action: Bridge
[ActionID:] <value>
Channel1: <value>
Channel2: <value>
[Tone:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel1` - Channel to Bridge to Channel2.
- `Channel2` - Channel to Bridge to Channel1.
- `Tone` - Play courtesy tone to Channel 2.
  - `yes`
  - `no`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Challenge

Challenge

Synopsis

Generate Challenge for MD5 Auth.

Description

Generate a challenge for MD5 authentication.

Syntax

```
Action: Challenge
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_ChangeMonitor

ChangeMonitor

Synopsis

Change monitoring filename of a channel.

Description

This action may be used to change the file started by a previous 'Monitor' action.

Syntax

```
Action: ChangeMonitor
[ActionID:] <value>
Channel: <value>
File: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Used to specify the channel to record.
- `File` - Is the new name of the file created in the monitor spool directory.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Command

Command

Synopsis

Execute Asterisk CLI Command.

Description

Run a CLI command.

Syntax

```
Action: Command
[ActionID:] <value>
Command: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

- `Command` - Asterisk CLI command to run.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_CoreSettings

CoreSettings

Synopsis

Show PBX core settings (version etc).

Description

Query for Core PBX settings.

Syntax

```
Action: CoreSettings
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_CoreShowChannels

CoreShowChannels

Synopsis

List currently active channels.

Description

List currently defined channels and some information about them.

Syntax

```
Action: CoreShowChannels
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_CoreStatus

CoreStatus

Synopsis

Show PBX core status variables.

Description

Query for Core PBX status.

Syntax

```
Action: CoreStatus
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_CreateConfig

CreateConfig

Synopsis

Creates an empty file in the configuration directory.

Description

This action will create an empty file in the configuration directory. This action is intended to be used before an UpdateConfig action.

Syntax

```
Action: CreateConfig
[ActionID:] <value>
Filename: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Filename` - The configuration filename to create (e.g. `foo.conf`).

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DAHDIDialOffhook

DAHDIDialOffhook

Synopsis

Dial over DAHDI channel while offhook.

Description

Syntax

```
Action: DAHDIDialOffhook
[ActionID:] <value>
DAHDIChannel: <value>
Number: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `DAHDIChannel`
- `Number`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DAHDIDNDoff

DAHDIDNDoff

Synopsis

Toggle DAHDI channel Do Not Disturb status OFF.

Description

```
Action: DAHDIDNDoff
[ActionID:] <value>
DAHDIChannel: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `DAHDIChannel`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DAHDIDNDon

DAHDIDNDon

Synopsis

Toggle DAHDI channel Do Not Disturb status ON.

Description

Syntax

```
Action: DAHDIDNDon
[ActionID:] <value>
DAHDIChannel: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `DAHDIChannel`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DAHDIHangup

DAHDIHangup

Synopsis

Hangup DAHDI Channel.

Hangup a DAHDI channel.

```
Action: DAHDIHangup
[ActionID:] <value>
DAHDIChannel: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `DAHDIChannel` - DAHDI channel name to hangup.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DAHDIRestart

DAHDIRestart

Fully Restart DAHDI channels (terminates calls).

```
Action: DAHDIRestart
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DAHDIShowChannels

DAHDIShowChannels

Show status DAHDI channels.

Description

Syntax

```
Action: DAHDIShowChannels
[ActionID:] <value>
[DAHDIChannel:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `DAHDIChannel` - Specify the specific channel to show. Show all channels if zero or not present.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r314732.

ManagerAction_DAHDITransfer

DAHDITransfer

Synopsis

Transfer DAHDI Channel.

Description

Transfer a DAHDI channel.

Syntax

```
Action: DAHDITransfer
[ActionID:] <value>
DAHDIChannel: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `DAHDIChannel` - DAHDI channel name to transfer.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DataGet

DataGet

Synopsis

Retrieve the data api tree.

Retrieve the data api tree.

```
Action: DataGet
[ActionID:] <value>
Path: <value>
[Search:] <value>
[Filter:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Path`
- `Search`
- `Filter`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DBDel

DBDel

Delete DB entry.

```
Action: DBDel
[ActionID:] <value>
Family: <value>
Key: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Family`
- `Key`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DBDelTree

DBDelTree

Synopsis

Delete DB Tree.

Description

Syntax

```
Action: DBDelTree
[ActionID:] <value>
Family: <value>
[Key:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Family`
- `Key`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DBGet

DBGet

Synopsis

Get DB Entry.

Description

Syntax

```
Action: DBGet
[ActionID:] <value>
Family: <value>
Key: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Family`
- `Key`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_DBPut

DBPut

Synopsis

Put DB entry.

Description

Syntax

```
Action: DBPut
[ActionID:] <value>
Family: <value>
Key: <value>
[Val:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Family`
- `Key`
- `Val`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Events

Events

Synopsis

Control Event Flow.

Description

Enable/Disable sending of events to this manager client.

Syntax

```
Action: Events
[ActionID:] <value>
EventMask: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `EventMask`
    - `on` - If all events should be sent.
    - `off` - If no events should be sent.
    - `system,call,log,...` - To select which flags events should have to be sent.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_ExtensionState

ExtensionState

Synopsis

Check Extension Status.

Description

Report the extension state for given extension. If the extension has a hint, will use devicestate to check the status of the device connected to the extension.

Will return an `Extension Status` message. The response will include the hint for the extension and the status.

Syntax

```
Action: ExtensionState
[ActionID:] <value>
Exten: <value>
Context: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Exten` - Extension to check state on.
- `Context` - Context for extension.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_GetConfig

GetConfig

Synopsis

Retrieve configuration.

Description

This action will dump the contents of a configuration file by category and contents or optionally by specified category only.

Syntax

```
Action: GetConfig
[ActionID:] <value>
Filename: <value>
[Category:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Filename` - Configuration filename (e.g. `foo.conf`).
- `Category` - Category in configuration file.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_GetConfigJSON

GetConfigJSON

Synopsis

Retrieve configuration (JSON format).

Description

This action will dump the contents of a configuration file by category and contents in JSON format. This only makes sense to be used using rawman over the HTTP interface.

Syntax

```
Action: GetConfigJSON
[ActionID:] <value>
Filename: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Filename` - Configuration filename (e.g. `foo.conf`).

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Getvar

Getvar

Synopsis

Gets a channel variable.

Description

Get the value of a global or local channel variable.

Syntax

```
Action: Getvar
[ActionID:] <value>
[Channel:] <value>
Variable: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Channel to read variable from.
- `Variable` - Variable name.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Hangup

Hangup

Synopsis

Hangup channel.

Description

Hangup a channel.

Syntax

```
Action: Hangup
[ActionID:] <value>
Channel: <value>
[Cause:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - The channel name to be hangup.
- `Cause` - Numeric hangup cause.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_IAXnetstats

IAXnetstats

Synopsis

Show IAX Netstats.

Description

Show IAX channels network statistics.

Syntax

```
Action: IAXnetstats
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_IAXpeerlist

IAXpeerlist

Synopsis

List IAX Peers.

Description

List all the IAX peers.

```
Action: IAXpeerlist
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_IAXpeers

IAXpeers

Synopsis

List IAX peers.

Description

Syntax

```
Action: IAXpeers
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_IAXregistry

IAXregistry

Synopsis

Show IAX registrations.

Description

Show IAX registrations.

```
Action: IAXregistry
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_JabberSend

JabberSend

Synopsis

Sends a message to a Jabber Client.

Description

Sends a message to a Jabber Client.

Syntax

```
Action: JabberSend
[ActionID:] <value>
Jabber: <value>
JID: <value>
Message: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Jabber` - Client or transport Asterisk uses to connect to JABBER.
- `JID` - XMPP/Jabber JID (Name) of recipient.
- `Message` - Message to be sent to the buddy.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_ListCategories

ListCategories

Synopsis

List categories in configuration file.

This action will dump the categories in a given file.

Syntax

```
Action: ListCategories
[ActionID:] <value>
Filename: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Filename` - Configuration filename (e.g. `foo.conf`).

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_ListCommands

ListCommands

Synopsis

List available manager commands.

Description

Returns the action name and synopsis for every action that is available to the user.

Syntax

```
Action: ListCommands
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_LocalOptimizeAway

LocalOptimizeAway

Optimize away a local channel when possible.

Description

A local channel created with "/n" will not automatically optimize away. Calling this command on the local channel will clear that flag and allow it to optimize away if it's bridged or when it becomes bridged.

Syntax

```
Action: LocalOptimizeAway
[ActionID:] <value>
Channel: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - The channel name to optimize away.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Login

Login

Synopsis

Login Manager.

Description

Login Manager.

Syntax

```
Action: Login
[ActionID:] <value>
Username: <value>
[Secret:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Username` - Username to login with as specified in manager.conf.
- `Secret` - Secret to login with as specified in manager.conf.

See Also

ManagerAction_Logoff

Logoff

Synopsis

Logoff Manager.

Description

Logoff the current manager session.

Syntax

```
Action: Logoff
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

ManagerAction_MailboxCount

MailboxCount

Synopsis

Check Mailbox Message Count.

Description

Checks a voicemail account for new messages.

Returns number of urgent, new and old messages.

Message: Mailbox Message Count

Mailbox: *mailboxid*

UrgentMessages: *count*

NewMessages: *count*

OldMessages: *count*

```
Action: MailboxCount
[ActionID:] <value>
Mailbox: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Mailbox` - Full mailbox ID *mailbox @ vm-context*.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_MailboxStatus

MailboxStatus

Synopsis

Check mailbox.

Description

Checks a voicemail account for status.

Returns number of messages.

Message: Mailbox Status.

Mailbox: *mailboxid.*

Waiting: *count.*

Syntax

```
Action: MailboxStatus
[ActionID:] <value>
Mailbox: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Mailbox` - Full mailbox ID *mailbox @ vm-context*.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_MeetmeList

MeetmeList

Synopsis

List participants in a conference.

Description

Lists all users in a particular MeetMe conference. MeetmeList will follow as separate events, followed by a final event called MeetmeListComplete.

Syntax

```
Action: MeetmeList
[ActionID:] <value>
[Conference:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Conference` - Conference number.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_MeetmeMute

MeetmeMute

Synopsis

Mute a Meetme user.

Description

Syntax

```
Action: MeetmeMute
[ActionID:] <value>
Meetme: <value>
Usernum: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Meetme`
- `Usernum`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_MeetmeUnmute

MeetmeUnmute

Synopsis

Unmute a Meetme user.

Description

Syntax

```
Action: MeetmeUnmute
[ActionID:] <value>
Meetme: <value>
Usernum: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Meetme`
- `Usernum`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_MixMonitorMute

MixMonitorMute

Synopsis

Mute / unMute a Mixmonitor recording.

This action may be used to mute a MixMonitor recording.

```
Action: MixMonitorMute
[ActionID:] <value>
Channel: <value>
[Direction:] <value>
[State:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Used to specify the channel to mute.
- `Direction` - Which part of the recording to mute: read, write or both (from channel, to channel or both channels).
- `State` - Turn mute on or off : 1 to turn on, 0 to turn off.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_ModuleCheck

ModuleCheck

Check if module is loaded.

Checks if Asterisk module is loaded. Will return Success/Failure. For success returns, the module revision number is included.

```
Action: ModuleCheck
Module: <value>
```

Arguments

- `Module` - Asterisk module name (not including extension).

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_ModuleLoad

ModuleLoad

Module management.

Loads, unloads or reloads an Asterisk module in a running system.

```
Action: ModuleLoad
[ActionID:] <value>
[Module:] <value>
LoadType: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Module` - Asterisk module name (including.so extension) or subsystem identifier:
    - `cdr`
    - `enum`
    - `dnsmgr`
    - `extconfig`
    - `manager`
    - `rtp`
    - `http`
- `LoadType` - The operation to be done on module. If no module is specified for a `reload` loadtype, all modules are reloaded.
    - `load`
    - `unload`
    - `reload`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Monitor

Monitor

Monitor a channel.

This action may be used to record the audio on a specified channel.

```
Action: Monitor
[ActionID:] <value>
Channel: <value>
[File:] <value>
[Format:] <value>
[Mix:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Used to specify the channel to record.
- `File` - Is the name of the file created in the monitor spool directory. Defaults to the same name as the channel (with slashes replaced with dashes).
- `Format` - Is the audio recording format. Defaults to `wav`.
- `Mix` - Boolean parameter as to whether to mix the input and output channels together after the recording is finished.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Originate

Originate

Synopsis

Originate a call.

Description

Generates an outgoing call to a *Extension* / *Context* / *Priority* or *Application* / *Data*

Syntax

```
Action: Originate
[ActionID:] <value>
Channel: <value>
[Exten:] <value>
[Context:] <value>
[Priority:] <value>
[Application:] <value>
[Data:] <value>
[Timeout:] <value>
[CallerID:] <value>
[Variable:] <value>
[Account:] <value>
[Async:] <value>
[Codecs:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Channel name to call.
- `Exten` - Extension to use (requires `Context` and `Priority`)
- `Context` - Context to use (requires `Exten` and `Priority`)
- `Priority` - Priority to use (requires `Exten` and `Context`)
- `Application` - Application to execute.
- `Data` - Data to use (requires `Application`).
- `Timeout` - How long to wait for call to be answered (in ms.).
- `CallerID` - Caller ID to be set on the outgoing channel.
- `Variable` - Channel variable to set, multiple Variable: headers are allowed.
- `Account` - Account code.
- `Async` - Set to `true` for fast origination.
- `Codecs` - Comma-separated list of codecs to use for this call.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Park

Park

Synopsis

Park a channel.

Description

Park a channel.

Syntax

```
Action: Park
[ActionID:] <value>
Channel: <value>
Channel2: <value>
[Timeout:] <value>
[Parkinglot:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Channel name to park.
- `Channel2` - Channel to announce park info to (and return to if timeout).
- `Timeout` - Number of milliseconds to wait before callback.
- `Parkinglot` - Parking lot to park channel in.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_ParkedCalls

ParkedCalls

Synopsis

List parked calls.

Description

List parked calls.

Syntax

```
Action: ParkedCalls
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_PauseMonitor

PauseMonitor

Synopsis

Pause monitoring of a channel.

Description

This action may be used to temporarily stop the recording of a channel.

Syntax

```
Action: PauseMonitor
[ActionID:] <value>
Channel: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Used to specify the channel to record.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Ping

Ping

Synopsis

Keepalive command.

Description

A 'Ping' action will ellicit a 'Pong' response. Used to keep the manager connection open.

Syntax

```
Action: Ping
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_PlayDTMF

PlayDTMF

Play DTMF signal on a specific channel.

Description

Plays a dtmf digit on the specified channel.

Syntax

```
Action: PlayDTMF
[ActionID:] <value>
Channel: <value>
Digit: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Channel name to send digit to.
- `Digit` - The DTMF digit to play.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_QueueAdd

QueueAdd

Synopsis

Add interface to queue.

Description

Syntax

```
Action: QueueAdd
[ActionID:] <value>
Queue: <value>
Interface: <value>
[Penalty:] <value>
[Paused:] <value>
[MemberName:] <value>
[StateInterface:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Queue`
- `Interface`
- `Penalty`
- `Paused`
- `MemberName`
- `StateInterface`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_QueueLog

QueueLog

Synopsis

Adds custom entry in queue_log.

Description

Syntax

```
Action: QueueLog
[ActionID:] <value>
Queue: <value>
Event: <value>
[Uniqueid:] <value>
[Interface:] <value>
[Message:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Queue`
- `Event`
- `Uniqueid`
- `Interface`
- `Message`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_QueuePause

QueuePause

Synopsis

Makes a queue member temporarily unavailable.

Description

Syntax

```
Action: QueuePause
[ActionID:] <value>
Interface: <value>
Paused: <value>
[Queue:] <value>
[Reason:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Interface`
- `Paused`
- `Queue`
- `Reason`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_QueuePenalty

QueuePenalty

Synopsis

Set the penalty for a queue member.

Description

Syntax

```
Action: QueuePenalty
[ActionID:] <value>
Interface: <value>
Penalty: <value>
[Queue:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Interface`
- `Penalty`
- `Queue`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_QueueReload

QueueReload

Synopsis

Reload a queue, queues, or any sub-section of a queue or queues.

Description

Syntax

```
Action: QueueReload
[ActionID:] <value>
[Queue:] <value>
[Members:] <value>
[Rules:] <value>
[Parameters:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Queue`
- `Members`
    - yes
    - no
- `Rules`
    - yes
    - no
- `Parameters`
    - yes
    - no

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_QueueRemove

QueueRemove

Synopsis

Remove interface from queue.

Description

Syntax

```
Action: QueueRemove
[ActionID:] <value>
Queue: <value>
Interface: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Queue`
- `Interface`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_QueueReset

QueueReset

Synopsis

Reset queue statistics.

Description

Syntax

```
Action: QueueReset
[ActionID:] <value>
[Queue:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Queue`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_QueueRule

QueueRule

Synopsis

Queue Rules.

Description

```
Action: QueueRule
[ActionID:] <value>
[Rule:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Rule`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Queues

Queues.

```
Action: Queues
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_QueueStatus

Show queue status.

```
Action: QueueStatus
[ActionID:] <value>
[Queue:] <value>
[Member:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Queue`
- `Member`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_QueueSummary

QueueSummary

Synopsis

Show queue summary.

Description

Syntax

```
Action: QueueSummary
[ActionID:] <value>
[Queue:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Queue`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Redirect

Redirect

Synopsis

Redirect (transfer) a call.

Description

Redirect (transfer) a call.

```
Action: Redirect
[ActionID:] <value>
Channel: <value>
[ExtraChannel:] <value>
Exten: <value>
[ExtraExten:] <value>
Context: <value>
[ExtraContext:] <value>
Priority: <value>
[ExtraPriority:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Channel to redirect.
- `ExtraChannel` - Second call leg to transfer (optional).
- `Exten` - Extension to transfer to.
- `ExtraExten` - Extension to transfer extrachannel to (optional).
- `Context` - Context to transfer to.
- `ExtraContext` - Context to transfer extrachannel to (optional).
- `Priority` - Priority to transfer to.
- `ExtraPriority` - Priority to transfer extrachannel to (optional).

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Reload

Reload

Synopsis

Send a reload event.

Description

Send a reload event.

Syntax

```
Action: Reload
[ActionID:] <value>
[Module:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Module` - Name of the module to reload.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_SendText

SendText

Synopsis

Send text message to channel.

Description

Sends A Text Message to a channel while in a call.

Syntax

```
Action: SendText
[ActionID:] <value>
Channel: <value>
Message: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Channel to send message to.
- `Message` - Message to send.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Setvar

Setvar

Synopsis

Set a channel variable.

Description

Set a global or local channel variable.

Syntax

```
Action: Setvar
[ActionID:] <value>
[Channel:] <value>
Variable: <value>
Value: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Channel to set variable for.
- `Variable` - Variable name.
- `Value` - Variable value.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_ShowDialPlan

ShowDialPlan

Synopsis

Show dialplan contexts and extensions

Description

Show dialplan contexts and extensions. Be aware that showing the full dialplan may take a lot of capacity.

Syntax

```
Action: ShowDialPlan
[ActionID:] <value>
[Extension:] <value>
[Context:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Extension` - Show a specific extension.
- `Context` - Show a specific context.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_SIPnotify

SIPnotify

Send a SIP notify.

Description

Sends a SIP Notify event.

All parameters for this event must be specified in the body of this request via multiple Variable: name=value sequences.

Syntax

```
Action: SIPnotify
[ActionID:] <value>
Channel: <value>
Variable: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Peer to receive the notify.
- `Variable` - At least one variable pair must be specified. *name* = *value*

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_SIPpeers

SIPpeers

Synopsis

List SIP peers (text format).

Description

Lists SIP peers in text format with details on current status. Peerlist will follow as separate events, followed by a final event called PeerlistComplete.

Syntax

```
Action: SIPpeers
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_SIPqualifypeer

SIPqualifypeer

Synopsis

Qualify SIP peers.

Description

Qualify a SIP peer.

Syntax

```
Action: SIPqualifypeer
[ActionID:] <value>
Peer: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Peer` - The peer name you want to qualify.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_SIPshowpeer

SIPshowpeer

Synopsis

show SIP peer (text format).

Description

Show one SIP peer with details on current status.

Syntax

```
Action: SIPshowpeer
[ActionID:] <value>
Peer: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Peer` - The peer name you want to check.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_SIPshowregistry

SIPshowregistry

Synopsis

Show SIP registrations (text format).

Description

Lists all registration requests and status. Registrations will follow as separate events. followed by a final event called RegistrationsComplete.

Syntax

```
Action: SIPshowregistry
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_SKINNYdevices

SKINNYdevices

Synopsis

List SKINNY devices (text format).

Description

Lists Skinny devices in text format with details on current status. Devicelist will follow as separate events, followed by a final event called DevicelistComplete.

```
Action: SKINNYdevices
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_SKINNYlines

SKINNYlines

Synopsis

List SKINNY lines (text format).

Description

Lists Skinny lines in text format with details on current status. Linelist will follow as separate events, followed by a final event called LinelistComplete.

Syntax

```
Action: SKINNYlines
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_SKINNYshowdevice

SKINNYshowdevice

Synopsis

Show SKINNY device (text format).

Show one SKINNY device with details on current status.

Syntax

```
Action: SKINNYshowdevice
[ActionID:] <value>
Device: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Device` - The device name you want to check.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_SKINNYshowline

SKINNYshowline

Synopsis

Show SKINNY line (text format).

Description

Show one SKINNY line with details on current status.

Syntax

```
Action: SKINNYshowline
[ActionID:] <value>
Line: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Line` - The line name you want to check.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_Status

Status

List channel status.

Description

Will return the status information of each channel along with the value for the specified channel variables.

Syntax

```
Action: Status
[ActionID:] <value>
Channel: <value>
[Variables:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - The name of the channel to query for status.
- `Variables` - Comma , separated list of variable to include.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_StopMonitor

StopMonitor

Synopsis

Stop monitoring a channel.

Description

This action may be used to end a previously started 'Monitor' action.

Syntax

```
Action: StopMonitor
[ActionID:] <value>
Channel: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - The name of the channel monitored.

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_UnpauseMonitor

UnpauseMonitor

Synopsis

Unpause monitoring of a channel.

Description

This action may be used to re-enable recording of a channel after calling PauseMonitor.

Syntax

```
Action: UnpauseMonitor
[ActionID:] <value>
Channel: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Channel` - Used to specify the channel to record.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_UpdateConfig

UpdateConfig

Synopsis

Update basic configuration.

Description

This action will modify, create, or delete configuration elements in Asterisk configuration files.

Syntax

```
Action: UpdateConfig
[ActionID:] <value>
SrcFilename: <value>
DstFilename: <value>
[Reload:] <value>
[Action-XXXXXX:] <value>
[Cat-XXXXXX:] <value>
[Var-XXXXXX:] <value>
[Value-XXXXXX:] <value>
[Match-XXXXXX:] <value>
[Line-XXXXXX:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `SrcFilename` - Configuration filename to read (e.g. `foo.conf`).
- `DstFilename` - Configuration filename to write (e.g. `foo.conf`)
- `Reload` - Whether or not a reload should take place (or name of specific module).
- `Action-XXXXXX` - Action to take. X's represent 6 digit number beginning with 000000.
    - `NewCat`
    - `RenameCat`
    - `DelCat`
    - `EmptyCat`
    - `Update`
    - `Delete`
    - `Append`
    - `Insert`
- `Cat-XXXXXX` - Category to operate on. X's represent 6 digit number beginning with 000000.
- `Var-XXXXXX` - Variable to work on. X's represent 6 digit number beginning with 000000.
- `Value-XXXXXX` - Value to work on. X's represent 6 digit number beginning with 000000.
- `Match-XXXXXX` - Extra match required to match line. X's represent 6 digit number beginning with 000000.
- `Line-XXXXXX` - Line in category to operate on (used with delete and insert actions). X's represent 6 digit number beginning with 000000.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_UserEvent

UserEvent

Synopsis

Send an arbitrary event.

Description

Send an event to manager sessions.

Syntax

```
Action: UserEvent
[ActionID:] <value>
UserEvent: <value>
[Header1:] <value>
[HeaderN:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `UserEvent` - Event string to send.
- `Header1` - Content1.
- `HeaderN` - ContentN.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_VoicemailUsersList

VoicemailUsersList

Synopsis

List All Voicemail User Information.

Description

Syntax

```
Action: VoicemailUsersList
[ActionID:] <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

ManagerAction_WaitEvent

WaitEvent

Synopsis

Wait for an event to occur.

Description

This action will ellicit a `Success` response. Whenever a manager event is queued. Once WaitEvent has been called on an HTTP manager session, events will be generated and queued.

Syntax

```
Action: WaitEvent
[ActionID:] <value>
Timeout: <value>
```

Arguments

- `ActionID` - ActionID for this transaction. Will be returned.
- `Timeout` - Maximum time (in seconds) to wait for events, -1 means forever.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Dialplan Applications

Application_AddQueueMember

AddQueueMember()

Synopsis

Dynamically adds queue members.

Description

Dynamically adds interface to an existing queue. If the interface is already in the queue it will return an error.

This application sets the following channel variable upon completion:

- `AQMSTATUS` - The status of the attempt to add a queue member as a text string.
  - `ADDED`
  - `MEMBERALREADY`
  - `NOSUCHQUEUE`

Syntax

```
AddQueueMember(queuename[,interface[,penalty[,options[,membername[,sta
```

Arguments

- `queuename`
- `interface`
- `penalty`
- `options`
- `membername`

- `stateinterface`

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS
Function_QUEUE_WAITING_COUNT
Function_QUEUE_MEMBER_LIST
Function_QUEUE_MEMBER_PENALTY

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ADSIProg

ADSIProg()

Synopsis

Load Asterisk ADSI Scripts into phone

Description

This application programs an ADSI Phone with the given script

Syntax

```
ADSIProg([script])
```

Arguments

- `script` - adsi script to use. If not given uses the default script `asterisk.adsi`

See Also

Application_GetCPEID
`adsi.conf`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_AELSub

AELSub()

Launch subroutine built with AEL

Description

Execute the named subroutine, defined in AEL, from another dialplan language, such as extensions.conf, Realtime extensions, or Lua.

The purpose of this application is to provide a sane entry point into AEL subroutines, the implementation of which may change from time to time.

Syntax

```
AELSub(routine[,args])
```

Arguments

- `routine` - Named subroutine to execute.
- `args`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_AgentLogin

AgentLogin()

Synopsis

Call agent login.

Description

Asks the agent to login to the system. Always returns `-1`. While logged in, the agent can receive calls and will hear a `beep` when a new call comes in. The agent can dump the call by pressing the star key.

Syntax

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

Arguments

- `AgentNo`
- `options`
  - `s` - silent login - do not announce the login ok segment after agent logged on/off

Application_Queue
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_AGENT
`agents.conf`
`queues.conf`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_AgentMonitorOutgoing

AgentMonitorOutgoing()

Synopsis

Record agent's outgoing call.

Description

Tries to figure out the id of the agent who is placing outgoing call based on comparison of the callerid of the current interface and the global variable placed by the AgentCallbackLogin application. That's why it should be used only with the AgentCallbackLogin app. Uses the monitoring functions in chan_agent instead of Monitor application. That has to be configured in the `agents.conf` file.

Normally the app returns `0` unless the options are passed.

Syntax

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

Arguments

- `options`
    - `d` - make the app return `-1` if there is an error condition.
    - `c` - change the CDR so that the source of the call is `Agent/agent_id`
    - `n` - don't generate the warnings when there is no callerid or the agentid is not known. It's handy if you want to have one context for agent and non-agent calls.

See Also

`agents.conf`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_AGI

AGI()

Synopsis

Executes an AGI compliant application.

Description

Executes an Asterisk Gateway Interface compliant program on a channel. AGI allows Asterisk to launch external programs written in any language to control a telephony channel, play audio, read DTMF digits, etc. by communicating with the AGI protocol on **stdin** and **stdout**. As of `1.6.0`, this channel will not stop dialplan execution on hangup inside of this application. Dialplan execution will continue normally, even upon hangup until the AGI application signals a desire to stop (either by exiting or, in the case of a net script, by closing the connection). A locally executed AGI script will receive SIGHUP on hangup from the channel except when using DeadAGI. A fast AGI server will correspondingly receive a HANGUP inline with the command dialog. Both of theses signals may be disabled by setting the Executes an Asterisk Gateway Interface compliant program on a channel. AGI allows Asterisk to launch external programs written in any language to control a telephony channel, play audio, read DTMF digits, etc. by communicating with the AGI protocol on `None - AGISIGHUP` channel variable to `no` before executing the AGI application.

Use the CLI command `agi show commands` to list available agi commands.

This application sets the following channel variable upon completion:

- `AGISTATUS` - The status of the attempt to the run the AGI script text string, one of:
    - `SUCCESS`
    - `FAILURE`
    - `NOTFOUND`
    - `HANGUP`

Syntax

```
AGI(command[,arg1[,arg2]])
```

Arguments

- `command`
- `args`
    - `arg1`
    - `arg2`

See Also

Application_EAGI
Application_DeadAGI

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_AlarmReceiver

AlarmReceiver()

Synopsis

Provide support for receiving alarm reports from a burglar or fire alarm panel.

Description

This application should be called whenever there is an alarm panel calling in to dump its events. The application will handshake with the alarm panel, and receive events, validate them, handshake them, and store them until the panel hangs up. Once the panel hangs up, the application will run the system command specified by the eventcmd setting in `alarmreceiver.conf` and pipe the events to the standard input of the application. The configuration file also contains settings for DTMF timing, and for the loudness of the acknowledgement tones.

Only 1 signalling format is supported at this time: Ademco Contact ID.Only 1 signalling format is supported at this time: Ademco Contact ID.

Syntax

```
AlarmReceiver()
```

Arguments

See Also

`alarmreceiver.conf`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_AMD

AMD()

Synopsis

Attempt to detect answering machines.

Description

This application attempts to detect answering machines at the beginning of outbound calls. Simply call this application after the call has been answered (outbound only, of course).

When loaded, AMD reads amd.conf and uses the parameters specified as default values. Those default values get overwritten when the calling AMD with parameters.

This application sets the following channel variables:

- `AMDSTATUS` - This is the status of the answering machine detection
  - `MACHINE`
  - `HUMAN`
  - `NOTSURE`
  - `HANGUP`
- `AMDCAUSE` - Indicates the cause that led to the conclusion
  - `TOOLONG` - Total Time.
  - `INITIALSILENCE` - Silence Duration - Initial Silence.
  - `HUMAN` - Silence Duration - afterGreetingSilence.
  - `LONGGREETING` - Voice Duration - Greeting.
  - `MAXWORDLENGTH` - Word Count - maximum number of words.

Syntax

```
AMD([initialSilence[,greeting[,afterGreetingSilence[,totalAnalysis
Time[,miniumWordLength[,betweenWordSilence[,maximumNumberOfWords[,sile
```

Arguments

- `initialSilence` - Is maximum initial silence duration before greeting. If this is exceeded set as MACHINE
- `greeting` - is the maximum length of a greeting. If this is exceeded set as MACHINE
- `afterGreetingSilence` - Is the silence after detecting a greeting. If this is exceeded set as HUMAN
- `totalAnalysis Time` - Is the maximum time allowed for the algorithm to decide HUMAN or MACHINE
- `miniumWordLength` - Is the minimum duration of Voice considered to be a word
- `betweenWordSilence` - Is the minimum duration of silence after a word to consider the audio that follows to be a new word
- `maximumNumberOfWords` - Is the maximum number of words in a greeting If this is exceeded set as MACHINE
- `silenceThreshold` - How long do we consider silence
- `maximumWordLength` - Is the maximum duration of a word to accept. If exceeded set as MACHINE

See Also

[Application_WaitForSilence](#)
[Application_WaitForNoise](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Answer

Answer()

Synopsis

Answer a channel if ringing.

Description

If the call has not been answered, this application will answer it. Otherwise, it has no effect on the call.

Syntax

```
Answer([delay[,nocdr]])
```

Arguments

- `delay` - Asterisk will wait this number of milliseconds before returning to the dialplan after answering the call.
- `nocdr` - Asterisk will send an answer signal to the calling phone, but will not set the disposition or answer time in the CDR for this call.

See Also

Application_Hangup

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Authenticate

Authenticate()

Synopsis

Authenticate a user

Description

This application asks the caller to enter a given password in order to continue dialplan execution.

If the password begins with the `/` character, it is interpreted as a file which contains a list of valid passwords, listed 1 password per line in the file.

When using a database key, the value associated with the key can be anything.

Users have three attempts to authenticate before the channel is hung up.

Syntax

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

Arguments

- `password` - Password the user should know
- `options`
    - `a` - Set the channels' account code to the password that is entered
    - `d` - Interpret the given path as database key, not a literal file
    - `m` - Interpret the given path as a file which contains a list of account codes and password hashes delimited with `:`, listed one per line in the file. When one of the passwords is matched, the channel will have its account code set to the corresponding account code in the file.
    - `r` - Remove the database key upon successful entry (valid with `d` only)
- `maxdigits` - maximum acceptable number of digits. Stops reading after maxdigits have been entered (without requiring the user to press the `#` key). Defaults to 0 - no limit - wait for the user press the `#` key.
- `prompt` - Override the agent-pass prompt file.

See Also

Application_VMAuthenticate
Application_DISA

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_BackGround

BackGround()

Synopsis

Play an audio file while waiting for digits of an extension to go to.

Description

This application will play the given list of files **(do not put extension)** while waiting for an extension to be dialed by the calling channel. To continue waiting for digits after this application has finished playing files, the `WaitExten` application should be used.

If one of the requested sound files does not exist, call processing will be terminated.

This application sets the following channel variable upon completion:

- `BACKGROUNDSTATUS` - The status of the background attempt as a text string.
    - `SUCCESS`
    - `FAILED`

Syntax

```
BackGround(filename1[&filename2[&...]][,options[,langoverride[,context
```

Arguments

- `filenames`
    - `filename1`
    - `filename2`
- `options`
    - `s` - Causes the playback of the message to be skipped if the channel is not in the `up` state (i.e. it hasn't been answered yet). If this happens, the application will return immediately.
    - `n` - Don't answer the channel before playing the files.
    - `m` - Only break if a digit hit matches a one digit extension in the destination context.
- `langoverride` - Explicitly specifies which language to attempt to use for the requested sound files.
- `context` - This is the dialplan context that this application will use when exiting to a dialed extension.

See Also

Application_ControlPlayback
Application_WaitExten
Application_BackgroundDetect
Function_TIMEOUT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_BackgroundDetect

BackgroundDetect()

Background a file with talk detect.

Description

Plays back *filename*, waiting for interruption from a given digit (the digit must start the beginning of a valid extension, or it will be ignored). During the playback of the file, audio is monitored in the receive direction, and if a period of non-silence which is greater than *min* ms yet less than *max* ms is followed by silence for at least *sil* ms, which occurs during the first *analysistime* ms, then the audio playback is aborted and processing jumps to the *talk* extension, if available.

Syntax

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

Arguments

- `filename`
- `sil` - If not specified, defaults to `1000`.
- `min` - If not specified, defaults to `100`.
- `max` - If not specified, defaults to `infinity`.
- `analysistime` - If not specified, defaults to `infinity`.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Bridge

Bridge()

Synopsis

Bridge two channels.

Description

Allows the ability to bridge two channels via the dialplan.

This application sets the following channel variable upon completion:

- `BRIDGERESULT` - The result of the bridge attempt as a text string.
  - `SUCCESS`
  - `FAILURE`
  - `LOOP`
  - `NONEXISTENT`
  - `INCOMPATIBLE`

Syntax

```
Bridge(channel[,options])
```

Arguments

- `channel` - The current channel is bridged to the specified *channel*.
- `options`
  - `p` - Play a courtesy tone to *channel*.
  - `h` - Allow the called party to hang up by sending the `*` DTMF digit.
  - `H` - Allow the calling party to hang up by pressing the `*` DTMF digit.
  - `k` - Allow the called party to enable parking of the call by sending the DTMF sequence defined for call parking in features.conf.
  - `K` - Allow the calling party to enable parking of the call by sending the DTMF sequence defined for call parking in features.conf.
  - `L(x[:y][:z])` - Limit the call to *x* ms. Play a warning when *y* ms are left. Repeat the warning every *z* ms. The following special variables can be used with this option: Play sounds to the caller. yes|no (default yes) Play sounds to the callee. yes|no File to play when time is up. File to play when call begins. File to play as warning if *y* is defined. The default is to say the time remaining.
  - `S` - Hang up the call after *x* seconds **after** the called party has answered the call.
  - `t` - Allow the called party to transfer the calling party by sending the DTMF sequence defined in features.conf.
  - `T` - Allow the calling party to transfer the called party by sending the DTMF sequence defined in features.conf.
  - `w` - Allow the called party to enable recording of the call by sending the DTMF sequence defined for one-touch recording in features.conf.
  - `W` - Allow the calling party to enable recording of the call by sending the DTMF sequence defined for one-touch recording in features.conf.
  - `x` - Cause the called party to be hung up after the bridge, instead of being restarted in the dialplan.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Busy

Busy()

Synopsis

Indicate the Busy condition.

Description

This application will indicate the busy condition to the calling channel.

Syntax

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

Arguments

- `timeout` - If specified, the calling channel will be hung up after the specified number of seconds. Otherwise, this application will wait until the calling channel hangs up.

See Also

Application_Congestion
[Application_Progess]
Application_PlayTones
Application_Hangup

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_CallCompletionCancel

CallCompletionCancel()

Synopsis

Cancel call completion service

Description

Cancel a Call Completion Request.

This application sets the following channel variables:

- `CC_CANCEL_RESULT` - This is the returned status of the cancel.
    - `SUCCESS`
    - `FAIL`
- `CC_CANCEL_REASON` - This is the reason the cancel failed.
    - `NO_CORE_INSTANCE`
    - `NOT_GENERIC`
    - `UNSPECIFIED`

Syntax

```
CallCompletionCancel()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r312509.

Application_CallCompletionRequest

CallCompletionRequest()

Synopsis

Request call completion service for previous call

Description

Request call completion service for a previously failed call attempt.

This application sets the following channel variables:

- `CC_REQUEST_RESULT` - This is the returned status of the request.
    - `SUCCESS`
    - `FAIL`
- `CC_REQUEST_REASON` - This is the reason the request failed.
    - `NO_CORE_INSTANCE`
    - `NOT_GENERIC`
    - `TOO_MANY_REQUESTS`

- `UNSPECIFIED`

Syntax

```
CallCompletionRequest()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r312509.

Application_CELGenUserEvent

CELGenUserEvent()

Synopsis

Generates a CEL User Defined Event.

Description

A CEL event will be immediately generated by this channel, with the supplied name for a type.

Syntax

```
CELGenUserEvent(event-name[,extra])
```

Arguments

- `event-name`
    - `event-name`
    - `extra` - Extra text to be included with the event.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ChangeMonitor

ChangeMonitor()

Synopsis

Change monitoring filename of a channel.

Description

Changes monitoring filename of a channel. Has no effect if the channel is not monitored.

```
ChangeMonitor(filename_base)
```

Arguments

- `filename_base` - The new filename base to use for monitoring this channel.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ChanIsAvail

ChanIsAvail()

Synopsis

Check channel availability

Description

This application will check to see if any of the specified channels are available.

This application sets the following channel variables:

- `AVAILCHAN` - The name of the available channel, if one exists
- `AVAILORIGCHAN` - The canonical channel name that was used to create the channel
- `AVAILSTATUS` - The device state for the device
- `AVAILCAUSECODE` - The cause code returned when requesting the channel

Syntax

```
ChanIsAvail([Technology2/Resource2[&...]][,options])
```

Arguments

- `Technology/Resource` - Specification of the device(s) to check. These must be in the format of `Technology/Resource`, where *Technology* represents a particular channel driver, and *Resource* represents a resource available to that particular channel driver.
    - `Technology2/Resource2` - Optional extra devices to check If you need more then one enter them as Technology2/Resource2&Technology3/Resourse3&.....
- `options`
    - `a` - Check for all available channels, not only the first one
    - `s` - Consider the channel unavailable if the channel is in use at all
    - `t` - Simply checks if specified channels exist in the channel list

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ChannelRedirect

ChannelRedirect()

Redirects given channel to a dialplan target

Description

Sends the specified channel to the specified extension priority

This application sets the following channel variables upon completion

- CHANNELREDIRECT_STATUS -
  - NOCHANNEL
  - SUCCESS Are set to the result of the redirection

Syntax

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

Arguments

- channel
- context
- extension
- priority

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ChanSpy

ChanSpy()

Synopsis

Listen to a channel, and optionally whisper into it.

Description

This application is used to listen to the audio from an Asterisk channel. This includes the audio coming in and out of the channel being spied on. If the chanprefix parameter is specified, only channels beginning with this string will be spied upon.

While spying, the following actions may be performed:

- Dialing # cycles the volume level.

- Dialing * will stop spying and look for another channel to spy on.

- Dialing a series of digits followed by # builds a channel name to append to 'chanprefix'. For example, executing ChanSpy(Agent) and then dialing the digits '1234#' while spying will begin spying on the channel 'Agent/1234'. Note that this feature will be overridden if the 'd' option is used

The The *X* option supersedes the three features above in that if a valid single digit extension exists in the correct context ChanSpy will exit to it. This also disables choosing a channel based on `chanprefix` and a digit sequence.

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

Arguments

- `chanprefix`
- `options`
  - `b` - Only spy on channels involved in a bridged call.
  - `B` - Instead of whispering on a single channel barge in on both channels involved in the call.
  - `c`
    - `digit` - Specify a DTMF digit that can be used to spy on the next available channel.
  - `d` - Override the typical numeric DTMF functionality and instead use DTMF to switch between spy modes.
    - `4` - spy mode
    - `5` - whisper mode
    - `6` - barge mode
  - `e` - Enable **enforced** mode, so the spying channel can only monitor extensions whose name is in the *ext* : delimited list.
    - `ext`
  - `E` - Exit when the spied-on channel hangs up.
  - `g` - both both *grp* and `SPYGROUP` can contain either a single group or a colon-delimited list of groups, such as `sales:support:accounting`.
    - `grp` - Only spy on channels in which one or more of the groups listed in *grp* matches one or more groups from the `SPYGROUP` variable set on the channel to be spied upon.
  - `n` - Say the name of the person being spied on if that person has recorded his/her name. If a context is specified, then that voicemail context will be searched when retrieving the name, otherwise the `default` context be used when searching for the name (i.e. if SIP/1000 is the channel being spied on and no mailbox is specified, then `1000` will be used when searching for the name).
    - `mailbox`
    - `context`
  - `o` - Only listen to audio coming from this channel.
  - `q` - Don't play a beep when beginning to spy on a channel, or speak the selected channel name.
  - `r` - Record the session to the monitor spool directory. An optional base for the filename may be specified. The default is `chanspy`.
    - `basename`
  - `s` - Skip the playback of the channel type (i.e. SIP, IAX, etc) when speaking the selected channel name.
  - `S` - Stop when no more channels are left to spy on.
  - `v` - Adjust the initial volume in the range from `-4` to `4`. A negative value refers to a quieter setting.
    - `value`
  - `w` - Enable `whisper` mode, so the spying channel can talk to the spied-on channel.
  - `W` - Enable `private whisper` mode, so the spying channel can talk to the spied-on channel but cannot listen to that channel.
  - `x`
    - `digit` - Specify a DTMF digit that can be used to exit the application.
  - `X` - Allow the user to exit ChanSpy to a valid single digit numeric extension in the current context or the context specified by the `SPY_EXIT_CONTEXT` channel variable. The name of the last channel that was spied on will be stored in the `SPY_CHANNEL` variable.
  - `4` - spy mode
  - `5` - whisper mode
  - `6` - barge mode

Application_ExtenSpy

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ClearHash

ClearHash()

Clear the keys from a specified hashname.

Clears all keys out of the specified *hashname*.

```
ClearHash(hashname)
```

Arguments

- `hashname`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ConfBridge

ConfBridge()

Conference bridge application.

Enters the user into a specified conference bridge. The user can exit the conference by hangup only.

The join sound can be set using the `CONFBRIDGE_JOIN_SOUND` variable and the leave sound can be set using the `CONFBRIDGE_LEAVE_SOUND` variable. These can be unique to the caller.

This application will not automatically answer the channel.This application will not automatically answer the channel.

```
ConfBridge([confno[,options]])
```

Arguments

- `confno` - The conference number
- `options`
    - `a` - Set admin mode.
    - `A` - Set marked mode.

- `c` - Announce user(s) count on joining a conference.
- `m` - Set initially muted.
- `M` - Enable music on hold when the conference has a single caller. Optionally, specify a musiconhold class to use. If one is not provided, it will use the channel's currently set music class, or `default`.
    - `class`
- `l` - Do not play message when first person enters
- `s` - Present menu (user or admin) when * is received (send to menu).
- `w` - Wait until the marked user enters the conference.
- `q` - Quiet mode (don't play enter/leave sounds).

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Congestion

Congestion()

Synopsis

Indicate the Congestion condition.

Description

This application will indicate the congestion condition to the calling channel.

Syntax

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

Arguments

- `timeout` - If specified, the calling channel will be hung up after the specified number of seconds. Otherwise, this application will wait until the calling channel hangs up.

See Also

Application_Busy
[Application_Progess]
Application_PlayTones
Application_Hangup

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ContinueWhile

ContinueWhile()

Synopsis

Restart a While loop.

Description

Returns to the top of the while loop and re-evaluates the conditional.

Syntax

```
ContinueWhile()
```

Arguments

See Also

[Application_While](#)
[Application_EndWhile](#)
[Application_ExitWhile](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ControlPlayback

ControlPlayback()

Synopsis

Play a file with fast forward and rewind.

Description

This application will play back the given *filename*.

It sets the following channel variables upon completion:

- CPLAYBACKSTATUS - Contains the status of the attempt as a text string
    - SUCCESS
    - USERSTOPPED
    - ERROR
- CPLAYBACKOFFSET - Contains the offset in ms into the file where playback was at when it stopped. -1 is end of file.
- CPLAYBACKSTOPKEY - If the playback is stopped by the user this variable contains the key that was pressed.

Syntax

```
ControlPlayback(filename[,skipms[,ff[,rew[,stop[,pause[,restart[,optio
```

Arguments

- filename
- skipms - This is number of milliseconds to skip when rewinding or fast-forwarding.
- ff - Fast-forward when this DTMF digit is received. (defaults to #)
- rew - Rewind when this DTMF digit is received. (defaults to *)
- stop - Stop playback when this DTMF digit is received.
- pause - Pause playback when this DTMF digit is received.
- restart - Restart playback when this DTMF digit is received.
- options
    - o

- `time` - Start at *time* ms from the beginning of the file.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DAHDIAcceptR2Call

DAHDIAcceptR2Call()

Synopsis

Accept an R2 call if its not already accepted (you still need to answer it)

Description

This application will Accept the R2 call either with charge or no charge.

Syntax

```
DAHDIAcceptR2Call(charge)
```

Arguments

- `charge` - Yes or No. Whether you want to accept the call with charge or without charge.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DAHDIBarge

DAHDIBarge()

Synopsis

Barge in (monitor) DAHDI channel.

Description

Barges in on a specified DAHDI *channel* or prompts if one is not specified. Returns -1 when caller user hangs up and is independent of the state of the channel being monitored.

Syntax

```
DAHDIBarge([channel])
```

Arguments

- `channel` - Channel to barge.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DAHDIRAS

DAHDIRAS()

Synopsis

Executes DAHDI ISDN RAS application.

Description

Executes a RAS server using pppd on the given channel. The channel must be a clear channel (i.e. PRI source) and a DAHDI channel to be able to use this function (No modem emulation is included).

Your pppd must be patched to be DAHDI aware.

Syntax

```
DAHDIRAS(args)
```

Arguments

- `args` - A list of parameters to pass to the pppd daemon, separated by `,` characters.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DAHDIScan

DAHDIScan()

Synopsis

Scan DAHDI channels to monitor calls.

Description

Allows a call center manager to monitor DAHDI channels in a convenient way. Use # to select the next channel and use * to exit.

Syntax

```
DAHDIScan([group])
```

Arguments

- `group` - Limit scanning to a channel *group* by setting this option.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DAHDISendCallreroutingFacility

DAHDISendCallreroutingFacility()

Synopsis

Send an ISDN call rerouting/deflection facility message.

Description

This application will send an ISDN switch specific call rerouting/deflection facility message over the current channel. Supported switches depend upon the version of libpri in use.

Syntax

```
DAHDISendCallreroutingFacility(destination[,original[,reason]])
```

Arguments

- `destination` - Destination number.
- `original` - Original called number.
- `reason` - Diversion reason, if not specified defaults to `unknown`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DAHDISendKeypadFacility

DAHDISendKeypadFacility()

Synopsis

Send digits out of band over a PRI.

Description

This application will send the given string of digits in a Keypad Facility IE over the current

channel.

```
DAHDISendKeypadFacility(digits)
```

Arguments

- `digits`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DateTime

DateTime()

Synopsis

Says a specified time in a custom format.

Description

Say the date and time in a specified format.

Syntax

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

Arguments

- `unixtime` - time, in seconds since Jan 1, 1970. May be negative. Defaults to now.
- `timezone` - timezone, see `/usr/share/zoneinfo` for a list. Defaults to machine default.
- `format` - a format the time is to be said in. See `voicemail.conf`. Defaults to `ABdY "digits/at" IMp`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DBdel

DBdel()

Synopsis

Delete a key from the asterisk database.

Description

This application will delete a *key* from the Asterisk database.

This application has been DEPRECATED in favor of the DB_DELETE function.This application has been DEPRECATED in favor of the DB_DELETE function.

Syntax

```
DBdel(family,key)
```

Arguments

- `family`
- `key`

See Also

[Function_DB_DELETE](#)
[Application_DBdeltree](#)
[Function_DB](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DBdeltree

DBdeltree()

Synopsis

Delete a family or keytree from the asterisk database.

Description

This application will delete a *family* or *keytree* from the Asterisk database.

Syntax

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

Arguments

- `family`
- `keytree`

See Also

[Function_DB_DELETE](#)
[Application_DBdel](#)
[Function_DB](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DeadAGI

DeadAGI()

Synopsis

Executes AGI on a hungup channel.

Description

Executes an Asterisk Gateway Interface compliant program on a channel. AGI allows Asterisk to launch external programs written in any language to control a telephony channel, play audio, read DTMF digits, etc. by communicating with the AGI protocol on **stdin** and **stdout**. As of `1.6.0`, this channel will not stop dialplan execution on hangup inside of this application. Dialplan execution will continue normally, even upon hangup until the AGI application signals a desire to stop (either by exiting or, in the case of a net script, by closing the connection). A locally executed AGI script will receive SIGHUP on hangup from the channel except when using DeadAGI. A fast AGI server will correspondingly receive a HANGUP inline with the command dialog. Both of theses signals may be disabled by setting the Executes an Asterisk Gateway Interface compliant program on a channel. AGI allows Asterisk to launch external programs written in any language to control a telephony channel, play audio, read DTMF digits, etc. by communicating with the AGI protocol on `None - AGISIGHUP` channel variable to `no` before executing the AGI application.

Use the CLI command `agi show commands` to list available agi commands.

This application sets the following channel variable upon completion:

- `AGISTATUS` - The status of the attempt to the run the AGI script text string, one of:
    - `SUCCESS`
    - `FAILURE`
    - `NOTFOUND`
    - `HANGUP`

Syntax

```
DeadAGI(command[,arg1[,arg2]])
```

Arguments

- `command`
- `args`
    - `arg1`
    - `arg2`

See Also

Application_AGI
Application_EAGI

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Dial

Dial()

Synopsis

Attempt to connect to another device or endpoint and bridge the call.

Description

This application will place calls to one or more specified channels. As soon as one of the requested channels answers, the originating channel will be answered, if it has not already been answered. These two channels will then be active in a bridged call. All other channels that were requested will then be hung up.

Unless there is a timeout specified, the Dial application will wait indefinitely until one of the called channels answers, the user hangs up, or if all of the called channels are busy or unavailable. Dialplan executing will continue if no requested channels can be called, or if the timeout expires. This application will report normal termination if the originating channel hangs up, or if the call is bridged and either of the parties in the bridge ends the call.

If the If the `None` - `OUTBOUND_GROUP` variable is set, all peer channels created by this application will be put into that group (as in Set(GROUP()=...). If the If the `None` - `OUTBOUND_GROUP_ONCE` variable is set, all peer channels created by this application will be put into that group (as in Set(GROUP()=...). Unlike OUTBOUND_GROUP, however, the variable will be unset after use.

This application sets the following channel variables:

- `DIALEDTIME` - This is the time from dialing a channel until when it is disconnected.
- `ANSWEREDTIME` - This is the amount of time for actual call.
- `DIALSTATUS` - This is the status of the call
    - `CHANUNAVAIL`
    - `CONGESTION`
    - `NOANSWER`
    - `BUSY`
    - `ANSWER`
    - `CANCEL`
    - `DONTCALL` - For the Privacy and Screening Modes. Will be set if the called party chooses to send the calling party to the 'Go Away' script.
    - `TORTURE` - For the Privacy and Screening Modes. Will be set if the called party chooses to send the calling party to the 'torture' script.
    - `INVALIDARGS`

Syntax

```
Dial(Technology/Resource[&Technology2/Resource2[&...]][,timeout[,optic
```

Arguments

- `Technology/Resource`
    - `Technology/Resource` - Specification of the device(s) to dial. These must be in the format of `Technology/Resource`, where *Technology* represents a particular channel driver, and *Resource* represents a resource available to that particular channel driver.

- `Technology2/Resource2` - Optional extra devices to dial in parallel If you need more then one enter them as Technology2/Resource2&Technology3/Resourse3&.....
- `timeout` - Specifies the number of seconds we attempt to dial the specified devices If not specified, this defaults to 136 years.
- `options`
  - A - Play an announcement to the called party, where *x* is the prompt to be played
    - `x` - The file to play to the called party
  - a - Immediately answer the calling channel when the called channel answers in all cases. Normally, the calling channel is answered when the called channel answers, but when options such as A() and M() are used, the calling channel is not answered until all actions on the called channel (such as playing an announcement) are completed. This option can be used to answer the calling channel before doing anything on the called channel. You will rarely need to use this option, the default behavior is adequate in most cases.
  - C - Reset the call detail record (CDR) for this call.
  - c - If the Dial() application cancels this call, always set the flag to tell the channel driver that the call is answered elsewhere.
  - d - Allow the calling user to dial a 1 digit extension while waiting for a call to be answered. Exit to that extension if it exists in the current context, or the context defined in the `EXITCONTEXT` variable, if it exists.
  - D - Send the specified DTMF strings **after** the called party has answered, but before the call gets bridged. The *called* DTMF string is sent to the called party, and the *calling* DTMF string is sent to the calling party. Both arguments can be used alone. If *progress* is specified, its DTMF is sent immediately after receiving a PROGRESS message.
    - `called`
    - `calling`
    - `progress`
  - e - Execute the `h` extension for peer after the call ends
  - f - If *x* is not provided, force the CallerID sent on a call-forward or deflection to the dialplan extension of this Dial() using a dialplan `hint`. For example, some PSTNs do not allow CallerID to be set to anything other than the numbers assigned to you. If *x* is provided, force the CallerID sent to *x*.
    - `x`
  - F - When the caller hangs up, transfer the called party to the specified destination and continue execution at that location.
    - `context`
    - `exten`
    - `priority`
  - F - Proceed with dialplan execution at the next priority in the current extension if the source channel hangs up.
  - g - Proceed with dialplan execution at the next priority in the current extension if the destination channel hangs up.
  - G - If the call is answered, transfer the calling party to the specified *priority* and the called party to the specified *priority* plus one. You cannot use any additional action post answer options in conjunction with this option.You cannot use any additional action post answer options in conjunction with this option.
    - `context`
    - `exten`
    - `priority`
  - h - Allow the called party to hang up by sending the * DTMF digit.
  - H - Allow the calling party to hang up by hitting the * DTMF digit.
  - i - Asterisk will ignore any forwarding requests it may receive on this dial attempt.
  - I - Asterisk will ignore any connected line update requests or redirecting party update requests it may receiveon this dial attempt.
  - k - Allow the called party to enable parking of the call by sending the DTMF sequence defined for call parking in `features.conf`.
  - K - Allow the calling party to enable parking of the call by sending the DTMF sequence defined for call parking in `features.conf`.
  - L - Limit the call to *x* milliseconds. Play a warning when *y* milliseconds are left. Repeat the warning every *z* milliseconds until time expires. This option is affected by the following variables: If set, this variable causes Asterisk to play the prompts to the caller. If set, this variable causes Asterisk to play the prompts to the callee. If specified, *filename* specifies the sound prompt to play when the timeout is reached. If not set, the time remaining will be announced. If specified, *filename* specifies the sound prompt to play when the call begins. If not set, the time remaining will be announced. If specified, *filename* specifies the sound prompt to play as a warning when time *x* is reached. If not set, the time remaining will be announced.
    - `x` - Maximum call time, in milliseconds
    - `y` - Warning time, in milliseconds
    - `z` - Repeat time, in milliseconds
  - m - Provide hold music to the calling party until a requested channel answers. A specific music on hold *class* (as defined in `musiconhold.conf`) can be specified.
    - `class`
  - M - Execute the specified *macro* for the **called** channel before connecting to the calling channel. Arguments can be specified to the Macro using ^ as a delimiter. The macro can set the variable `MACRO_RESULT` to specify the following actions after the macro is finished executing: If set, this action will be taken after the macro finished executing. You cannot use any additional action post answer options in conjunction with this option. Also, pbx services are not run on the peer (called) channel, so you will not be able to set timeouts via the TIMEOUT() function in this macro.You cannot use any additional action post answer options in conjunction with this option. Also, pbx services are not run on the peer (called) channel, so you will not be able to set timeouts via the TIMEOUT() function in this macro. Be aware of the limitations that macros have, specifically with regards to use of the `WaitExten` application. For more information, see the documentation for Macro()
    - `macro` - Name of the macro that should be executed.
    - `arg` - Macro arguments
  - n - This option is a modifier for the call screening/privacy mode. (See the `p` and `P` options.) It specifies that no introductions are to be saved in the `priv-callerintros` directory.
    - `delete` - With *delete* either not specified or set to `0`, the recorded introduction will not be deleted if the caller hangs up while the remote party has not yet answered. With *delete* set to `1`, the introduction will always be deleted.
  - N - This option is a modifier for the call screening/privacy mode. It specifies that if Caller*ID is present, do not screen the call.

- **o** - If *x* is not provided, specify that the CallerID that was present on the **calling** channel be stored as the CallerID on the **called** channel. This was the behavior of Asterisk 1.0 and earlier. If *x* is provided, specify the CallerID stored on the **called** channel. Note that o(${CALLERID(all)}) is similar to option o without the parameter.
  - `x`
- **O** - Enables **operator services** mode. This option only works when bridging a DAHDI channel to another DAHDI channel only. if specified on non-DAHDI interfaces, it will be ignored. When the destination answers (presumably an operator services station), the originator no longer has control of their line. They may hang up, but the switch will not release their line until the destination party (the operator) hangs up.
  - `mode` - With *mode* either not specified or set to `1`, the originator hanging up will cause the phone to ring back immediately. With *mode* set to `2`, when the operator flashes the trunk, it will ring their phone back.
- **p** - This option enables screening mode. This is basically Privacy mode without memory.
- **P** - Enable privacy mode. Use *x* as the family/key in the AstDB database if it is provided. The current extension is used if a database family/key is not specified.
  - `x`
- **r** - Default: Indicate ringing to the calling party, even if the called party isn't actually ringing. Pass no audio to the calling party until the called channel has answered.
  - `tone` - Indicate progress to calling party. Send audio 'tone' from indications.conf
- **S** - Hang up the call *x* seconds **after** the called party has answered the call.
  - `x`
- **s** - Force the outgoing callerid tag parameter to be set to the string *x*. Works with the f option.
  - `x`
- **t** - Allow the called party to transfer the calling party by sending the DTMF sequence defined in `features.conf`. This setting does not perform policy enforcement on transfers initiated by other methods.
- **T** - Allow the calling party to transfer the called party by sending the DTMF sequence defined in `features.conf`. This setting does not perform policy enforcement on transfers initiated by other methods.
- **U** - Execute via Gosub the routine *x* for the **called** channel before connecting to the calling channel. Arguments can be specified to the Gosub using ^ as a delimiter. The Gosub routine can set the variable `GOSUB_RESULT` to specify the following actions after the Gosub returns. You cannot use any additional action post answer options in conjunction with this option. Also, pbx services are not run on the peer (called) channel, so you will not be able to set timeouts via the TIMEOUT() function in this routine.You cannot use any additional action post answer options in conjunction with this option. Also, pbx services are not run on the peer (called) channel, so you will not be able to set timeouts via the TIMEOUT() function in this routine.
  - `x` - Name of the subroutine to execute via Gosub
  - `arg` - Arguments for the Gosub routine
- **u** - Works with the f option.
  - `x` - Force the outgoing callerid presentation indicator parameter to be set to one of the values passed in *x* :
    ```
    allowed_not_screened
    allowed_passed_screen
    allowed_failed_screen
    allowed
    prohib_not_screened
    prohib_passed_screen
    prohib_failed_screen
    prohib
    unavailable
    ```
- **w** - Allow the called party to enable recording of the call by sending the DTMF sequence defined for one-touch recording in `features.conf`.
- **W** - Allow the calling party to enable recording of the call by sending the DTMF sequence defined for one-touch recording in `features.conf`.
- **x** - Allow the called party to enable recording of the call by sending the DTMF sequence defined for one-touch automixmonitor in `features.conf`.
- **X** - Allow the calling party to enable recording of the call by sending the DTMF sequence defined for one-touch automixmonitor in `features.conf`.
- **z** - On a call forward, cancel any dial timeout which has been set for this call.
- `URL` - The optional URL will be sent to the called party if the channel driver supports it.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Dictate

Dictate()

Synopsis

Virtual Dictation Machine.

Start dictation machine using optional *base_dir* for files.

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

Arguments

- `base_dir`
- `filename`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Directory

Directory()

Provide directory of voicemail extensions.

This application will present the calling channel with a directory of extensions from which they can search by name. The list of names and corresponding extensions is retrieved from the voicemail configuration file, `voicemail.conf`.

This application will immediately exit if one of the following DTMF digits are received and the extension to jump to exists:
`0` - Jump to the 'o' extension, if it exists. `*` - Jump to the 'a' extension, if it exists.

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

Arguments

- `vm-context` - This is the context within voicemail.conf to use for the Directory. If not specified and `searchcontexts=no` in `voicemail.conf`, then `default` will be assumed.
- `dial-context` - This is the dialplan context to use when looking for an extension that the user has selected, or when jumping to the `o` or `a` extension.
- `options` - Only one of the Only one of the *f*, *l*, or *b* options may be specified. **If more than one is specified**, then Directory will act as if *b* was specified. The number of characters for the user to type defaults to `3`.
  - `e` - In addition to the name, also read the extension number to the caller before presenting dialing options.
  - `f` - Allow the caller to enter the first name of a user in the directory instead of using the last name. If specified, the optional number argument will be used for the number of characters the user should enter.
    - `n`
  - `l` - Allow the caller to enter the last name of a user in the directory. This is the default. If specified, the optional number argument will be used for the number of characters the user should enter.

- n
  - b - Allow the caller to enter either the first or the last name of a user in the directory. If specified, the optional number argument will be used for the number of characters the user should enter.
    - n
  - m - Instead of reading each name sequentially and asking for confirmation, create a menu of up to 8 names.
  - n - Read digits even if the channel is not answered.
  - p - Pause for n milliseconds after the digits are typed. This is helpful for people with cellphones, who are not holding the receiver to their ear while entering DTMF.
    - n

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DISA

DISA()

Synopsis

Direct Inward System Access.

Description

The DISA, Direct Inward System Access, application allows someone from outside the telephone switch (PBX) to obtain an **internal** system dialtone and to place calls from it as if they were placing a call from within the switch. DISA plays a dialtone. The user enters their numeric passcode, followed by the pound sign #. If the passcode is correct, the user is then given system dialtone within *context* on which a call may be placed. If the user enters an invalid extension and extension i exists in the specified *context*, it will be used.

Be aware that using this may compromise the security of your PBX.

The arguments to this application (in extensions.conf) allow either specification of a single global *passcode* (that everyone uses), or individual passcodes contained in a file (*filename*).

The file that contains the passcodes (if used) allows a complete specification of all of the same arguments available on the command line, with the sole exception of the options. The file may contain blank lines, or comments starting with # or ; .

Syntax

```
DISA(passcode|filename[,context[,cid[,mailbox[@context][,options]]]])
```

Arguments

- passcode|filename - If you need to present a DISA dialtone without entering a password, simply set *passcode* to no-password You may specified a *filename* instead of a *passcode*, this filename must contain individual passcodes
- context - Specifies the dialplan context in which the user-entered extension will be matched. If no context is specified, the DISA application defaults to the disa context. Presumably a normal system will have a special context set up for DISA use with some or a lot of restrictions.
- cid - Specifies a new (different) callerid to be used for this call.
- mailbox - Will cause a stutter-dialtone (indication **dialrecall**) to be used, if the specified mailbox contains any new messages.
  - mailbox
  - context

- options
  - n - The DISA application will not answer initially.
  - p - The extension entered will be considered complete when a # is entered.

See Also

Application_Authenticate
Application_VMAuthenticate

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_DumpChan

DumpChan()

Synopsis

Dump Info About The Calling Channel.

Description

Displays information on channel and listing of all channel variables. If *level* is specified, output is only displayed when the verbose level is currently set to that number or greater.

Syntax

```
DumpChan([level])
```

Arguments

- level - Minimun verbose level

See Also

Application_NoOp
Application_Verbose

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_EAGI

EAGI()

Synopsis

Executes an EAGI compliant application.

Description

Using 'EAGI' provides enhanced AGI, with incoming audio available out of band on file descriptor 3.

Executes an Asterisk Gateway Interface compliant program on a channel. AGI allows Asterisk to launch external programs written in any language to control a telephony channel, play audio, read DTMF digits, etc. by communicating with the AGI protocol on **stdin** and **stdout**. As of `1.6.0`, this channel will not stop dialplan execution on hangup inside of this application. Dialplan execution will continue normally, even upon hangup until the AGI application signals a desire to stop (either by exiting or, in the case of a net script, by closing the connection). A locally executed AGI script will receive SIGHUP on hangup from the channel except when using DeadAGI. A fast AGI server will correspondingly receive a HANGUP inline with the command dialog. Both of theses signals may be disabled by setting the Executes an Asterisk Gateway Interface compliant program on a channel. AGI allows Asterisk to launch external programs written in any language to control a telephony channel, play audio, read DTMF digits, etc. by communicating with the AGI protocol on `None` - `AGISIGHUP` channel variable to `no` before executing the AGI application.

Use the CLI command `agi show commands` to list available agi commands.

This application sets the following channel variable upon completion:

- `AGISTATUS` - The status of the attempt to the run the AGI script text string, one of:
    - `SUCCESS`
    - `FAILURE`
    - `NOTFOUND`
    - `HANGUP`

Syntax

```
EAGI(command[,arg1[,arg2]])
```

Arguments

- `command`
- `args`
    - `arg1`
    - `arg2`

See Also

Application_AGI
Application_DeadAGI

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Echo

Echo()

Synopsis

Echo audio, video, DTMF back to the calling party

Description

Echos back any audio, video or DTMF frames read from the calling channel back to itself. Note: If '#' detected application exits

Syntax

```
Echo()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_EndWhile

EndWhile()

Synopsis

End a while loop.

Description

Return to the previous called `While()`.

Syntax

```
EndWhile()
```

Arguments

See Also

Application_While
Application_ExitWhile
Application_ContinueWhile

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Exec

Exec()

Synopsis

Executes dialplan application.

Allows an arbitrary application to be invoked even when not hard coded into the dialplan. If the underlying application terminates the dialplan, or if the application cannot be found, Exec will terminate the dialplan.

To invoke external applications, see the application System. If you would like to catch any error instead, see TryExec.

Syntax

```
Exec(arguments)
```

Arguments

- `appname` - Application name and arguments of the dialplan application to execute.
    - `arguments`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ExecIf

ExecIf()

Synopsis

Executes dialplan application, conditionally.

Description

If *expr* is true, execute and return the result of *appiftrue(args)*.

If *expr* is true, but *appiftrue* is not found, then the application will return a non-zero value.

Syntax

```
ExecIf(expression?appiftrue[:...][:appiffalse[:...]])
```

Arguments

- `expression`
- `execapp`
    - `appiftrue`
    - `appiffalse`

See Also

Application_ExecIfTime

ExecIfTime()

Synopsis

Conditional application execution based on the current time.

Description

This application will execute the specified dialplan application, with optional arguments, if the current time matches the given time specification.

Syntax

```
ExecIfTime(timesweekdaysmdaysmonths[,timezone]?appargs)
```

Arguments

- `day_condition`
  - `times`
  - `weekdays`
  - `mdays`
  - `months`
  - `timezone`
- `appname`
  - `appargs`

See Also

Application_Exec
Application_TryExec

Application_ExitWhile

ExitWhile()

Synopsis

End a While loop.

Description

Exits a `While()` loop, whether or not the conditional has been satisfied.

Syntax

```
ExitWhile()
```

Arguments

See Also

Application_While
Application_EndWhile
Application_ContinueWhile

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ExtenSpy

ExtenSpy()

Synopsis

Listen to a channel, and optionally whisper into it.

Description

This application is used to listen to the audio from an Asterisk channel. This includes the audio coming in and out of the channel being spied on. Only channels created by outgoing calls for the specified extension will be selected for spying. If the optional context is not supplied, the current channel's context will be used.

While spying, the following actions may be performed:

- Dialing # cycles the volume level.

- Dialing * will stop spying and look for another channel to spy on.

The The *X* option supersedes the three features above in that if a valid single digit extension exists in the correct context ChanSpy will exit to it. This also disables choosing a channel based on `chanprefix` and a digit sequence.

Syntax

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

Arguments

- `exten`
    - `exten` - Specify extension.
    - `context` - Optionally specify a context, defaults to `default`.
- `options`
    - `b` - Only spy on channels involved in a bridged call.
    - `B` - Instead of whispering on a single channel barge in on both channels involved in the call.
    - `c`
        - `digit` - Specify a DTMF digit that can be used to spy on the next available channel.

- d - Override the typical numeric DTMF functionality and instead use DTMF to switch between spy modes.
  - 4 - spy mode
  - 5 - whisper mode
  - 6 - barge mode
- e - Enable **enforced** mode, so the spying channel can only monitor extensions whose name is in the *ext* : delimited list.
  - ext
- E - Exit when the spied-on channel hangs up.
- g - both both *grp* and SPYGROUP can contain either a single group or a colon-delimited list of groups, such as sales:support:accounting.
  - grp - Only spy on channels in which one or more of the groups listed in *grp* matches one or more groups from the SPYGROUP variable set on the channel to be spied upon.
- n - Say the name of the person being spied on if that person has recorded his/her name. If a context is specified, then that voicemail context will be searched when retrieving the name, otherwise the default context be used when searching for the name (i.e. if SIP/1000 is the channel being spied on and no mailbox is specified, then 1000 will be used when searching for the name).
  - mailbox
  - context
- o - Only listen to audio coming from this channel.
- q - Don't play a beep when beginning to spy on a channel, or speak the selected channel name.
- r - Record the session to the monitor spool directory. An optional base for the filename may be specified. The default is chanspy.
  - basename
- s - Skip the playback of the channel type (i.e. SIP, IAX, etc) when speaking the selected channel name.
- S - Stop when there are no more extensions left to spy on.
- v - Adjust the initial volume in the range from -4 to 4. A negative value refers to a quieter setting.
  - value
- w - Enable whisper mode, so the spying channel can talk to the spied-on channel.
- W - Enable private whisper mode, so the spying channel can talk to the spied-on channel but cannot listen to that channel.
- x
  - digit - Specify a DTMF digit that can be used to exit the application.
- X - Allow the user to exit ChanSpy to a valid single digit numeric extension in the current context or the context specified by the SPY_EXIT_CONTEXT channel variable. The name of the last channel that was spied on will be stored in the SPY_CHANNEL variable.
- 4 - spy mode
- 5 - whisper mode
- 6 - barge mode

See Also

Application_ChanSpy

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ExternalIVR

ExternalIVR()

Synopsis

Interfaces with an external IVR application.

Description

Either forks a process to run given command or makes a socket to connect to given host and starts a generator on the channel. The generator's play list is controlled by the external application, which can add and clear entries via simple commands issued over its stdout. The external application will receive all DTMF events received on the channel, and notification if the channel is hung up. The 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. For more information see doc/AST.pdf.

Syntax

```
ExternalIVR([arg1][,arg2][,options])
```

Arguments

- `command|ivr://host`
    - `arg1`
    - `arg2`
- `options`
    - `n` - Tells ExternalIVR() not to answer the channel.
    - `i` - Tells ExternalIVR() not to send a hangup and exit when the channel receives a hangup, instead it sends an `I` informative message meaning that the external application MUST hang up the call with an `H` command.
    - `d` - Tells ExternalIVR() to run on a channel that has been hung up and will not look for hangups. The external application must exit with an `E` command.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Festival

Festival()

Synopsis

Say text to the user.

Description

Connect to Festival, send the argument, get back the waveform, play it to the user, allowing any given interrupt keys to immediately terminate and return the value, or `any` to allow any number back (useful in dialplan).

Syntax

```
Festival(text[,intkeys])
```

Arguments

- `text`
- `intkeys`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Flash

Flash()

Flashes a DAHDI Trunk.

Description

Performs a flash on a DAHDI trunk. This can be used to access features provided on an incoming analogue circuit such as conference and call waiting. Use with SendDTMF() to perform external transfers.

Syntax

```
Flash()
```

Arguments

See Also

Application_SendDTMF

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_FollowMe

FollowMe()

Synopsis

Find-Me/Follow-Me application.

Description

This application performs Find-Me/Follow-Me functionality for the caller as defined in the profile matching the *followmeid* parameter in `followme.conf`. If the specified *followmeid* profile doesn't exist in `followme.conf`, execution will be returned to the dialplan and call execution will continue at the next priority.

Returns -1 on hangup.

Syntax

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

Arguments

- `followmeid`
- `options`
    - `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 or the callee has elected not to be reachable.
    - `d` - Disable the 'Please hold while we try to connect your call' announcement.

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ForkCDR

ForkCDR()

Synopsis

Forks the Call Data Record.

Description

Causes the Call Data Record to fork an additional cdr record starting from the time of the fork call. This new cdr record will be linked to end of the list of cdr records attached to the channel. The original CDR has a LOCKED flag set, which forces most cdr operations to skip it, except for the functions that set the answer and end times, which ignore the LOCKED flag. This allows all the cdr records in the channel to be 'ended' together when the channel is closed.

The CDR() func (when setting CDR values) normally ignores the LOCKED flag also, but has options to vary its behavior. The 'T' option (described below), can override this behavior, but beware the risks.

First, this app finds the last cdr record in the list, and makes a copy of it. This new copy will be the newly forked cdr record. Next, this new record is linked to the end of the cdr record list. Next, The new cdr record is RESET (unless you use an option to prevent this)

This means that:

1. All flags are unset on the cdr record

2. the start, end, and answer times are all set to zero.

3. the billsec and duration fields are set to zero.

4. the start time is set to the current time.

5. the disposition is set to NULL.

Next, unless you specified the v option, all variables will be removed from the original cdr record. Thus, the v option allows any CDR variables to be replicated to all new forked cdr records. Without the v option, the variables on the original are effectively moved to the new forked cdr record.

Next, if the s option is set, the provided variable and value are set on the original cdr record.

Next, if the a option is given, and the original cdr record has an answer time set, then the new forked cdr record will have its answer time set to its start time. If the old answer time were carried

forward, the answer time would be earlier than the start time, giving strange duration and billsec times.

If the `d` option was specified, the disposition is copied from the original cdr record to the new forked cdr. If the `D` option was specified, the destination channel field in the new forked CDR is erased. If the `e` option was specified, the 'end' time for the original cdr record is set to the current time. Future hang-up or ending events will not override this time stamp. If the `A` option is specified, the original cdr record will have it ANS_LOCKED flag set, which prevent future answer events from updating the original cdr record's disposition. Normally, an `ANSWERED` event would mark all cdr records in the chain as `ANSWERED`. If the `T` option is specified, the original cdr record will have its `DONT_TOUCH` flag set, which will force the cdr_answer, cdr_end, and cdr_setvar functions to leave that cdr record alone.

And, last but not least, the original cdr record has its LOCKED flag set. Almost all internal CDR functions (except for the funcs that set the end, and answer times, and set a variable) will honor this flag and leave a LOCKED cdr record alone. This means that the newly created forked cdr record will be affected by events transpiring within Asterisk, with the previously noted exceptions.

Syntax

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

Arguments

- `options`
  - `a` - Update the answer time on the NEW CDR just after it's been inited. The new CDR may have been answered already. The reset that forkcdr does will erase the answer time. This will bring it back, but the answer time will be a copy of the fork/start time. It will only do this if the initial cdr was indeed already answered.
  - `A` - Lock the original CDR against the answer time being updated. This will allow the disposition on the original CDR to remain the same.
  - `d` - Copy the disposition forward from the old cdr, after the init.
  - `D` - Clear the `dstchannel` on the new CDR after reset.
  - `e` - End the original CDR. Do this after all the necessary data is copied from the original CDR to the new forked CDR.
  - `r` - Do **NOT** reset the new cdr.
  - `s(name=val)` - Set the CDR var *name* in the original CDR, with value *val*.
  - `T` - Mark the original CDR with a DONT_TOUCH flag. setvar, answer, and end cdr funcs will obey this flag; normally they don't honor the LOCKED flag set on the original CDR record. Using this flag may cause CDR's not to have their end times updated! It is suggested that if you specify this flag, you might wish to use the Using this flag may cause CDR's not to have their end times updated! It is suggested that if you specify this flag, you might wish to use the `e` flag as well!.
  - `v` - When the new CDR is forked, it gets a copy of the vars attached to the current CDR. The vars attached to the original CDR are removed unless this option is specified.

See Also

[Function_CDR](#)
[Application_NoCDR](#)
[Application_ResetCDR](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_GetCPEID

GetCPEID()

Get ADSI CPE ID.

Description

Obtains and displays ADSI CPE ID and other information in order to properly setup `dahdi.conf` for on-hook operations.

Syntax

```
GetCPEID()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Gosub

Gosub()

Synopsis

Jump to label, saving return address.

Description

Jumps to the label specified, saving the return address.

Syntax

```
Gosub([context[,exten,arg1[,...][,argN]]])
```

Arguments

- context
- exten
- priority
    - arg1
    - argN

See Also

Application_GosubIf
Application_Macro
Application_Goto
Application_Return
Application_StackPop

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_GosubIf

GosubIf()

Synopsis

Conditionally jump to label, saving return address.

Description

If the condition is true, then jump to labeliftrue. If false, jumps to labeliffalse, if specified. In either case, a jump saves the return point in the dialplan, to be returned to with a Return.

Syntax

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

Arguments

- condition
- destination
    - labeliftrue
    - labeliffalse

See Also

Application_Gosub
Application_Return
Application_MacroIf
Function_IF
Application_GotoIf

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Goto

Goto()

Synopsis

Jump to a particular priority, extension, or context.

Description

This application will set the current context, extension, and priority in the channel structure. After it completes, the pbx engine will continue dialplan execution at the specified location. If no specific *extension*, or *extension* and *context*, are specified, then this application will just set the

specified *priority* of the current extension.

At least a *priority* is required as an argument, or the goto will return a `-1`, and the channel and call will be terminated.

If the location that is put into the channel information is bogus, and asterisk cannot find that location in the dialplan, then the execution engine will try to find and execute the code in the `i` (invalid) extension in the current context. If that does not exist, it will try to execute the `h` extension. If neither the `h` nor `i` extensions have been defined, the channel is hung up, and the execution of instructions on the channel is terminated. What this means is that, for example, you specify a context that does not exist, then it will not be possible to find the `h` or `i` extensions, and the call will terminate!

Syntax

```
Goto([context[,extensions,priority]])
```

Arguments

- `context`
- `extensions`
- `priority`

See Also

[Application_GotoIf](#)
[Application_GotoIfTime](#)
[Application_Gosub](#)
[Application_Macro](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r324178.

Application_GotoIf

GotoIf()

Synopsis

Conditional goto.

Description

This application will set the current context, extension, and priority in the channel structure based on the evaluation of the given condition. After this application completes, the pbx engine will continue dialplan execution at the specified location in the dialplan. The labels are specified with the same syntax as used within the Goto application. If the label chosen by the condition is omitted, no jump is performed, and the execution passes to the next instruction. If the target location is bogus, and does not exist, the execution engine will try to find and execute the code in the `i` (invalid) extension in the current context. If that does not exist, it will try to execute the `h` extension. If neither the `h` nor `i` extensions have been defined, the channel is hung up, and the

execution of instructions on the channel is terminated. Remember that this command can set the current context, and if the context specified does not exist, then it will not be able to find any 'h' or 'i' extensions there, and the channel and call will both be terminated!.

Syntax

```
GotoIf(condition?[labeliftrue][:labeliffalse])
```

Arguments

- `condition`
- `destination`
  - `labeliftrue` - Continue at *labeliftrue* if the condition is true.
  - `labeliffalse` - Continue at *labeliffalse* if the condition is false.

See Also

Application_Goto
Application_GotoIfTime
Application_GosubIf
Application_MacroIf

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r324178.

Application_GotoIfTime

GotoIfTime()

Synopsis

Conditional Goto based on the current time.

Description

This application will set the context, extension, and priority in the channel structure based on the evaluation of the given time specification. After this application completes, the pbx engine will continue dialplan execution at the specified location in the dialplan. If the current time is within the given time specification, the channel will continue at *labeliftrue*. Otherwise the channel will continue at *labeliffalse*. If the label chosen by the condition is omitted, no jump is performed, and execution passes to the next instruction. If the target jump location is bogus, the same actions would be taken as for `Goto`. Further information on the time specification can be found in examples illustrating how to do time-based context includes in the dialplan.

Syntax

```
GotoIfTime(timesweekdaysmdaysmonths[,timezone]?[labeliftrue][:labeliff
```

Arguments

- `condition`

- times
- weekdays
- mdays
- months
- timezone
- destination
  - labeliftrue
  - labeliffalse

See Also

[Application_GotoIf](#)
[Function_IFTIME](#)
[Function_TESTTIME](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Hangup

Hangup()

Synopsis

Hang up the calling channel.

Description

This application will hang up the calling channel.

Syntax

```
Hangup([causecode])
```

Arguments

- `causecode` - If a *causecode* is given the channel's hangup cause will be set to the given value.

See Also

[Application_Answer](#)
[Application_Busy](#)
[Application_Congestion](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_IAX2Provision

IAX2Provision()

Synopsis

Provision a calling IAXy with a given template.

Provisions the calling IAXy (assuming the calling entity is in fact an IAXy) with the given *template*. Returns `-1` on error or `0` on success.

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

Arguments

- `template` - If not specified, defaults to `default`.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ICES

ICES()

Encode and stream using 'ices'.

Streams to an icecast server using ices (available separately). A configuration file must be supplied for ices (see contrib/asterisk-ices.xml).

ICES version 2 client and server required.ICES version 2 client and server required.

```
ICES(config)
```

Arguments

- `config` - ICES configuration file.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ImportVar

ImportVar()

Synopsis

Import a variable from a channel into a new variable.

Description

This application imports a *variable* from the specified *channel* (as opposed to the current one) and stores it as a variable (*newvar*) in the current channel (the channel that is calling this application). Variables created by this application have the same inheritance properties as those created with the `Set` application.

Syntax

```
ImportVar(newvar=channelnamevariable)
```

Arguments

- newvar
- vardata
    - channelname
    - variable

See Also

[Application_Set](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Incomplete

Incomplete()

Synopsis

Returns AST_PBX_INCOMPLETE value.

Description

Signals the PBX routines that the previous matched extension is incomplete and that further input should be allowed before matching can be considered to be complete. Can be used within a pattern match when certain criteria warrants a longer match.

Syntax

```
Incomplete([n])
```

Arguments

- n - If specified, then Incomplete will not attempt to answer the channel first. Most channel types need to be in Answer state in order to

receive DTMF.Most channel types need to be in Answer state in order to receive DTMF.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_IVRDemo

IVRDemo()

Synopsis

IVR Demo Application.

Description

This is a skeleton application that shows you the basic structure to create your own asterisk applications and demonstrates the IVR demo.

Syntax

```
IVRDemo(filename)
```

Arguments

- `filename`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_JabberJoin

JabberJoin()

Synopsis

Join a chat room

Description

Allows Asterisk to join a chat room.

Syntax

```
JabberJoin(Jabber,RoomJID[,Nickname])
```

Arguments

- `Jabber` - Client or transport Asterisk uses to connect to Jabber.
- `RoomJID` - XMPP/Jabber JID (Name) of chat room.
- `Nickname` - The nickname Asterisk will use in the chat room. If a different nickname is supplied to an already joined room, the old nick will be changed to the new one.If a different nickname is supplied to an already joined room, the old nick will be changed to the new one.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_JabberLeave

JabberLeave()

Synopsis

Leave a chat room

Description

Allows Asterisk to leave a chat room.

Syntax

```
JabberLeave(Jabber,RoomJID[,Nickname])
```

Arguments

- `Jabber` - Client or transport Asterisk uses to connect to Jabber.
- `RoomJID` - XMPP/Jabber JID (Name) of chat room.
- `Nickname` - The nickname Asterisk uses in the chat room.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_JabberSend

JabberSend()

Synopsis

Sends an XMPP message to a buddy.

Description

Sends the content of *message* as text message from the given *account* to the buddy identified by *jid*

Example: JabberSend(asterisk,bob@domain.com,Hello world) sends "Hello world" to *bob@domain.com* as an XMPP message from the account *asterisk*, configured in jabber.conf.

```
JabberSend(account,jid,message)
```

Arguments

- `account` - The local named account to listen on (specified in jabber.conf)
- `jid` - Jabber ID of the buddy to send the message to. It can be a bare JID (username@domain) or a full JID (username@domain/resource).
- `message` - The message to send.

See Also

[Function_JABBER_STATUS](#)
[Function_JABBER_RECEIVE](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_JabberSendGroup

JabberSendGroup()

Synopsis

Send a Jabber Message to a specified chat room

Description

Allows user to send a message to a chat room via XMPP.

To be able to send messages to a chat room, a user must have previously joined it. Use the To be able to send messages to a chat room, a user must have previously joined it. Use the *JabberJoin* function to do so.

Syntax

```
JabberSendGroup(Jabber,RoomJID,Message[,Nickname])
```

Arguments

- `Jabber` - Client or transport Asterisk uses to connect to Jabber.
- `RoomJID` - XMPP/Jabber JID (Name) of chat room.
- `Message` - Message to be sent to the chat room.
- `Nickname` - The nickname Asterisk uses in the chat room.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_JabberStatus

JabberStatus()

Synopsis

Retrieve the status of a jabber list member

Description

This application is deprecated. Please use the JABBER_STATUS() function instead.

Retrieves the numeric status associated with the specified buddy *JID*. The return value in the *Variable* will be one of the following.

Online.

Chatty.

Away.

Extended Away.

Do Not Disturb.

Offline.

Not In Roster.

Syntax

```
JabberStatus(Jabber,JID,Variable)
```

Arguments

- `Jabber` - Client or transport Asterisk users to connect to Jabber.
- `JID` - XMPP/Jabber JID (Name) of recipient.
- `Variable` - Variable to store the status of requested user.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_JACK

JACK()

Synopsis

Jack Audio Connection Kit

Description

When executing this application, two jack ports will be created; one input and one output. Other applications can be hooked up to these ports to access audio coming from, or being send to the channel.

```
JACK([options])
```

Arguments

- `options`
  - `s`
    - `name` - Connect to the specified jack server name
  - `i`
    - `name` - Connect the output port that gets created to the specified jack input port
  - `o`
    - `name` - Connect the input port that gets created to the specified jack output port
  - `c`
    - `name` - By default, Asterisk will use the channel name for the jack client name. Use this option to specify a custom client name.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Log

Log()

Synopsis

Send arbitrary text to a selected log level.

Description

Sends an arbitrary text message to a selected log level.

Syntax

```
Log(level,message)
```

Arguments

- `level` - Level must be one of `ERROR`, `WARNING`, `NOTICE`, `DEBUG`, `VERBOSE` or `DTMF`.
- `message` - Output text message.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Macro

Macro()

Macro Implementation.

Description

Executes a macro using the context macro- *name*, 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 The calling extension, context, and priority are stored in `None` - `MACRO_EXTEN`, The calling extension, context, and priority are stored in `None` - `MACRO_CONTEXT` and The calling extension, context, and priority are stored in `None` - `MACRO_PRIORITY` respectively. Arguments become The calling extension, context, and priority are stored in `None` - `ARG1`, The calling extension, context, and priority are stored in `None` - `ARG2`, etc in the macro context.

If you Goto out of the Macro context, the Macro will terminate and control will be returned at the location of the Goto.

If If `None` - `MACRO_OFFSET` is set at termination, Macro will attempt to continue at priority MACRO_OFFSET + N + 1 if such a step exists, and N + 1 otherwise.

Because of the way Macro is implemented (it executes the priorities contained within it via sub-engine), and a fixed per-thread memory stack allowance, macros are limited to 7 levels of nesting (macro calling macro calling macro, etc.); It may be possible that stack-intensive applications in deeply nested macros could cause asterisk to crash earlier than this limit. It is advised that if you need to deeply nest macro calls, that you use the Gosub application (now allows arguments like a Macro) with explicit Return() calls instead.

Use of the application `WaitExten` within a macro will not function as expected. Please use the `Read` application in order to read DTMF from a channel currently executing a macro.

Syntax

```
Macro(name[,arg1[,arg2[,...]]])
```

Arguments

- `name` - The name of the macro
- `args`
  - `arg1`
  - `arg2`

See Also

Application_MacroExit
Application_Goto
Application_Gosub

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MacroExclusive

MacroExclusive()

Synopsis

Exclusive Macro Implementation.

Description

Executes macro defined in the context macro- *name*. Only one call at a time may run the macro. (we'll wait if another call is busy executing in the Macro)

Arguments and return values as in application Macro()

Use of the application `WaitExten` within a macro will not function as expected. Please use the `Read` application in order to read DTMF from a channel currently executing a macro.

Syntax

```
MacroExclusive(name[,arg1[,arg2[,...]]])
```

Arguments

- `name` - The name of the macro
- `arg1`
- `arg2`

See Also

Application_Macro

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MacroExit

MacroExit()

Synopsis

Exit from Macro.

Description

Causes the currently running macro to exit as if it had ended normally by running out of priorities to execute. If used outside a macro, will likely cause unexpected behavior.

```
MacroExit()
```

Arguments

Application_Macro

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MacroIf

MacroIf()

Conditional Macro implementation.

Executes macro defined in *macroiftrue* if *expr* is true (otherwise *macroiffalse* if provided)

Arguments and return values as in application Macro()

Use of the application `WaitExten` within a macro will not function as expected. Please use the `Read` application in order to read DTMF from a channel currently executing a macro.

```
MacroIf(expr?macroiftrue[:macroiffalse])
```

Arguments

- `expr`
- `destination`
    - `macroiftrue`
    - `macroiffalse`

Application_GotoIf
Application_GosubIf
Function_IF

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MailboxExists

MailboxExists()

Synopsis

Check to see if Voicemail mailbox exists.

Description

Check to see if the specified *mailbox* exists. If no voicemail *context* is specified, the `default` context will be used.

This application will set the following channel variable upon completion:

- `VMBOXEXISTSSTATUS` - This will contain the status of the execution of the MailboxExists application. Possible values include:
    - `SUCCESS`
    - `FAILED`

Syntax

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

Arguments

- `mailbox`
    - `mailbox`
    - `context`
- `options` - None options.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MeetMe

MeetMe()

Synopsis

MeetMe conference bridge.

Description

Enters the user into a specified MeetMe conference. If the *confno* is omitted, the user will be prompted to enter one. User can exit the conference by hangup, or if the `p` option is specified, by pressing `#`.

The DAHDI kernel modules and at least one hardware driver (or dahdi_dummy) must be present for conferencing to operate properly. In addition, the chan_dahdi channel driver must be loaded for the The DAHDI kernel modules and at least one hardware driver (or dahdi_dummy) must be present for conferencing to operate properly. In addition, the chan_dahdi channel driver must be loaded for the `i` and `r` options to operate at all.

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

Arguments

- `confno` - The conference number
- `options`
  - `a` - Set admin mode.
  - `A` - Set marked mode.
  - `b` - Run AGI script specified in `MEETME_AGI_BACKGROUND` Default: `conf-background.agi`. This does not work with non-DAHDI channels in the same conference).This does not work with non-DAHDI channels in the same conference).
  - `c` - Announce user(s) count on joining a conference.
  - `C` - Continue in dialplan when kicked out of conference.
  - `d` - Dynamically add conference.
  - `D` - Dynamically add conference, prompting for a PIN.
  - `e` - Select an empty conference.
  - `E` - Select an empty pinless conference.
  - `F` - Pass DTMF through the conference.
  - `G` - Play an intro announcement in conference.
    - `x` - The file to playback
  - `i` - Announce user join/leave with review.
  - `I` - Announce user join/leave without review.
  - `l` - Set listen only mode (Listen only, no talking).
  - `m` - Set initially muted.
  - `M` - Enable music on hold when the conference has a single caller. Optionally, specify a musiconhold class to use. If one is not provided, it will use the channel's currently set music class, or `default`.
    - `class`
  - `o` - Set talker optimization - treats talkers who aren't speaking as being muted, meaning (a) No encode is done on transmission and (b) Received audio that is not registered as talking is omitted causing no buildup in background noise.
  - `p` - Allow user to exit the conference by pressing # (default) or any of the defined keys. If keys contain * this will override option `s`. The key used is set to channel variable `MEETME_EXIT_KEY`.
    - `keys`
  - `P` - Always prompt for the pin even if it is specified.
  - `q` - Quiet mode (don't play enter/leave sounds).
  - `r` - Record conference (records as `MEETME_RECORDINGFILE` using format `MEETME_RECORDINGFORMAT`. Default filename is `meetme-conf-rec-${CONFNO}-${UNIQEID}` and the default format is wav.
  - `s` - Present menu (user or admin) when * is received (send to menu).
  - `t` - Set talk only mode. (Talk only, no listening).
  - `T` - Set talker detection (sent to manager interface and meetme list).
  - `w` - Wait until the marked user enters the conference.
    - `secs`
  - `x` - Close the conference when last marked user exits
  - `X` - Allow user to exit the conference by entering a valid single digit extension `MEETME_EXIT_CONTEXT` or the current context if that variable is not defined.
  - `1` - Do not play message when first person enters
  - `S` - Kick the user *x* seconds **after** he entered into the conference.
    - `x`
  - `L` - Limit the conference to *x* ms. Play a warning when *y* ms are left. Repeat the warning every *z* ms. The following special variables can be used with this option: File to play when time is up. File to play as warning if *y* is defined. The default is to say the time remaining.
    - `x`
    - `y`
    - `z`
- `pin`

See Also

[Application_MeetMeCount](#)
[Application_MeetMeAdmin](#)
[Application_MeetMeChannelAdmin](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MeetMeAdmin

MeetMeAdmin()

Synopsis

MeetMe conference administration.

Description

Run admin *command* for conference *confno*.

Will additionally set the variable Will additionally set the variable `None` - `MEETMEADMINSTATUS` with one of the following values:

- `MEETMEADMINSTATUS` -
    - `NOPARSE` - Invalid arguments.
    - `NOTFOUND` - User specified was not found.
    - `FAILED` - Another failure occurred.
    - `OK` - The operation was completed successfully.

Syntax

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

Arguments

- `confno`
- `command`
    - `e` - Eject last user that joined.
    - `E` - Extend conference end time, if scheduled.
    - `k` - Kick one user out of conference.
    - `K` - Kick all users out of conference.
    - `l` - Unlock conference.
    - `L` - Lock conference.
    - `m` - Unmute one user.
    - `M` - Mute one user.
    - `n` - Unmute all users in the conference.
    - `N` - Mute all non-admin users in the conference.
    - `r` - Reset one user's volume settings.
    - `R` - Reset all users volume settings.
    - `s` - Lower entire conference speaking volume.
    - `S` - Raise entire conference speaking volume.
    - `t` - Lower one user's talk volume.
    - `T` - Raise one user's talk volume.
    - `u` - Lower one user's listen volume.
    - `U` - Raise one user's listen volume.
    - `v` - Lower entire conference listening volume.
    - `V` - Raise entire conference listening volume.
- `user`

See Also

Application_MeetMe

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MeetMeChannelAdmin

MeetMeChannelAdmin()

Synopsis

MeetMe conference Administration (channel specific).

Description

Run admin *command* for a specific *channel* in any conference.

Syntax

```
MeetMeChannelAdmin(channel,command)
```

Arguments

- channel
- command
    - k - Kick the specified user out of the conference he is in.
    - m - Unmute the specified user.
    - M - Mute the specified user.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r324178.

Application_MeetMeCount

MeetMeCount()

Synopsis

MeetMe participant count.

Description

Plays back the number of users in the specified MeetMe conference. If *var* is specified, playback will be skipped and the value will be returned in the variable. Upon application completion, MeetMeCount will hangup the channel, unless priority n+1 exists, in which case priority progress will continue.

Syntax

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

Arguments

- `confno` - Conference number.
- `var`

Application_MeetMe

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Milliwatt

Milliwatt()

Synopsis

Generate a Constant 1004Hz tone at 0dbm (mu-law).

Description

Previous versions of this application generated the tone at 1000Hz. If for some reason you would prefer that behavior, supply the `o` option to get the old behavior.

Syntax

```
Milliwatt([options])
```

Arguments

- `options`
  - `o` - Generate the tone at 1000Hz like previous version.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MinivmAccMess

MinivmAccMess()

Synopsis

Record account specific messages.

Description

This application is part of the Mini-Voicemail system, configured in `minivm.conf`.

Use this application to record account specific audio/video messages for busy, unavailable and temporary messages.

Account specific directories will be created if they do not exist.

- MVM_ACCMESS_STATUS - This is the result of the attempt to record the specified greeting. FAILED is set if the file can't be created.
    - SUCCESS
    - FAILED

Syntax

```
MinivmAccMess(usernamedomain[,options])
```

Arguments

- mailbox
    - username - Voicemail username
    - domain - Voicemail domain
- options
    - u - Record the unavailable greeting.
    - b - Record the busy greeting.
    - t - Record the temporary greeting.
    - n - Account name.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MinivmDelete

MinivmDelete()

Synopsis

Delete Mini-Voicemail voicemail messages.

Description

This application is part of the Mini-Voicemail system, configured in minivm.conf.

It deletes voicemail file set in MVM_FILENAME or given filename.

- MVM_DELETE_STATUS - This is the status of the delete operation.
    - SUCCESS
    - FAILED

Syntax

```
MinivmDelete(filename)
```

Arguments

- filename - File to delete

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MinivmGreet

MinivmGreet()

Synopsis

Play Mini-Voicemail prompts.

Description

This application is part of the Mini-Voicemail system, configured in minivm.conf.

MinivmGreet() plays default prompts or user specific prompts for an account.

Busy and unavailable messages can be choosen, but will be overridden if a temporary message exists for the account.

- `MVM_GREET_STATUS` - This is the status of the greeting playback.
    - `SUCCESS`
    - `USEREXIT`
    - `FAILED`

Syntax

```
MinivmGreet(usernamedomain[,options])
```

Arguments

- `mailbox`
    - `username` - Voicemail username
    - `domain` - Voicemail domain
- `options`
    - `b` - Play the `busy` greeting to the calling party.
    - `s` - Skip the playback of instructions for leaving a message to the calling party.
    - `u` - Play the `unavailable` greeting.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MinivmMWI

MinivmMWI()

Synopsis

Send Message Waiting Notification to subscriber(s) of mailbox.

Description

This application is part of the Mini-Voicemail system, configured in `minivm.conf`.

MinivmMWI is used to send message waiting indication to any devices whose channels have subscribed to the mailbox passed in the first parameter.

```
MinivmMWI(usernamedomain,urgent,new,old)
```

Arguments

- `mailbox`
    - `username` - Voicemail username
    - `domain` - Voicemail domain
- `urgent` - Number of urgent messages in mailbox.
- `new` - Number of new messages in mailbox.
- `old` - Number of old messages in mailbox.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MinivmNotify

MinivmNotify()

Synopsis

Notify voicemail owner about new messages.

Description

This application is part of the Mini-Voicemail system, configured in minivm.conf.

MiniVMnotify forwards messages about new voicemail to e-mail and pager. If there's no user account for that address, a temporary account will be used with default options (set in `minivm.conf`).

If the channel variable If the channel variable `None` - `MVM_COUNTER` is set, this will be used in the message file name and available in the template for the message.

If no template is given, the default email template will be used to send email and default pager template to send paging message (if the user account is configured with a paging address.

- `MVM_NOTIFY_STATUS` - This is the status of the notification attempt
    - `SUCCESS`
    - `FAILED`

Syntax

```
MinivmNotify(usernamedomain[,options])
```

Arguments

- mailbox
    - `username` - Voicemail username
    - `domain` - Voicemail domain
- `options`
    - `template` - E-mail template to use for voicemail notification

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MinivmRecord

MinivmRecord()

Synopsis

Receive Mini-Voicemail and forward via e-mail.

Description

This application is part of the Mini-Voicemail system, configured in `minivm.conf`

MiniVM records audio file in configured format and forwards message to e-mail and pager.

If there's no user account for that address, a temporary account will be used with default options.

The recorded file name and path will be stored in The recorded file name and path will be stored in `None` - `MVM_FILENAME` and the duration of the message will be stored in The recorded file name and path will be stored in `None` - `MVM_DURATION`

If the caller hangs up after the recording, the only way to send the message and clean up is to execute in the If the caller hangs up after the recording, the only way to send the message and clean up is to execute in the `h` extension. The application will exit if any of the following DTMF digits are received and the requested extension exist in the current context.

- `MVM_RECORD_STATUS` - This is the status of the record operation
    - `SUCCESS`
    - `USEREXIT`
    - `FAILED`

Syntax

```
MinivmRecord(usernamedomain[,options])
```

Arguments

- `mailbox`
    - `username` - Voicemail username
    - `domain` - Voicemail domain
- `options`
    - `0` - Jump to the `o` extension in the current dialplan context.
    - `*` - Jump to the `a` extension in the current dialplan context.
    - `g` - Use the specified amount of gain when recording the voicemail message. The units are whole-number decibels (dB).
        - `gain` - Amount of gain to use

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MixMonitor

MixMonitor()

Synopsis

Record a call and mix the audio during the recording. Use of StopMixMonitor is required to guarantee the audio file is available for processing during dialplan execution.

Description

Records the audio on the current channel to the specified file.

- `MIXMONITOR_FILENAME` - Will contain the filename used to record.

Syntax

```
MixMonitor(filenameextension[,options[,command]])
```

Arguments

- `file`
    - `filename` - If *filename* is an absolute path, uses that path, otherwise creates the file in the configured monitoring directory from `asterisk.conf`.
    - `extension`
- `options`
    - `a` - Append to the file instead of overwriting it.
    - `b` - Only save audio to the file while the channel is bridged. Does not include conferences or sounds played to each bridged partyDoes not include conferences or sounds played to each bridged party If you utilize this option inside a Local channel, you must make sure the Local channel is not optimized away. To do this, be sure to call your Local channel with the If you utilize this option inside a Local channel, you must make sure the Local channel is not optimized away. To do this, be sure to call your Local channel with the `/n` option. For example: Dial(Local/start@mycontext/n)
    - `v` - Adjust the **heard** volume by a factor of *x* (range `-4` to `4`)
        - `x`
    - `V` - Adjust the **spoken** volume by a factor of *x* (range `-4` to `4`)
        - `x`
    - `W` - Adjust both, **heard and spoken** volumes by a factor of *x* (range `-4` to `4`)
        - `x`
- `command` - Will be executed when the recording is over. Any strings matching `^{x}` will be unescaped to `x`. All variables will be evaluated at the time MixMonitor is called.

Application_Monitor
Application_StopMixMonitor
Application_PauseMonitor
Application_UnpauseMonitor

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Monitor

Monitor()

Monitor a channel.

Description

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

By default, files are stored to `/var/spool/asterisk/monitor/`. Returns `-1` if monitor files can't be opened or if the channel is already monitored, otherwise `0`.

Syntax

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

Arguments

- `file_format`
  - `file_format` - optional, if not set, defaults to `wav`
  - `urlbase`
- `fname_base` - if set, changes the filename used to the one specified.
- `options`
  - `m` - when the recording ends mix the two leg files into one and delete the two leg files. If the variable `MONITOR_EXEC` is set, the application referenced in it will be executed instead of soxmix/sox and the raw leg files will NOT be deleted automatically. soxmix/sox or `MONITOR_EXEC` is handed 3 arguments, the two leg files and a target mixed file name which is the same as the leg file names only without the in/out designator. If `MONITOR_EXEC_ARGS` is set, the contents will be passed on as additional arguments to `MONITOR_EXEC`. Both `MONITOR_EXEC` and the Mix flag can be set from the administrator interface.
  - `b` - Don't begin recording unless a call is bridged to another channel.
  - `i` - Skip recording of input stream (disables `m` option).
  - `o` - Skip recording of output stream (disables `m` option).

See Also

Application_StopMonitor

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Morsecode

Morsecode()

Synopsis

Plays morse code.

Description

Plays the Morse code equivalent of the passed string.

This application uses the following variables:

- `MORSEDITLEN` - Use this value in (ms) for length of dit
- `MORSETONE` - The pitch of the tone in (Hz), default is 800

Syntax

```
Morsecode(string)
```

Arguments

- `string` - String to playback as morse code to channel

See Also

Application_SayAlpha
Application_SayPhonetic

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MP3Player

MP3Player()

Synopsis

Play an MP3 file or M3U playlist file or stream.

Description

Executes mpg123 to play the given location, which typically would be a mp3 filename or m3u playlist filename or a URL. Please read http://en.wikipedia.org/wiki/M3U to see how M3U playlist file format is like, Example usage would be exten => 1234,1,MP3Player(/var/lib/asterisk/playlist.m3u) User can exit by pressing any key on the dialpad, or by hanging up.

Syntax

```
MP3Player(Location)
```

Arguments

- `Location` - Location of the file to be played. (argument passed to mpg123)

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MSet

MSet()

Synopsis

Set channel variable(s) or function value(s).

Description

This function can be used to set the value of channel variables or dialplan functions. When setting variables, if the variable name is prefixed with {}, *the variable will be inherited into channels created from the current channel If the variable name is prefixed with _*, the variable will be inherited into channels created from the current channel and all children channels. MSet behaves in a similar fashion to the way Set worked in 1.2/1.4 and is thus prone to doing things that you may not expect. For example, it strips surrounding double-quotes from the right-hand side (value). If you need to put a separator character (comma or vert-bar), you will need to escape them by inserting a backslash before them. Avoid its use if possible.

Syntax

```
MSet(name1value1[,name2value2[,...]])
```

Arguments

- set1
    - name1
    - value1
- set2
    - name2
    - value2

See Also

Application_Set

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_MusicOnHold

MusicOnHold()

Synopsis

Play Music On Hold indefinitely.

Description

Plays hold music specified by class. If omitted, the default music source for the channel will be used. Change the default class with Set(CHANNEL(musicclass)=...). If duration is given, hold music will be played specified number of seconds. If duration is ommited, music plays

indefinitely. Returns `0` when done, `-1` on hangup.

Syntax

```
MusicOnHold(class[,duration])
```

Arguments

- `class`
- `duration`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_NBScat

NBScat()

Synopsis

Play an NBS local stream.

Description

Executes nbscat to listen to the local NBS stream. User can exit by pressing any key.

Syntax

```
NBScat()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_NoCDR

NoCDR()

Synopsis

Tell Asterisk to not maintain a CDR for the current call

Description

This application will tell Asterisk not to maintain a CDR for the current call.

Syntax

```
NoCDR()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_NoOp

NoOp()

Synopsis

Do Nothing (No Operation).

Description

This application does nothing. However, it is useful for debugging purposes.

This method can be used to see the evaluations of variables or functions without having any effect.

Syntax

```
NoOp([text])
```

Arguments

- `text` - Any text provided can be viewed at the Asterisk CLI.

See Also

Application_Verbose
Application_Log

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ODBC_Commit

ODBC_Commit()

Synopsis

Commits a currently open database transaction.

Description

Commits the database transaction specified by *transaction ID* or the current active transaction, if not specified.

```
ODBC_Commit([transaction ID])
```

Arguments

- `transaction ID`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ODBC_Rollback

ODBC_Rollback()

Rollback a currently open database transaction.

Rolls back the database transaction specified by *transaction ID* or the current active transaction, if not specified.

```
ODBC_Rollback([transaction ID])
```

Arguments

- `transaction ID`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ODBCFinish

ODBCFinish()

Clear the resultset of a sucessful multirow query.

For queries which are marked as mode=multirow, this will clear any remaining rows of the specified resultset.

```
ODBCFinish(result-id)
```

Arguments

- `result-id`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Originate

Originate()

Originate a call.

This application originates an outbound call and connects it to a specified extension or application. This application will block until the outgoing call fails or gets answered. At that point, this application will exit with the status variable set and dialplan processing will continue.

This application sets the following channel variable before exiting:

- `ORIGINATE_STATUS` - This indicates the result of the call origination.
  - `FAILED`
  - `SUCCESS`
  - `BUSY`
  - `CONGESTION`
  - `HANGUP`
  - `RINGING`
  - `UNKNOWN` - In practice, you should never see this value. Please report it to the issue tracker if you ever see it.

```
Originate(tech_data,type,arg1[,arg2[,arg3]])
```

Arguments

- `tech_data` - Channel technology and data for creating the outbound channel. For example, SIP/1234.
- `type` - This should be `app` or `exten`, depending on whether the outbound channel should be connected to an application or extension.
- `arg1` - If the type is `app`, then this is the application name. If the type is `exten`, then this is the context that the channel will be sent to.
- `arg2` - If the type is `app`, then this is the data passed as arguments to the application. If the type is `exten`, then this is the extension that the channel will be sent to.
- `arg3` - If the type is `exten`, then this is the priority that the channel is sent to. If the type is `app`, then this parameter is ignored.

Application_OSPAuth

OSPAuth()

Synopsis

OSP Authentication.

Description

Authenticate a call by OSP.

Input variables:

- `OSPINPEERIP` - The last hop IP address.
- `OSPINTOKEN` - The inbound OSP token.

Output variables:

- `OSPINHANDLE` - The inbound call OSP transaction handle.
- `OSPINTIMELIMIT` - The inbound call duration limit in seconds.

This application sets the following channel variable upon completion:

- `OSPAUTHSTATUS` - The status of OSPAuth attempt as a text string, one of
  - `SUCCESS`
  - `FAILED`
  - `ERROR`

Syntax

```
OSPAuth([provider[,options]])
```

Arguments

- `provider` - The name of the provider that authenticates the call.
- `options` - Reserverd.

See Also

Application_OSPLookup
Application_OSPNext
Application_OSPFinish

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_OSPFinish

OSPFinish()

Synopsis

Report OSP entry.

Description

Report call state.

Input variables:

- `OSPINHANDLE` - The inbound call OSP transaction handle.
- `OSPOUTHANDLE` - The outbound call OSP transaction handle.
- `OSPAUTHSTATUS` - The OSPAuth status.
- `OSPLOOKUPSTATUS` - The OSPLookup status.
- `OSPNEXTSTATUS` - The OSPNext status.
- `OSPINAUDIOQOS` - The inbound call leg audio QoS string.
- `OSPOUTAUDIOQOS` - The outbound call leg audio QoS string.

This application sets the following channel variable upon completion:

- `OSPFINISHSTATUS` - The status of the OSPFinish attempt as a text string, one of
    - `SUCCESS`
    - `FAILED`
    - `ERROR`

Syntax

```
OSPFinish([cause[,options]])
```

Arguments

- `cause` - Hangup cause.
- `options` - Reserved.

See Also

Application_OSPAuth
Application_OSPLookup
Application_OSPNext

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_OSPLookup

OSPLookup()

Synopsis

Lookup destination by OSP.

Description

Looks up destination via OSP.

Input variables:

- `OSPINACTUALSRC` - The actual source device IP address in indirect mode.
- `OSPINPEERIP` - The last hop IP address.
- `OSPINHANDLE` - The inbound call OSP transaction handle.
- `OSPINTIMELIMIT` - The inbound call duration limit in seconds.
- `OSPINNETWORKID` - The inbound source network ID.
- `OSPINNPRN` - The inbound routing number.
- `OSPINNPCIC` - The inbound carrier identification code.
- `OSPINNPDI` - The inbound number portability database dip indicator.
- `OSPINSPID` - The inbound service provider identity.
- `OSPINOCN` - The inbound operator company number.
- `OSPINSPN` - The inbound service provider name.
- `OSPINALTSPN` - The inbound alternate service provider name.
- `OSPINMCC` - The inbound mobile country code.
- `OSPINMNC` - The inbound mobile network code.
- `OSPINTOHOST` - The inbound To header host part.
- `OSPINDIVUSER` - The inbound Diversion header user part.
- `OSPINDIVHOST` - The inbound Diversion header host part.
- `OSPINCUSTOMINFOn` - The inbound custom information, where `n` is the index beginning with `1` upto `8`.

Output variables:

- `OSPOUTHANDLE` - The outbound call OSP transaction handle.
- `OSPOUTTECH` - The outbound channel technology for the call.
- `OSPDESTINATION` - The outbound destination IP address.
- `OSPOUTCALLING` - The outbound calling number.
- `OSPOUTCALLED` - The outbound called number.
- `OSPOUTNETWORKID` - The outbound destination network ID.
- `OSPOUTNPRN` - The outbound routing number.
- `OSPOUTNPCIC` - The outbound carrier identification code.
- `OSPOUTNPDI` - The outbound number portability database dip indicator.
- `OSPOUTSPID` - The outbound service provider identity.
- `OSPOUTOCN` - The outbound operator company number.
- `OSPOUTSPN` - The outbound service provider name.
- `OSPOUTALTSPN` - The outbound alternate service provider name.
- `OSPOUTMCC` - The outbound mobile country code.
- `OSPOUTMNC` - The outbound mobile network code.
- `OSPOUTTOKEN` - The outbound OSP token.
- `OSPDESTREMAILS` - The number of remained destinations.
- `OSPOUTTIMELIMIT` - The outbound call duration limit in seconds.
- `OSPOUTCALLIDTYPES` - The outbound Call-ID types.
- `OSPOUTCALLID` - The outbound Call-ID. Only for H.323.
- `OSPDIALSTR` - The outbound Dial command string.

This application sets the following channel variable upon completion:

- `OSPLOOKUPSTATUS` - The status of OSPLookup attempt as a text string, one of
  - `SUCCESS`
  - `FAILED`
  - `ERROR`

Syntax

```
OSPLookup(exten[,provider[,options]])
```

Arguments

- `exten` - The exten of the call.
- `provider` - The name of the provider that is used to route the call.

- options
  - `h` - generate H323 call id for the outbound call
  - `s` - generate SIP call id for the outbound call. Have not been implemented
  - `i` - generate IAX call id for the outbound call. Have not been implemented

See Also

[Application_OSPAuth](#)
[Application_OSPNext](#)
[Application_OSPFinish](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_OSPNext

OSPNext()

Synopsis

Lookup next destination by OSP.

Description

Looks up the next destination via OSP.

Input variables:

- `OSPINHANDLE` - The inbound call OSP transaction handle.
- `OSPOUTHANDLE` - The outbound call OSP transaction handle.
- `OSPINTIMELIMIT` - The inbound call duration limit in seconds.
- `OSPOUTCALLIDTYPES` - The outbound Call-ID types.
- `OSPDESTREMAILS` - The number of remained destinations.

Output variables:

- `OSPOUTTECH` - The outbound channel technology.
- `OSPDESTINATION` - The destination IP address.
- `OSPOUTCALLING` - The outbound calling number.
- `OSPOUTCALLED` - The outbound called number.
- `OSPOUTNETWORKID` - The outbound destination network ID.
- `OSPOUTNPRN` - The outbound routing number.
- `OSPOUTNPCIC` - The outbound carrier identification code.
- `OSPOUTNPDI` - The outbound number portability database dip indicator.
- `OSPOUTSPID` - The outbound service provider identity.
- `OSPOUTOCN` - The outbound operator company number.
- `OSPOUTSPN` - The outbound service provider name.
- `OSPOUTALTSPN` - The outbound alternate service provider name.
- `OSPOUTMCC` - The outbound mobile country code.
- `OSPOUTMNC` - The outbound mobile network code.
- `OSPOUTTOKEN` - The outbound OSP token.
- `OSPDESTREMAILS` - The number of remained destinations.
- `OSPOUTTIMELIMIT` - The outbound call duration limit in seconds.
- `OSPOUTCALLID` - The outbound Call-ID. Only for H.323.
- `OSPDIALSTR` - The outbound Dial command string.

This application sets the following channel variable upon completion:

- `OSPNEXTSTATUS` - The status of the OSPNext attempt as a text string, one of

- SUCCESS
- FAILED
- ERROR

Syntax

```
OSPNext()
```

Arguments

See Also

Application_OSPAuth
Application_OSPLookup
Application_OSPFinish

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Page

Page()

Synopsis

Page series of phones

Description

Places outbound calls to the given *technology* / *resource* and dumps them into a conference bridge as muted participants. The original caller is dumped into the conference as a speaker and the room is destroyed when the original callers leaves.

Syntax

```
Page(Technology/Resource[&Technology2/Resource2[&...]][,options[,timeo
```

Arguments

- `Technology/Resource`
    - `Technology/Resource` - Specification of the device(s) to dial. These must be in the format of `Technology/Resource`, where *Technology* represents a particular channel driver, and *Resource* represents a resource available to that particular channel driver.
    - `Technology2/Resource2` - Optional extra devices to dial inparallel If you need more then one enter them as Technology2/Resource2& Technology3/Resourse3&.....
- `options`
    - `d` - Full duplex audio
    - `i` - Ignore attempts to forward the call
    - `q` - Quiet, do not play beep to caller
    - `r` - Record the page into a file (meetme option `r`)
    - `s` - Only dial a channel if its device state says that it is `NOT_INUSE`
    - `A` - Play an announcement simultaneously to all paged participants
        - `x` - The announcement to playback in all devices
    - `n` - Do not play simultaneous announcement to caller (implies A ❌)
- `timeout` - Specify the length of time that the system will attempt to connect a call. After this duration, any intercom calls that have not been answered will be hung up by the system.

Application_MeetMe

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Park

Park()

Synopsis

Park yourself.

Description

Used to park yourself (typically in combination with a supervised transfer to know the parking space). This application is always registered internally and does not need to be explicitly added into the dialplan, although you should include the `parkedcalls` context (or the context specified in `features.conf`).

If you set the If you set the `None` - `PARKINGLOT` variable, the call will be parked in the specifed parking context. Note setting this variable overrides the If you set the `None` - {{ PARKINGLOT}} set by the `CHANNEL` function.

If you set the If you set the `None` - `PARKINGEXTEN` variable to an extension in your parking context, Park() will park the call on that extension, unless it already exists. In that case, execution will continue at next priority.

If you set the If you set the `None` - `PARKINGLOT` variable, Park() will park the call in that parkinglot.

If you set the If you set the `None` - `PARKINGDYNAMIC` variable, this parkinglot from features.conf will be used as template for the newly created dynamic lot.

If you set the If you set the `None` - `PARKINGDYNCONTEXT` variable the newly created dynamic parking lot will use this context.

If you set the If you set the `None` - `PARKINGDYNPOS` variable the newly created dynamic parkinglot will use those parking postitions.

Syntax

```
Park([timeout[,return_context[,return_exten[,return_priority[,options]
```

Arguments

- `timeout` - A custom parking timeout for this parked call. Value in milliseconds.

- return_context - The context to return the call to after it times out.
- return_exten - The extension to return the call to after it times out.
- return_priority - The priority to return the call to after it times out.
- options - A list of options for this parked call.
  - r - Send ringing instead of MOH to the parked call.
  - R - Randomize the selection of a parking space.
  - s - Silence announcement of the parking space number.

See Also

[Application_ParkAndAnnounce](#)
[Application_ParkedCall](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ParkAndAnnounce

ParkAndAnnounce()

Synopsis

Park and Announce.

Description

Park a call into the parkinglot and announce the call to another channel.

The variable The variable `None` - `PARKEDAT` will contain the parking extension into which the call was placed. Use with the Local channel to allow the dialplan to make use of this information.

Syntax

```
ParkAndAnnounce(announce[:announce1[:...]],timeout,dial[,return_contex
```

Arguments

- announce_template
  - announce - Colon-separated list of files to announce. The word `PARKED` will be replaced by a say_digits of the extension in which the call is parked.
  - announce1
- timeout - Time in seconds before the call returns into the return context.
- dial - The app_dial style resource to call to make the announcement. Console/dsp calls the console.
- return_context - The goto-style label to jump the call back into after timeout. Default `priority+1`.

See Also

[Application_Park](#)
[Application_ParkedCall](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ParkedCall

ParkedCall()

Synopsis

Answer a parked call.

Description

Used to connect to a parked call. This application is always registered internally and does not need to be explicitly added into the dialplan, although you should include the `parkedcalls` context. If no extension is provided, then the first available parked call will be acquired.

Syntax

```
ParkedCall(exten)
```

Arguments

- `exten`

See Also

Application_Park
Application_ParkAndAnnounce

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_PauseMonitor

PauseMonitor()

Synopsis

Pause monitoring of a channel.

Description

Pauses monitoring of a channel until it is re-enabled by a call to UnpauseMonitor.

Syntax

```
PauseMonitor()
```

Arguments

See Also

Application_UnpauseMonitor

Application_PauseQueueMember

PauseQueueMember()

Synopsis

Pauses a queue member.

Description

Pauses (blocks calls for) a queue member. The given interface will be paused in the given queue. This prevents any calls from being sent from the queue to the interface until it is unpaused with UnpauseQueueMember or the manager interface. If no queuename is given, the interface is paused in every queue it is a member of. The application will fail if the interface is not found.

This application sets the following channel variable upon completion:

- PQMSTATUS - The status of the attempt to pause a queue member as a text string.
  - PAUSED
  - NOTFOUND

Example: PauseQueueMember(,SIP/3000)

Syntax

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

Arguments

- queuename
- interface
- options
- reason - Is used to add extra information to the appropriate queue_log entries and manager events.

See Also

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Pickup

Pickup()

Synopsis

Directed extension call pickup.

Description

This application can pickup any ringing channel that is calling the specified *extension*. If no *context* is specified, the current context will be used. If you use the special string `PICKUPMARK` for the context parameter, for example 10@PICKUPMARK, this application tries to find a channel which has defined a This application can pickup any ringing channel that is calling the specified `None` - `PICKUPMARK` channel variable with the same value as *extension* (in this example, `10`). When no parameter is specified, the application will pickup a channel matching the pickup group of the active channel.

Syntax

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

Arguments

- ext
  - extension
  - context
- ext2
  - extension2
  - context2

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_PickupChan

PickupChan()

Synopsis

Pickup a ringing channel.

Description

This will pickup a specified *channel* if ringing.

```
PickupChan(channel[,channel2[,...][,options]])
```

Arguments

- `channel`
- `channel2`
- `options`
    - `p` - Channel name specified partial name. Used when find channel by callid.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Playback

Playback()

Play a file.

Plays back given filenames (do not put extension of wav/alaw etc). The playback command answer the channel if no options are specified. If the file is non-existant it will fail

This application sets the following channel variable upon completion:

- `PLAYBACKSTATUS` - The status of the playback attempt as a text string.
    - `SUCCESS`
    - `FAILED`

See Also: Background (application) – for playing sound files that are interruptible

WaitExten (application) – wait for digits from caller, optionally play music on hold

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

Arguments

- `filenames`
    - `filename`
    - `filename2`
- `options` - Comma separated list of options
    - `skip` - Do not play if not answered
    - `noanswer` - Playback without answering, otherwise the channel will be answered before the sound is played. Not all channel types support playing messages while still on hook.Not all channel types support playing messages while still on hook.

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_PlayTones

PlayTones()

Synopsis

Play a tone list.

Description

Plays a tone list. Execution will continue with the next step in the dialplan immediately while the tones continue to play.

See the sample `indications.conf` for a description of the specification of a tonelist.

Syntax

```
PlayTones(arg)
```

Arguments

- `arg` - Arg is either the tone name defined in the `indications.conf` configuration file, or a directly specified list of frequencies and durations.

See Also

Application_StopPlayTones

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_PrivacyManager

PrivacyManager()

Synopsis

Require phone number to be entered, if no CallerID sent

Description

If no Caller*ID is sent, PrivacyManager answers the channel and asks the caller to enter their phone number. The caller is given *maxretries* attempts to do so. The application does **nothing** if Caller*ID was received on the channel.

The application sets the following channel variable upon completion:

- `PRIVACYMGRSTATUS` - The status of the privacy manager's attempt to collect a phone number from the user.
    - `SUCCESS`
    - `FAILED`

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

Arguments

- `maxretries` - Total tries caller is allowed to input a callerid. Defaults to `3`.
- `minlength` - Minimum allowable digits in the input callerid number. Defaults to `10`.
- `options` - Position reserved for options.
- `context` - Context to check the given callerid against patterns.

See Also

Application_Zapateller

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r321436.

Application_Proceeding

Proceeding()

Synopsis

Indicate proceeding.

Description

This application will request that a proceeding message be provided to the calling channel.

Syntax

```
Proceeding()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Progress

Progress()

Synopsis

Indicate progress.

Description

This application will request that in-band progress information be provided to the calling channel.

Syntax

```
Progress()
```

Arguments

See Also

[Application_Busy](#)
[Application_Congestion](#)
[Application_Ringing](#)
[Application_PlayTones](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Queue

Queue()

Synopsis

Queue a call for a call queue.

Description

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

This application will return to the dialplan if the queue does not exist, or any of the join options cause the caller to not enter the queue.

This application sets the following channel variable upon completion:

- QUEUESTATUS - The status of the call as a text string.
  - TIMEOUT
  - FULL
  - JOINEMPTY
  - LEAVEEMPTY
  - JOINUNAVAIL
  - LEAVEUNAVAIL
  - CONTINUE

Syntax

```
Queue(queuename[,options[,URL[,announceoverride[,timeout[,AGI[,macro[,
```

Arguments

- `queuename`
- `options`
  - `C` - Mark all calls as "answered elsewhere" when cancelled.
  - `c` - Continue in the dialplan if the callee hangs up.
  - `d` - data-quality (modem) call (minimum delay).
  - `h` - Allow **callee** to hang up by pressing *.
  - `H` - Allow **caller** to hang up by pressing *.
  - `n` - No retries on the timeout; will exit this application and go to the next step.
  - `i` - Ignore call forward requests from queue members and do nothing when they are requested.
  - `I` - Asterisk will ignore any connected line update requests or any redirecting party update requests it may receive on this dial attempt.
  - `r` - Ring instead of playing MOH. Periodic Announcements are still made, if applicable.
  - `R` - Ring instead of playing MOH when a member channel is actually ringing.
  - `t` - Allow the **called** user to transfer the calling user.
  - `T` - Allow the **calling** user to transfer the call.
  - `w` - Allow the **called** user to write the conversation to disk via Monitor.
  - `W` - Allow the **calling** user to write the conversation to disk via Monitor.
  - `k` - Allow the **called** party to enable parking of the call by sending the DTMF sequence defined for call parking in `features.conf`.
  - `K` - Allow the **calling** party to enable parking of the call by sending the DTMF sequence defined for call parking in `features.conf`.
  - `x` - Allow the **called** user to write the conversation to disk via MixMonitor.
  - `X` - Allow the **calling** user to write the conversation to disk via MixMonitor.
- `URL` - *URL* will be sent to the called party if the channel supports it.
- `announceoverride`
- `timeout` - Will cause the queue to fail out after a specified number of seconds, checked between each `queues.conf` *timeout* and *retry* cycle.
- `AGI` - Will setup an AGI script to be executed on the calling party's channel once they are connected to a queue member.
- `macro` - Will run a macro on the calling party's channel once they are connected to a queue member.
- `gosub` - Will run a gosub on the calling party's channel once they are connected to a queue member.
- `rule` - Will cause the queue's defaultrule to be overridden by the rule specified.
- `position` - Attempt to enter the caller into the queue at the numerical position specified. `1` would attempt to enter the caller at the head of the queue, and `3` would attempt to place the caller third in the queue.

See Also

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS
Function_QUEUE_WAITING_COUNT
Function_QUEUE_MEMBER_LIST
Function_QUEUE_MEMBER_PENALTY

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_QueueLog

QueueLog()

Synopsis

Writes to the queue_log file.

Allows you to write your own events into the queue log.

Example: QueueLog(101,${UNIQUEID},${AGENT},WENTONBREAK,600)

Syntax

```
QueueLog(queuename,uniqueid,agent,event[,additionalinfo])
```

Arguments

- queuename
- uniqueid
- agent
- event
- additionalinfo

See Also

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS
Function_QUEUE_WAITING_COUNT
Function_QUEUE_MEMBER_LIST
Function_QUEUE_MEMBER_PENALTY

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_RaiseException

RaiseException()

Synopsis

Handle an exceptional condition.

Description

This application will jump to the e extension in the current context, setting the dialplan function EXCEPTION(). If the e extension does not exist, the call will hangup.

```
RaiseException(reason)
```

Arguments

- `reason`

Function_EXCEPTION

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Read

Read()

Synopsis

Read a variable.

Description

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

This application sets the following channel variable upon completion:

- `READSTATUS` - This is the status of the read operation.
    - `OK`
    - `ERROR`
    - `HANGUP`
    - `INTERRUPTED`
    - `SKIPPED`
    - `TIMEOUT`

Syntax

```
Read(variable[,filename[&filename2[&...]]][,maxdigits[,options[,attempt
```

Arguments

- `variable` - The input digits will be stored in the given *variable* name.
- `filenames`
    - `filename` - file(s) to play before reading digits or tone with option i
    - `filename2`
- `maxdigits` - Maximum acceptable number of digits. Stops reading after *maxdigits* have been entered (without requiring the user to press the # key). Defaults to `0` - no limit - wait for the user press the # key. Any value below `0` means the same. Max accepted value is `255`.
- `options`
    - `s` - to return immediately if the line is not up.
    - `i` - to play filename as an indication tone from your `indications.conf`.
    - `n` - to read digits even if the line is not up.

- attempts - If greater than `1`, that many *attempts* will be made in the event no data is entered.
- timeout - The number of seconds to wait for a digit response. If greater than `0`, that value will override the default timeout. Can be floating point.

See Also

Application_SendDTMF

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ReadExten

ReadExten()

Synopsis

Read an extension into a variable.

Description

Reads a `#` terminated string of digits from the user into the given variable.

Will set READEXTENSTATUS on exit with one of the following statuses:

- READEXTENSTATUS -
    - OK - A valid extension exists in ${variable}.
    - TIMEOUT - No extension was entered in the specified time. Also sets ${variable} to "t".
    - INVALID - An invalid extension, ${INVALID_EXTEN}, was entered. Also sets ${variable} to "i".
    - SKIP - Line was not up and the option 's' was specified.
    - ERROR - Invalid arguments were passed.

Syntax

```
ReadExten(variable[,filename[,context[,option[,timeout]]]])
```

Arguments

- variable
- filename - File to play before reading digits or tone with option `i`
- context - Context in which to match extensions.
- option
    - s - Return immediately if the channel is not answered.
    - i - Play *filename* as an indication tone from your `indications.conf` or a directly specified list of frequencies and durations.
    - n - Read digits even if the channel is not answered.
- timeout - An integer number of seconds to wait for a digit response. If greater than `0`, that value will override the default timeout.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ReadFile

ReadFile()

Read the contents of a text file into a channel variable.

Description

Read the contents of a text file into channel variable *varname*

ReadFile has been deprecated in favor of Set(varname=${FILE(file,0,length)})

Syntax

```
ReadFile(varname=file[,length])
```

Arguments

- `varname` - Result stored here.
- `fileparams`
  - `file` - The name of the file to read.
  - `length` - Maximum number of characters to capture. If not specified defaults to max.

See Also

[Application_System](#)
[Application_Read](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ReceiveFax

ReceiveFax()

Synopsis

Receive a FAX and save as a TIFF/F file.

Description

This application is provided by res_fax, which is a FAX technology agnostic module that utilizes FAX technology resource modules to complete a FAX transmission.

Session arguments can be set by the FAXOPT function and to check results of the ReceiveFax() application.

Syntax

```
ReceiveFax(filename[,options])
```

Arguments

- `filename`
- `options`
    - `d` - Enable FAX debugging.
    - `f` - Allow audio fallback FAX transfer on T.38 capable channels.
    - `s` - Send progress Manager events (overrides statusevents setting in res_fax.conf).

See Also

Function_FAXOPT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r312509.

Application_ReceiveFAX (app_fax)

ReceiveFAX()

Synopsis

Receive a Fax

Description

Receives a FAX from the channel into the given filename overwriting the file if it already exists.

File created will be in TIFF format.

This application sets the following channel variables:

- `LOCALSTATIONID` - To identify itself to the remote end
- `LOCALHEADERINFO` - To generate a header line on each page
- `FAXSTATUS` -
    - `SUCCESS`
    - `FAILED`
- `FAXERROR` - Cause of failure
- `REMOTESTATIONID` - The CSID of the remote side
- `FAXPAGES` - Number of pages sent
- `FAXBITRATE` - Transmission rate
- `FAXRESOLUTION` - Resolution of sent fax

Syntax

```
ReceiveFAX(filename[,c])
```

Arguments

- `filename` - Filename of TIFF file save incoming fax
- `c` - Makes the application behave as the calling machine (Default behavior is as answering machine)

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r312509.

Application_ReceiveFax (res_fax)

ReceiveFax()

Receive a FAX and save as a TIFF/F file.

Description

This application is provided by res_fax, which is a FAX technology agnostic module that utilizes FAX technology resource modules to complete a FAX transmission.

Session arguments can be set by the FAXOPT function and to check results of the ReceiveFax() application.

Syntax

```
ReceiveFax(filename[,options])
```

Arguments

- `filename`
- `options`
  - `d` - Enable FAX debugging.
  - `f` - Allow audio fallback FAX transfer on T.38 capable channels.
  - `s` - Send progress Manager events (overrides statusevents setting in res_fax.conf).

See Also

Function_FAXOPT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r312509.

Application_Record

Record()

Synopsis

Record to a file.

Description

If filename contains `%d`, these characters will be replaced with a number incremented by one each time the file is recorded. Use `core show file formats` to see the available formats on your system User can press `#` to terminate the recording and continue to the next priority. If the user hangs up during a recording, all data will be lost and the application will terminate.

- `RECORDED_FILE` - Will be set to the final filename of the recording.
- `RECORD_STATUS` - This is the final status of the command
  - `DTMF` - A terminating DTMF was received ('#' or '*', depending upon option 't')
  - `SILENCE` - The maximum silence occurred in the recording.
  - `SKIP` - The line was not yet answered and the 's' option was specified.

- `TIMEOUT` - The maximum length was reached.
- `HANGUP` - The channel was hung up.
- `ERROR` - An unrecoverable error occurred, which resulted in a WARNING to the logs.

Syntax

```
Record(filenameformat[,silence[,maxduration[,options]]])
```

Arguments

- `filename`
    - `filename`
    - `format` - Is the format of the file type to be recorded (wav, gsm, etc).
- `silence` - Is the number of seconds of silence to allow before returning.
- `maxduration` - Is the maximum recording duration in seconds. If missing or 0 there is no maximum.
- `options`
    - `a` - Append to existing recording rather than replacing.
    - `n` - Do not answer, but record anyway if line not yet answered.
    - `q` - quiet (do not play a beep tone).
    - `s` - skip recording if the line is not yet answered.
    - `t` - use alternate '*' terminator key (DTMF) instead of default '#'
    - `x` - Ignore all terminator keys (DTMF) and keep recording until hangup.
    - `k` - Keep recorded file upon hangup.
    - `y` - Terminate recording if **any** DTMF digit is received.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_RemoveQueueMember

RemoveQueueMember()

Synopsis

Dynamically removes queue members.

Description

If the interface is **NOT** in the queue it will return an error.

This application sets the following channel variable upon completion:

- `RQMSTATUS` -
    - `REMOVED`
    - `NOTINQUEUE`
    - `NOSUCHQUEUE`

Example: RemoveQueueMember(techsupport,SIP/3000)

Syntax

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

Arguments

- queuename
- interface
- options

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_ResetCDR

ResetCDR()

Synopsis

Resets the Call Data Record.

Description

This application causes the Call Data Record to be reset.

Syntax

```
ResetCDR([options])
```

Arguments

- options
    - w - Store the current CDR record before resetting it.
    - a - Store any stacked records.
    - v - Save CDR variables.
    - e - Enable CDR only (negate effects of NoCDR).

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_RetryDial

RetryDial()

Synopsis

Place a call, retrying on failure allowing an optional exit extension.

Description

This application will attempt to place a call using the normal Dial application. If no channel can be reached, the *announce* file will be played. Then, it will wait *sleep* number of seconds before retrying the call. After *retries* number of attempts, the calling channel will continue at the next priority in the dialplan. If the *retries* setting is set to 0, this application will retry endlessly. While waiting to retry a call, a 1 digit extension may be dialed. If that extension exists in either the context defined in This application will attempt to place a call using the normal Dial application. If no channel can be reached, the `None - EXITCONTEXT` or the current one, The call will jump to that extension immediately. The *dialargs* are specified in the same format that arguments are provided to the Dial application.

Syntax

```
RetryDial(announce,sleep,retries,dialargs)
```

Arguments

- `announce` - Filename of sound that will be played when no channel can be reached
- `sleep` - Number of seconds to wait after a dial attempt failed before a new attempt is made
- `retries` - Number of retries When this is reached flow will continue at the next priority in the dialplan
- `dialargs` - Same format as arguments provided to the Dial application

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Return

Return()

Synopsis

Return from gosub routine.

Description

Jumps to the last label on the stack, removing it. The return *value*, if any, is saved in the channel variable Jumps to the last label on the stack, removing it. The return `None - GOSUB_RETVAL`.

Syntax

```
Return([value])
```

Arguments

- `value` - Return value.

See Also

[Application_Gosub](#)
[Application_StackPop](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Ringing

Ringing()

Synopsis

Indicate ringing tone.

Description

This application will request that the channel indicate a ringing tone to the user.

Syntax

```
Ringing()
```

Arguments

See Also

[Application_Busy](#)
[Application_Congestion](#)
[Application_Progress](#)
[Application_PlayTones](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SayAlpha

SayAlpha()

Synopsis

Say Alpha.

Description

This application will play the sounds that correspond to the letters of the given *string*.

Syntax

```
SayAlpha(string)
```

Arguments

- string

See Also

Application_SayDigits
Application_SayNumber
Application_SayPhonetic
Function_CHANNEL

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SayCountedAdj

SayCountedAdj()

Synopsis

Say a adjective in declined form in order to count things

Description

Selects and plays the proper form of an adjective according to the gender and of the noun which it modifies and the number of objects named by the noun-verb combination which have been counted. Used when saying things such as "5 new messages". The various singular and plural forms of the adjective are selected by adding suffixes to *filename*.

If the channel language is English, then no suffix will ever be added (since, in English, adjectives are not declined). If the channel language is Russian or some other slavic language, then the suffix will the specified *gender* for nominative, and "x" for genative plural. (The genative singular is not used when counting things.) For example, SayCountedAdj(1,new,f) will play sound file "newa" (containing the word "novaya"), but SayCountedAdj(5,new,f) will play sound file "newx" (containing the word "novikh").

Syntax

```
SayCountedAdj(number,filename[,gender])
```

Arguments

- `number` - The number of things
- `filename` - File name stem for the adjective
- `gender` - The gender of the noun modified, one of 'm', 'f', 'n', or 'c'

See Also

Application_SayCountedNoun
Application_SayNumber

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SayCountedNoun

SayCountedNoun()

Synopsis

Say a noun in declined form in order to count things

Description

Selects and plays the proper singular or plural form of a noun when saying things such as "five calls". English has simple rules for deciding when to say "call" and when to say "calls", but other languages have complicated rules which would be extremely difficult to implement in the Asterisk dialplan language.

The correct sound file is selected by examining the *number* and adding the appropriate suffix to *filename*. If the channel language is English, then the suffix will be either empty or "s". If the channel language is Russian or some other Slavic language, then the suffix will be empty for nominative, "x1" for genative singular, and "x2" for genative plural.

Note that combining *filename* with a suffix will not necessarily produce a correctly spelled plural form. For example, SayCountedNoun(2,man) will play the sound file "mans" rather than "men". This behavior is intentional. Since the file name is never seen by the end user, there is no need to implement complicated spelling rules. We simply record the word "men" in the sound file named "mans".

Syntax

```
SayCountedNoun(number,filename)
```

Arguments

- `number` - The number of things
- `filename` - File name stem for the noun that is the the name of the things

See Also

Application_SayCountedAdj

Application_SayNumber

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SayCountPL

SayCountPL()

Synopsis

Say Polish counting words.

Description

Polish grammar has some funny rules for counting words. for example 1 zloty, 2 zlote, 5 zlotych. This application will take the words for 1, 2-4 and 5 and decide based on grammar rules which one to use with the number you pass to it.

Example: SayCountPL(zloty,zlote,zlotych,122) will give: zlote

Syntax

```
SayCountPL(word1,word2,word5,number)
```

Arguments

- word1
- word2
- word5
- number

See Also

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SayDigits

SayDigits()

Synopsis

Say Digits.

Description

This application will play the sounds that correspond to the digits of the given number. This will use the language that is currently set for the channel.

Syntax

```
SayDigits(digits)
```

Arguments

- `digits`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SayNumber

SayNumber()

Synopsis

Say Number.

Description

This application will play the sounds that correspond to the given *digits*. Optionally, a *gender* may be specified. This will use the language that is currently set for the channel. See the LANGUAGE() function for more information on setting the language for the channel.

Syntax

```
SayNumber(digits[,gender])
```

Arguments

- `digits`
- `gender`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SayPhonetic

SayPhonetic()

Synopsis

Say Phonetic.

Description

This application will play the sounds from the phonetic alphabet that correspond to the letters in the given *string*.

Syntax

```
SayPhonetic(string)
```

Arguments

- `string`

See Also

[Application_SayAlpha](#)
[Application_SayDigits](#)
[Application_SayNumber](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SayUnixTime

SayUnixTime()

Synopsis

Says a specified time in a custom format.

Description

Uses some of the sound files stored in `/var/lib/asterisk/sounds` to construct a phrase saying the specified date and/or time in the specified format.

Syntax

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

Arguments

- `unixtime` - time, in seconds since Jan 1, 1970. May be negative. Defaults to now.
- `timezone` - timezone, see `/usr/share/zoneinfo` for a list. Defaults to machine default.
- `format` - a format the time is to be said in. See `voicemail.conf`. Defaults to `ABdY "digits/at" IMp`

Function_STRFTIME
Function_STRPTIME
Function_IFTIME

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SendDTMF

SendDTMF()

Synopsis

Sends arbitrary DTMF digits

Description

DTMF digits sent to a channel with half second pause

It will pass all digits or terminate if it encounters an error.

Syntax

```
SendDTMF(digits[,timeout_ms[,duration_ms[,channel]]])
```

Arguments

- `digits` - List of digits 0-9,*#,abcd
- `timeout_ms` - Amount of time to wait in ms between tones. (defaults to.25s)
- `duration_ms` - Duration of each digit
- `channel` - Channel where digits will be played

Application_Read

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SendFax

SendFax()

Synopsis

Sends a specified TIFF/F file as a FAX.

Description

This application is provided by res_fax, which is a FAX technology agnostic module that utilizes FAX technology resource modules to complete a FAX transmission.

Session arguments can be set by the FAXOPT function and to check results of the SendFax() application.

Syntax

```
SendFax([filename2[&...]][,options])
```

Arguments

- `filename`
  - `filename2` - TIFF file to send as a FAX.
- `options`
  - `d` - Enable FAX debugging.
  - `f` - Allow audio fallback FAX transfer on T.38 capable channels.
  - `s` - Send progress Manager events (overrides statusevents setting in res_fax.conf).
  - `z` - Initiate a T.38 reinvite on the channel if the remote end does not.

See Also

Function_FAXOPT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r312509.

Application_SendFAX (app_fax)

SendFAX()

Synopsis

Send a Fax

Description

Send a given TIFF file to the channel as a FAX.

This application sets the following channel variables:

- `LOCALSTATIONID` - To identify itself to the remote end
- `LOCALHEADERINFO` - To generate a header line on each page
- `FAXSTATUS` -
  - `SUCCESS`
  - `FAILED`
- `FAXERROR` - Cause of failure
- `REMOTESTATIONID` - The CSID of the remote side
- `FAXPAGES` - Number of pages sent
- `FAXBITRATE` - Transmission rate
- `FAXRESOLUTION` - Resolution of sent fax

Syntax

```
SendFAX(filename[,a])
```

Arguments

- `filename` - Filename of TIFF file to fax
- `a` - Makes the application behave as the answering machine (Default behavior is as calling machine)

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r312509.

Application_SendFax (res_fax)

SendFax()

Synopsis

Sends a specified TIFF/F file as a FAX.

Description

This application is provided by res_fax, which is a FAX technology agnostic module that utilizes FAX technology resource modules to complete a FAX transmission.

Session arguments can be set by the FAXOPT function and to check results of the SendFax() application.

Syntax

```
SendFax([filename2[&...]][,options])
```

Arguments

- `filename`
    - `filename2` - TIFF file to send as a FAX.
- `options`
    - `d` - Enable FAX debugging.
    - `f` - Allow audio fallback FAX transfer on T.38 capable channels.
    - `s` - Send progress Manager events (overrides statusevents setting in res_fax.conf).
    - `z` - Initiate a T.38 reinvite on the channel if the remote end does not.

See Also

Function_FAXOPT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r312509.

Application_SendImage

SendImage()

Sends an image file.

Description

Send an image file on a channel supporting it.

Result of transmission will be stored in Result of transmission will be stored in `None` - `SENDIMAGESTATUS`

- `SENDIMAGESTATUS` -
    - `SUCCESS` - Transmission succeeded.
    - `FAILURE` - Transmission failed.
    - `UNSUPPORTED` - Image transmission not supported by channel.

Syntax

```
SendImage(filename)
```

Arguments

- `filename` - Path of the filename (image) to send.

See Also

[Application_SendText](#)
[Application_SendURL](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SendText

SendText()

Synopsis

Send a Text Message.

Description

Sends *text* to current channel (callee).

Result of transmission will be stored in the Result of transmission will be stored in the `None` - `SENDTEXTSTATUS`

- `SENDTEXTSTATUS` -
    - `SUCCESS` - Transmission succeeded.
    - `FAILURE` - Transmission failed.
    - `UNSUPPORTED` - Text transmission not supported by channel.

At this moment, text is supposed to be 7 bit ASCII in most channels.At this moment, text is supposed to be 7 bit ASCII in most channels.

Syntax

```
SendText(text)
```

Arguments

- `text`

See Also

[Application_SendImage](#)
[Application_SendURL](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SendURL

SendURL()

Synopsis

Send a URL.

Description

Requests client go to *URL* (IAX2) or sends the URL to the client (other channels).

Result is returned in the Result is returned in the `None` - `SENDURLSTATUS` channel variable:

- `SENDURLSTATUS` -
    - `SUCCESS` - URL successfully sent to client.
    - `FAILURE` - Failed to send URL.
    - `NOLOAD` - Client failed to load URL (wait enabled).
    - `UNSUPPORTED` - Channel does not support URL transport.

SendURL continues normally if the URL was sent correctly or if the channel does not support HTML transport. Otherwise, the channel is hung up.

Syntax

```
SendURL(URL[,option])
```

Arguments

- `URL`
- `option`
    - `w` - Execution will wait for an acknowledgement that the URL has been loaded before continuing.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Set

Set()

Synopsis

Set channel variable or function value.

Description

This function can be used to set the value of channel variables or dialplan functions. When setting variables, if the variable name is prefixed with {}, *the variable will be inherited into channels created from the current channel. If the variable name is prefixed with _,* the variable will be inherited into channels created from the current channel and all children channels.

If (and only if), in If (and only if), in `/etc/asterisk/asterisk.conf`, you have a `[compat]` category, and you have `app_set = 1.6` under that,then the behavior of this app changes, and does not strip surrounding quotes from the right hand side as it did previously in 1.4. The `app_set = 1.6` is only inserted if `make samples` is executed, or if users insert this by hand into the `asterisk.conf` file. The advantages of not stripping out quoting, and not caring about the separator characters (comma and vertical bar) were sufficient to make these changes in 1.6. Confusion about how many backslashes would be needed to properly protect separators and quotes in various database access strings has been greatly reduced by these changes.

Syntax

```
Set(name,value)
```

Arguments

- `name`
- `value`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SetAMAFlags

SetAMAFlags()

Synopsis

Set the AMA Flags.

Description

This application will set the channel's AMA Flags for billing purposes.

Syntax

```
SetAMAFlags([flag])
```

Arguments

- `flag`

See Also

Function_CDR

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SetCallerPres

SetCallerPres()

Synopsis

Set CallerID Presentation.

Description

Set Caller*ID presentation on a call.

Syntax

```
SetCallerPres(presentation)
```

Arguments

- `presentation`
  - `allowed_not_screened` - Presentation Allowed, Not Screened.
  - `allowed_passed_screen` - Presentation Allowed, Passed Screen.
  - `allowed_failed_screen` - Presentation Allowed, Failed Screen.
  - `allowed` - Presentation Allowed, Network Number.
  - `prohib_not_screened` - Presentation Prohibited, Not Screened.
  - `prohib_passed_screen` - Presentation Prohibited, Passed Screen.
  - `prohib_failed_screen` - Presentation Prohibited, Failed Screen.

- `prohib` - Presentation Prohibited, Network Number.
- `unavailable` - Number Unavailable.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SetMusicOnHold

SetMusicOnHold()

Synopsis

Set default Music On Hold class.

Description

!!! DEPRECATED. USe Set(CHANNEL(musicclass)=...) instead !!!

Sets the default class for music on hold for a given channel. When music on hold is activated, this class will be used to select which music is played.

!!! DEPRECATED. USe Set(CHANNEL(musicclass)=...) instead !!!

Syntax

```
SetMusicOnHold(class)
```

Arguments

- `class`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SIPAddHeader

SIPAddHeader()

Synopsis

Add a SIP header to the outbound call.

Description

Adds a header to a SIP call placed with DIAL.

Remember to use the X-header if you are adding non-standard SIP headers, like

`X-Asterisk-Accountcode:`. Use this with care. Adding the wrong headers may jeopardize the SIP dialog.

Always returns `0`.

Syntax

```
SIPAddHeader(Header,Content)
```

Arguments

- `Header`
- `Content`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SIPDtmfMode

SIPDtmfMode()

Synopsis

Change the dtmfmode for a SIP call.

Description

Changes the dtmfmode for a SIP call.

Syntax

```
SIPDtmfMode(mode)
```

Arguments

- `mode`
    - `inband`
    - `info`
    - `rfc2833`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SIPRemoveHeader

SIPRemoveHeader()

Synopsis

Remove SIP headers previously added with SIPAddHeader

SIPRemoveHeader() allows you to remove headers which were previously added with SIPAddHeader(). If no parameter is supplied, all previously added headers will be removed. If a parameter is supplied, only the matching headers will be removed.

For example you have added these 2 headers:

SIPAddHeader(P-Asserted-Identity: sip:foo@bar);

SIPAddHeader(P-Preferred-Identity: sip:bar@foo);

// remove all headers

SIPRemoveHeader();

// remove all P- headers

SIPRemoveHeader(P-);

// remove only the PAI header (note the : at the end)

SIPRemoveHeader(P-Asserted-Identity😊;

Always returns 0.

Syntax

```
SIPRemoveHeader([Header])
```

Arguments

- Header

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Skel

Skel()

Synopsis

Simple one line explaination.

Description

This application is a template to build other applications from. It shows you the basic structure to create your own Asterisk applications.

Syntax

```
Skel(dummy[,options])
```

Arguments

- dummy
- options
    - a - Option A.
    - b - Option B.
    - c - Option C.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SLAStation

SLAStation()

Synopsis

Shared Line Appearance Station.

Description

This application should be executed by an SLA station. The argument depends on how the call was initiated. If the phone was just taken off hook, then the argument *station* should be just the station name. If the call was initiated by pressing a line key, then the station name should be preceded by an underscore and the trunk name associated with that line button.

For example:
station1_line1

On exit, this application will set the variable On exit, this application will set the variable None - SLASTATION_STATUS to one of the following values:

- SLASTATION_STATUS -
    - FAILURE
    - CONGESTION
    - SUCCESS

Syntax

```
SLAStation(station)
```

Arguments

- `station` - Station name

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SLATrunk

SLATrunk()

Synopsis

Shared Line Appearance Trunk.

Description

This application should be executed by an SLA trunk on an inbound call. The channel calling this application should correspond to the SLA trunk with the name *trunk* that is being passed as an argument.

On exit, this application will set the variable On exit, this application will set the variable `None` - `SLATRUNK_STATUS` to one of the following values:

- `SLATRUNK_STATUS` -
    - `FAILURE`
    - `SUCCESS`
    - `UNANSWERED`
    - `RINGTIMEOUT`

Syntax

```
SLATrunk(trunk[,options])
```

Arguments

- `trunk` - Trunk name
- `options`
    - `M` - Play back the specified MOH *class* instead of ringing
        - `class`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SMS

SMS()

Synopsis

Communicates with SMS service centres and SMS capable analogue phones.

SMS handles exchange of SMS data with a call to/from SMS capable phone or SMS PSTN service center. Can send and/or receive SMS messages. Works to ETSI ES 201 912; compatible with BT SMS PSTN service in UK and Telecom Italia in Italy.

Typical usage is to use to handle calls from the SMS service centre CLI, or to set up a call using `outgoing` or manager interface to connect service centre to SMS().

"Messages are processed as per text file message queues. smsq (a separate software) is a command to generate message queues and send messages.

The protocol has tight delay bounds. Please use short frames and disable/keep short the jitter buffer on the ATA to make sure that respones (ACK etc.) are received in time.The protocol has tight delay bounds. Please use short frames and disable/keep short the jitter buffer on the ATA to make sure that respones (ACK etc.) are received in time.

Syntax

```
SMS(name[,options[,addr[,body]]])
```

Arguments

- `name` - The name of the queue used in `/var/spool/asterisk/sms`
- `options`
  - `a` - Answer, i.e. send initial FSK packet.
  - `s` - Act as service centre talking to a phone.
  - `t` - Use protocol 2 (default used is protocol 1).
  - `p` - Set the initial delay to N ms (default is `300`). addr and body are a deprecated format to send messages out.
  - `r` - Set the Status Report Request (SRR) bit.
  - `o` - The body should be coded as octets not 7-bit symbols.
- `addr`
- `body`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SoftHangup

SoftHangup()

Synopsis

Hangs up the requested channel.

Description

Hangs up the requested channel. If there are no channels to hangup, the application will report it.

Syntax

```
SoftHangup(Technology/Resource[,options])
```

Arguments

- `Technology/Resource`
- `options`
    - `a` - Hang up all channels on a specified device instead of a single resource

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SpeechActivateGrammar

SpeechActivateGrammar()

Synopsis

Activate a grammar.

Description

This activates the specified grammar to be recognized by the engine. A grammar tells the speech recognition engine what to recognize, and how to portray it back to you in the dialplan. The grammar name is the only argument to this application.

Syntax

```
SpeechActivateGrammar(grammar_name)
```

Arguments

- `grammar_name`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SpeechBackground

SpeechBackground()

Synopsis

Play a sound file and wait for speech to be recognized.

Description

This application plays a sound file and waits for the person to speak. Once they start speaking playback of the file stops, and silence is heard. Once they stop talking the processing sound is played to indicate the speech recognition engine is working. Once results are available the application returns and results (score and text) are available using dialplan functions.

The first text and score are ${SPEECH_TEXT(0)} AND ${SPEECH_SCORE(0)} while the second are ${SPEECH_TEXT(1)} and ${SPEECH_SCORE(1)}.

The first argument is the sound file and the second is the timeout integer in seconds.

Syntax

```
SpeechBackground(sound_file[,timeout[,options]])
```

Arguments

- `sound_file`
- `timeout` - Timeout integer in seconds. Note the timeout will only start once the sound file has stopped playing.
- `options`
  - `n` - Don't answer the channel if it has not already been answered.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SpeechCreate

SpeechCreate()

Synopsis

Create a Speech Structure.

Description

This application creates information to be used by all the other applications. It must be called before doing any speech recognition activities such as activating a grammar. It takes the engine name to use as the argument, if not specified the default engine will be used.

Syntax

```
SpeechCreate(engine_name)
```

Arguments

- `engine_name`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SpeechDeactivateGrammar

SpeechDeactivateGrammar()

Synopsis

Deactivate a grammar.

Description

This deactivates the specified grammar so that it is no longer recognized.

Syntax

```
SpeechDeactivateGrammar(grammar_name)
```

Arguments

- `grammar_name` - The grammar name to deactivate

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SpeechDestroy

SpeechDestroy()

Synopsis

End speech recognition.

Description

This destroys the information used by all the other speech recognition applications. If you call this application but end up wanting to recognize more speech, you must call SpeechCreate() again before calling any other application.

Syntax

```
SpeechDestroy()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SpeechLoadGrammar

SpeechLoadGrammar()

Synopsis

Load a grammar.

Description

Load a grammar only on the channel, not globally.

Syntax

```
SpeechLoadGrammar(grammar_name,path)
```

Arguments

- grammar_name
- path

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SpeechProcessingSound

SpeechProcessingSound()

Synopsis

Change background processing sound.

Description

This changes the processing sound that SpeechBackground plays back when the speech recognition engine is processing and working to get results.

Syntax

```
SpeechProcessingSound(sound_file)
```

Arguments

- sound_file

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SpeechStart

SpeechStart()

Synopsis

Start recognizing voice in the audio stream.

Description

Tell the speech recognition engine that it should start trying to get results from audio being fed to it.

Syntax

```
SpeechStart()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_SpeechUnloadGrammar

SpeechUnloadGrammar()

Synopsis

Unload a grammar.

Description

Unload a grammar.

Syntax

```
SpeechUnloadGrammar(grammar_name)
```

Arguments

- grammar_name

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_StackPop

StackPop()

Synopsis

Remove one address from gosub stack.

Description

Removes last label on the stack, discarding it.

Syntax

```
StackPop()
```

Arguments

See Also

Application_Return
Application_Gosub

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_StartMusicOnHold

StartMusicOnHold()

Synopsis

Play Music On Hold.

Description

Starts playing music on hold, uses default music class for channel. Starts playing music specified by class. If omitted, the default music source for the channel will be used. Always returns 0.

Syntax

```
StartMusicOnHold(class)
```

Arguments

- `class`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_StopMixMonitor

StopMixMonitor()

Synopsis

Stop recording a call through MixMonitor, and free the recording's file handle.

Description

Stops the audio recording that was started with a call to `MixMonitor()` on the current channel.

Syntax

```
StopMixMonitor()
```

Arguments

See Also

Application_MixMonitor

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_StopMonitor

StopMonitor()

Synopsis

Stop monitoring a channel.

Description

Stops monitoring a channel. Has no effect if the channel is not monitored.

Syntax

```
StopMonitor()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_StopMusicOnHold

StopMusicOnHold()

Synopsis

Stop playing Music On Hold.

Description

Stops playing music on hold.

Syntax

```
StopMusicOnHold()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_StopPlayTones

StopPlayTones()

Synopsis

Stop playing a tone list.

Description

Stop playing a tone list, initiated by PlayTones().

Syntax

```
StopPlayTones()
```

Arguments

See Also

Application_PlayTones

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_System

System()

Synopsis

Execute a system command.

Description

Executes a command by using system(). If the command fails, the console should report a fallthrough.

Result of execution is returned in the Result of execution is returned in the `None` - `SYSTEMSTATUS` channel variable:

- `SYSTEMSTATUS` -
    - `FAILURE` - Could not execute the specified command.
    - `SUCCESS` - Specified command successfully executed.

Syntax

```
System(command)
```

Arguments

- `command` - Command to execute

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_TestClient

TestClient()

Synopsis

Execute Interface Test Client.

Description

Executes test client with given *testid*. Results stored in `/var/log/asterisk/testreports/<testid>-client.txt`

Syntax

```
TestClient(testid)
```

Arguments

- `testid` - An ID to identify this test.

See Also

Application_TestServer

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_TestServer

TestServer()

Synopsis

Execute Interface Test Server.

Description

Perform test server function and write call report. Results stored in
`/var/log/asterisk/testreports/<testid>-server.txt`

Syntax

```
TestServer()
```

Arguments

See Also

[Application_TestClient](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Transfer

Transfer()

Synopsis

Transfer caller to remote extension.

Description

Requests the remote caller be transferred to a given destination. If TECH (SIP, IAX2, LOCAL etc) is used, only an incoming call with the same channel technology will be transfered. Note that for SIP, if you transfer before call is setup, a 302 redirect SIP message will be returned to the caller.

The result of the application will be reported in the The result of the application will be reported in the `None` - `TRANSFERSTATUS` channel variable:

- `TRANSFERSTATUS` -
    - `SUCCESS` - Transfer succeeded.
    - `FAILURE` - Transfer failed.
    - `UNSUPPORTED` - Transfer unsupported by channel driver.

```
Transfer([Tech]destination)
```

Arguments

- `dest`
    - `Tech`
    - `destination`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_TryExec

TryExec()

Synopsis

Executes dialplan application, always returning.

Description

Allows an arbitrary application to be invoked even when not hard coded into the dialplan. To invoke external applications see the application System. Always returns to the dialplan. The channel variable TRYSTATUS will be set to one of:

- `TRYSTATUS` -
    - `SUCCESS` - If the application returned zero.
    - `FAILED` - If the application returned non-zero.
    - `NOAPP` - If the application was not found or was not specified.

Syntax

```
TryExec(arguments)
```

Arguments

- `appname`
    - `arguments`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_TrySystem

TrySystem()

Try executing a system command.

Description

Executes a command by using system().

Result of execution is returned in the Result of execution is returned in the `None` - `SYSTEMSTATUS` channel variable:

- `SYSTEMSTATUS` -
    - `FAILURE` - Could not execute the specified command.
    - `SUCCESS` - Specified command successfully executed.
    - `APPERROR` - Specified command successfully executed, but returned error code.

Syntax

```
TrySystem(command)
```

Arguments

- `command` - Command to execute

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_UnpauseMonitor

UnpauseMonitor()

Synopsis

Unpause monitoring of a channel.

Description

Unpauses monitoring of a channel on which monitoring had previously been paused with PauseMonitor.

Syntax

```
UnpauseMonitor()
```

Arguments

See Also

Application_PauseMonitor

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_UnpauseQueueMember

UnpauseQueueMember()

Synopsis

Unpauses a queue member.

Description

Unpauses (resumes calls to) a queue member. This is the counterpart to `PauseQueueMember()` and operates exactly the same way, except it unpauses instead of pausing the given interface.

This application sets the following channel variable upon completion:

- `UPQMSTATUS` - The status of the attempt to unpause a queue member as a text string.
    - `UNPAUSED`
    - `NOTFOUND`

Example: UnpauseQueueMember(,SIP/3000)

Syntax

```
UnpauseQueueMember([queuename,interface[,options[,reason]]])
```

Arguments

- `queuename`
- `interface`
- `options`
- `reason` - Is used to add extra information to the appropriate queue_log entries and manager events.

See Also

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS
Function_QUEUE_WAITING_COUNT
Function_QUEUE_MEMBER_LIST
Function_QUEUE_MEMBER_PENALTY

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_UserEvent

UserEvent()

Synopsis

Send an arbitrary event to the manager interface.

Description

Sends an arbitrary event to the manager interface, with an optional *body* representing additional arguments. The *body* may be specified as a , delimited list of headers. Each additional argument will be placed on a new line in the event. The format of the event will be:

Event: UserEvent

UserEvent: <specified event name>

[body]

If no *body* is specified, only Event and UserEvent headers will be present.

Syntax

```
UserEvent(eventname[,body])
```

Arguments

- eventname
- body

See Also

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Verbose

Verbose()

Synopsis

Send arbitrary text to verbose output.

Description

Sends an arbitrary text message to verbose output.

```
Verbose([level,message])
```

Arguments

- `level` - Must be an integer value. If not specified, defaults to 0.
- `message` - Output text message.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_VMAuthenticate

VMAuthenticate()

Synopsis

Authenticate with Voicemail passwords.

Description

This application behaves the same way as the Authenticate application, but the passwords are taken from `voicemail.conf`. If the *mailbox* is specified, only that mailbox's password will be considered valid. If the *mailbox* is not specified, the channel variable This application behaves the same way as the Authenticate application, but the passwords are taken from `None` - `AUTH_MAILBOX` will be set with the authenticated mailbox.

The VMAuthenticate application will exit if the following DTMF digit is entered as Mailbox or Password, and the extension exists:

Jump to the `a` extension in the current dialplan context.

Syntax

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

Arguments

- `mailbox`
    - `mailbox`
    - `context`
- `options`
    - `s` - Skip playing the initial prompts.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_VMSayName

VMSayName()

Synopsis

Play the name of a voicemail user

Description

This application will say the recorded name of the voicemail user specified as the argument to this application. If no context is provided, `default` is assumed.

Syntax

```
VMSayName([mailbox][@context])
```

Arguments

- mailbox
    - mailbox
    - context

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_VoiceMail

VoiceMail()

Synopsis

Leave a Voicemail message.

Description

This application allows the calling party to leave a message for the specified list of mailboxes. When multiple mailboxes are specified, the greeting will be taken from the first mailbox specified. Dialplan execution will stop if the specified mailbox does not exist.

The Voicemail application will exit if any of the following DTMF digits are received:

Jump to the `o` extension in the current dialplan context.

Jump to the `a` extension in the current dialplan context.

This application will set the following channel variable upon completion:

- VMSTATUS - This indicates the status of the execution of the VoiceMail application.
    - SUCCESS
    - USEREXIT

- FAILED

```
VoiceMail(mailbox1[&mailbox2[&...]][,options])
```

Arguments

- `mailboxs`
  - `mailbox1`
  - `mailbox2`
- `options`
  - `b` - Play the `busy` greeting to the calling party.
  - `d` - Accept digits for a new extension in context *c*, if played during the greeting. Context defaults to the current context.
    - `c`
  - `g` - Use the specified amount of gain when recording the voicemail message. The units are whole-number decibels (dB). Only works on supported technologies, which is DAHDI only.
    - `#`
  - `s` - Skip the playback of instructions for leaving a message to the calling party.
  - `u` - Play the `unavailable` greeting.
  - `U` - Mark message as `URGENT`.
  - `P` - Mark message as `PRIORITY`.

See Also

Application_VoiceMailMain

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_VoiceMailMain

VoiceMailMain()

Synopsis

Check Voicemail messages.

Description

This application allows the calling party to check voicemail messages. A specific *mailbox*, and optional corresponding *context*, may be specified. If a *mailbox* is not provided, the calling party will be prompted to enter one. If a *context* is not specified, the `default` context will be used.

The VoiceMailMain application will exit if the following DTMF digit is entered as Mailbox or Password, and the extension exists:

Jump to the `a` extension in the current dialplan context.

Syntax

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

Arguments

- mailbox
  - mailbox
  - context
- options
  - `p` - Consider the *mailbox* parameter as a prefix to the mailbox that is entered by the caller.
  - `g` - Use the specified amount of gain when recording a voicemail message. The units are whole-number decibels (dB).
    - #
  - `s` - Skip checking the passcode for the mailbox.
  - `a` - Skip folder prompt and go directly to *folder* specified. Defaults to `INBOX` (or `0`).
    - `folder`
    - `0` - INBOX
    - `1` - Old
    - `2` - Work
    - `3` - Family
    - `4` - Friends
    - `5` - Cust1
    - `6` - Cust2
    - `7` - Cust3
    - `8` - Cust4
    - `9` - Cust5
  - `0` - INBOX
  - `1` - Old
  - `2` - Work
  - `3` - Family
  - `4` - Friends
  - `5` - Cust1
  - `6` - Cust2
  - `7` - Cust3
  - `8` - Cust4
  - `9` - Cust5

See Also

Application_VoiceMail

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Wait

Wait()

Synopsis

Waits for some time.

Description

This application waits for a specified number of *seconds*.

Syntax

```
Wait(seconds)
```

Arguments

- `seconds` - Can be passed with fractions of a second. For example, `1.5` will ask the application to wait for 1.5 seconds.

See Also

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_WaitExten

WaitExten()

Synopsis

Waits for an extension to be entered.

Description

This application waits for the user to enter a new extension for a specified number of *seconds*.

Use of the application `WaitExten` within a macro will not function as expected. Please use the `Read` application in order to read DTMF from a channel currently executing a macro.

Syntax

```
WaitExten([seconds[,options]])
```

Arguments

- `seconds` - Can be passed with fractions of a second. For example, `1.5` will ask the application to wait for 1.5 seconds.
- `options`
    - `m` - Provide music on hold to the caller while waiting for an extension.
        - `x` - Specify the class for music on hold.

See Also

[Application_BackGround](#)
[Function_TIMEOUT](#)

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_WaitForNoise

WaitForNoise()

Synopsis

Waits for a specified amount of noise.

Description

Waits for up to *noiserequired* milliseconds of noise, *iterations* times. An optional *timeout* specified the number of seconds to return after, even if we do not receive the specified amount of noise. Use *timeout* with caution, as it may defeat the purpose of this application, which is to wait indefinitely until noise is detected on the line.

Syntax

```
WaitForNoise(noiserequired[,iterations[,timeout]])
```

Arguments

- `noiserequired`
- `iterations` - If not specified, defaults to `1`.
- `timeout` - Is specified only to avoid an infinite loop in cases where silence is never achieved.

See Also

Application_WaitForSilence

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_WaitForRing

WaitForRing()

Synopsis

Wait for Ring Application.

Description

Returns `0` after waiting at least *timeout* seconds, and only after the next ring has completed. Returns `0` on success or `-1` on hangup.

Syntax

```
WaitForRing(timeout)
```

Arguments

- `timeout`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_WaitForSilence

WaitForSilence()

Synopsis

Waits for a specified amount of silence.

Description

Waits for up to *silencerequired* milliseconds of silence, *iterations* times. An optional *timeout* specified the number of seconds to return after, even if we do not receive the specified amount of silence. Use *timeout* with caution, as it may defeat the purpose of this application, which is to wait indefinitely until silence is detected on the line. This is particularly useful for reverse-911-type call broadcast applications where you need to wait for an answering machine to complete its spiel before playing a message.

Typically you will want to include two or more calls to WaitForSilence when dealing with an answering machine; first waiting for the spiel to finish, then waiting for the beep, etc.

Examples:

WaitForSilence(500,2) will wait for 1/2 second of silence, twice

WaitForSilence(1000) will wait for 1 second of silence, once

WaitForSilence(300,3,10) will wait for 300ms silence, 3 times, and returns after 10 sec, even if silence is not detected

Sets the channel variable Sets the channel variable `None` - `WAITSTATUS` to one of these values:

- `WAITSTATUS` -
  - `SILENCE` - if exited with silence detected.
  - `TIMEOUT` - if exited without silence detected after timeout.

Syntax

```
WaitForSilence(silencerequired[,iterations[,timeout]])
```

Arguments

- `silencerequired`
- `iterations` - If not specified, defaults to `1`.
- `timeout` - Is specified only to avoid an infinite loop in cases where silence is never achieved.

See Also

Application_WaitForNoise

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_WaitMusicOnHold

WaitMusicOnHold()

Synopsis

Wait, playing Music On Hold.

Description

!!! DEPRECATED. Use MusicOnHold instead !!!

Plays hold music specified number of seconds. Returns `0` when done, or `-1` on hangup. If no hold music is available, the delay will still occur with no sound.

!!! DEPRECATED. Use MusicOnHold instead !!!

Syntax

```
WaitMusicOnHold(delay)
```

Arguments

- `delay`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_WaitUntil

WaitUntil()

Synopsis

Wait (sleep) until the current time is the given epoch.

Description

Waits until the given *epoch*.

Sets Sets `None` - `WAITUNTILSTATUS` to one of the following values:

- `WAITUNTILSTATUS` -
  - `OK` - Wait succeeded.
  - `FAILURE` - Invalid argument.
  - `HANGUP` - Channel hungup before time elapsed.
  - `PAST` - Time specified had already past.

Syntax

```
WaitUntil(epoch)
```

Arguments

- `epoch`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_While

While()

Synopsis

Start a while loop.

Description

Start a While Loop. Execution will return to this point when `EndWhile()` is called until expr is no longer true.

Syntax

```
While(expr)
```

Arguments

- `expr`

See Also

Application_EndWhile
Application_ExitWhile
Application_ContinueWhile

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Application_Zapateller

Zapateller()

Synopsis

Block telemarketers with SIT.

Description

Generates special information tone to block telemarketers from calling you.

This application will set the following channel variable upon completion:

- `ZAPATELLERSTATUS` - This will contain the last action accomplished by the Zapateller application. Possible values include:
  - `NOTHING`
  - `ANSWERED`
  - `ZAPPED`

Syntax

```
Zapateller(options)
```

Arguments

- `options` - Comma delimited list of options.
    - `answer` - Causes the line to be answered before playing the tone.
    - `nocallerid` - Causes Zapateller to only play the tone if there is no callerid information available.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Dialplan Application Template Page

MyApplication()

Synopsys

…..

Description

…

Syntax

- `MyApplication(arg[,something[,options]])`

Arguments

- arg
- something
- options
    - a
        - option 'a' is asdfadf
    - b
        - option 'b' is asdfasdfadf
    - c
        - option 'c' is for cookie

See Also

[Dialplan Function Template Page](#)
[AGI Command Template Page](#)
[AMI Action Template Page](#)

Import Version

This documentation was imported from Asterisk version `VERSION STRING HERE`.

Dialplan Functions

Dialplan Function Template Page

MY_FUNCTION()

Synopsys

…..

Description

…

Syntax

- `MY_FUNCTION(arg[,something[,options]])`

Arguments

- arg
- something
- options
  - a
    - option 'a' is asdfadf
  - b
    - option 'b' is asdfasdfadf
  - c
    - option 'c' is for cookie

See Also

[Dialplan Application Template Page](#)
[AGI Command Template Page](#)
[AMI Action Template Page](#)

Import Version

This documentation was imported from Asterisk version `VERSION STRING HERE`.

Function_AES_DECRYPT

AES_DECRYPT()

Synopsis

Decrypt a string encoded in base64 with AES given a 16 character key.

Description

Returns the plain text string.

Syntax

```
AES_DECRYPT(key,string)
```

Arguments

- `key` - AES Key

- `string` - Input string.

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_AES_ENCRYPT

AES_ENCRYPT()

Synopsis

Encrypt a string with AES given a 16 character key.

Description

Returns an AES encrypted string encoded in base64.

Syntax

```
AES_ENCRYPT(key,string)
```

Arguments

- `key` - AES Key
- `string` - Input string

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_AGC

AGC()

Synopsis

Apply automatic gain control to audio on a channel.

Description

The AGC function will apply automatic gain control to the audio on the channel that it is executed on. Using `rx` for audio received and `tx` for audio transmitted to the channel. When using this function you set a target audio level. It is primarily intended for use with analog lines, but could be useful for other channels as well. The target volume is set with a number between `1-32768`. The larger the number the louder (more gain) the channel will receive.

Examples:

exten => 1,1,Set(AGC(rx)=8000)

exten => 1,2,Set(AGC(tx)=off)

Syntax

```
AGC(channeldirection)
```

Arguments

- `channeldirection` - This can be either `rx` or `tx`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_AGENT

AGENT()

Synopsis

Gets information about an Agent

Description

Syntax

```
AGENT(agentid[,item])
```

Arguments

- `agentid`
- `item` - The valid items to retrieve are:
  - `status` - (default) The status of the agent (LOGGEDIN | LOGGEDOUT)
  - `password` - The password of the agent
  - `name` - The name of the agent
  - `mohclass` - MusicOnHold class
  - `channel` - The name of the active channel for the Agent (AgentLogin)
  - `fullchannel` - The untruncated name of the active channel for the Agent (AgentLogin)

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_ARRAY

ARRAY()

Synopsis

Allows setting multiple variables at once.

Description

The comma-delimited list passed as a value to which the function is set will be interpreted as a set of values to which the comma-delimited list of variable names in the argument should be set.

Example: Set(ARRAY(var1,var2)=1,2) will set var1 to 1 and var2 to 2

Syntax

```
ARRAY(var1[,var2[,...][,varN]])
```

Arguments

- var1
- var2
- varN

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_AST_CONFIG

AST_CONFIG()

Synopsis

Retrieve a variable from a configuration file.

Description

This function reads a variable from an Asterisk configuration file.

Syntax

```
AST_CONFIG(config_file,category,variable_name)
```

Arguments

- config_file
- category

- `variable_name`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_AUDIOHOOK_INHERIT

AUDIOHOOK_INHERIT()

Synopsis

Set whether an audiohook may be inherited to another channel

Description

By enabling audiohook inheritance on the channel, you are giving permission for an audiohook to be inherited by a descendent channel. Inheritance may be be disabled at any point as well.

Example scenario:

exten => 2000,1,MixMonitor(blah.wav)

exten => 2000,n,Set(AUDIOHOOK_INHERIT(MixMonitor)=yes)

exten => 2000,n,Dial(SIP/2000)

exten => 4000,1,Dial(SIP/4000)

exten => 5000,1,MixMonitor(blah2.wav)

exten => 5000,n,Dial(SIP/5000)

In this basic dialplan scenario, let's consider the following sample calls

Call 1: Caller dials 2000. The person who answers then executes an attended

transfer to 4000.

Result: Since extension 2000 set MixMonitor to be inheritable, after the

transfer to 4000 has completed, the call will continue to be recorded to blah.wav

Call 2: Caller dials 5000. The person who answers then executes an attended

transfer to 4000.

Result: Since extension 5000 did not set MixMonitor to be inheritable, the

recording will stop once the call has been transferred to 4000.

```
AUDIOHOOK_INHERIT(source)
```

Arguments

- `source` - The built-in sources in Asterisk are Note that the names are not case-sensitive
  - `MixMonitor`
  - `Chanspy`
  - `Volume`
  - `Speex`
  - `JACK_HOOK`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_BASE64_DECODE

BASE64_DECODE()

Synopsis

Decode a base64 string.

Description

Returns the plain text string.

Syntax

```
BASE64_DECODE(string)
```

Arguments

- `string` - Input string.

See Also

Function_BASE64_ENCODE
Function_AES_DECRYPT
Function_AES_ENCRYPT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_BASE64_ENCODE

BASE64_ENCODE()

Synopsis

Encode a string in base64.

Description

Returns the base64 string.

Syntax

```
BASE64_ENCODE(string)
```

Arguments

- `string` - Input string

See Also

Function_BASE64_DECODE
Function_AES_DECRYPT
Function_AES_ENCRYPT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_BLACKLIST

BLACKLIST()

Synopsis

Check if the callerid is on the blacklist.

Description

Uses astdb to check if the Caller*ID is in family `blacklist`. Returns `1` or `0`.

Syntax

```
BLACKLIST()
```

Arguments

See Also

Function_DB

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CALENDAR_BUSY

CALENDAR_BUSY()

Synopsis

Determine if the calendar is marked busy at this time.

Description

Check the specified calendar's current busy status.

Syntax

```
CALENDAR_BUSY(calendar)
```

Arguments

- `calendar`

See Also

[Function_CALENDAR_EVENT](#)
[Function_CALENDAR_QUERY](#)
[Function_CALENDAR_QUERY_RESULT](#)
[Function_CALENDAR_WRITE](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CALENDAR_EVENT

CALENDAR_EVENT()

Synopsis

Get calendar event notification data from a notification call.

Description

Whenever a calendar event notification call is made, the event data may be accessed with this function.

Syntax

```
CALENDAR_EVENT(field)
```

Arguments

- `field`
  - `summary` - The VEVENT SUMMARY property or Exchange event 'subject'
  - `description` - The text description of the event
  - `organizer` - The organizer of the event

- `location` - The location of the eventt
- `categories` - The categories of the event
- `priority` - The priority of the event
- `calendar` - The name of the calendar associated with the event
- `uid` - The unique identifier for this event
- `start` - The start time of the event
- `end` - The end time of the event
- `busystate` - The busy state of the event 0=FREE, 1=TENTATIVE, 2=BUSY

See Also

Function_CALENDAR_BUSY
Function_CALENDAR_QUERY
Function_CALENDAR_QUERY_RESULT
Function_CALENDAR_WRITE

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CALENDAR_QUERY

CALENDAR_QUERY()

Synopsis

Query a calendar server and store the data on a channel

Description

Get a list of events in the currently accessible timeframe of the *calendar* The function returns the id for accessing the result with CALENDAR_QUERY_RESULT()

Syntax

```
CALENDAR_QUERY(calendar[,start[,end]])
```

Arguments

- `calendar` - The calendar that should be queried
- `start` - The start time of the query (in seconds since epoch)
- `end` - The end time of the query (in seconds since epoch)

See Also

Function_CALENDAR_BUSY
Function_CALENDAR_EVENT
Function_CALENDAR_QUERY_RESULT
Function_CALENDAR_WRITE

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CALENDAR_QUERY_RESULT

CALENDAR_QUERY_RESULT()

Synopsis

Retrieve data from a previously run CALENDAR_QUERY() call

Description

After running CALENDAR_QUERY and getting a result *id*, calling `CALENDAR_QUERY` with that *id* and a *field* will return the data for that field. If multiple events matched the query, and *entry* is provided, information from that event will be returned.

Syntax

```
CALENDAR_QUERY_RESULT(id,field[,entry])
```

Arguments

- `id` - The query ID returned by `CALENDAR_QUERY`
- `field`
  - `getnum` - number of events occurring during time range
  - `summary` - A summary of the event
  - `description` - The full event description
  - `organizer` - The event organizer
  - `location` - The event location
  - `categories` - The categories of the event
  - `priority` - The priority of the event
  - `calendar` - The name of the calendar associed with the event
  - `uid` - The unique identifier for the event
  - `start` - The start time of the event (in seconds since epoch)
  - `end` - The end time of the event (in seconds since epoch)
  - `busystate` - The busy status of the event 0=FREE, 1=TENTATIVE, 2=BUSY
- `entry` - Return data from a specific event returned by the query

See Also

Function_CALENDAR_BUSY
Function_CALENDAR_EVENT
Function_CALENDAR_QUERY
Function_CALENDAR_WRITE

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CALENDAR_WRITE

CALENDAR_WRITE()

Synopsis

Write an event to a calendar

Description

Example: CALENDAR_WRITE(calendar,field1,field2,field3)=val1,val2,val3

The field and value arguments can easily be set/passed using the HASHKEYS() and HASH() functions

```
CALENDAR_WRITE(calendar,field[,...])
```

Arguments

- `calendar` - The calendar to write to
- `field`
    - `summary` - A summary of the event
    - `description` - The full event description
    - `organizer` - The event organizer
    - `location` - The event location
    - `categories` - The categories of the event
    - `priority` - The priority of the event
    - `uid` - The unique identifier for the event
    - `start` - The start time of the event (in seconds since epoch)
    - `end` - The end time of the event (in seconds since epoch)
    - `busystate` - The busy status of the event 0=FREE, 1=TENTATIVE, 2=BUSY

See Also

Function_CALENDAR_BUSY
Function_CALENDAR_EVENT
Function_CALENDAR_QUERY
Function_CALENDAR_QUERY_RESULT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CALLCOMPLETION

CALLCOMPLETION()

Synopsis

Get or set a call completion configuration parameter for a channel.

Description

The CALLCOMPLETION function can be used to get or set a call completion configuration parameter for a channel. Note that setting a configuration parameter will only change the parameter for the duration of the call. For more information see `doc/AST.pdf`. For more information on call completion parameters, see `configs/ccss.conf.sample`.

Syntax

```
CALLCOMPLETION(option)
```

Arguments

- `option` - The allowable options are:
    - `cc_agent_policy`
    - `cc_monitor_policy`
    - `cc_offer_timer`
    - `ccnr_available_timer`
    - `ccbs_available_timer`
    - `cc_recall_timer`
    - `cc_max_agents`
    - `cc_max_monitors`
    - `cc_callback_macro`
    - `cc_agent_dialstring`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CALLERID

CALLERID()

Synopsis

Gets or sets Caller*ID data on the channel.

Description

Gets or sets Caller*ID data on the channel. Uses channel callerid by default or optional callerid, if specified.

The allowable values for the *name-charset* field are the following:

Unknown

ISO8859-1

Withdrawn

ISO8859-2

ISO8859-3

ISO8859-4

ISO8859-5

ISO8859-7

ISO10646 Bmp String

ISO10646 UTF-8 String

```
CALLERID(datatype[,CID])
```

Arguments

- `datatype` - The allowable datatypes are:
    - `all`
    - `name`
    - `name-valid`
    - `name-charset`
    - `name-pres`
    - `num`
    - `num-valid`
    - `num-plan`
    - `num-pres`
    - `subaddr`
    - `subaddr-valid`
    - `subaddr-type`
    - `subaddr-odd`
    - `tag`
    - `ANI-all`
    - `ANI-name`
    - `ANI-name-valid`
    - `ANI-name-charset`
    - `ANI-name-pres`
    - `ANI-num`
    - `ANI-num-valid`
    - `ANI-num-plan`
    - `ANI-num-pres`
    - `ANI-tag`
    - `RDNIS`
    - `DNID`
    - `dnid-num-plan`
    - `dnid-subaddr`
    - `dnid-subaddr-valid`
    - `dnid-subaddr-type`
    - `dnid-subaddr-odd`
- `CID` - Optional Caller*ID

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CALLERPRES

CALLERPRES()

Synopsis

Gets or sets Caller*ID presentation on the channel.

Description

Gets or sets Caller*ID presentation on the channel. This function is deprecated in favor of CALLERID(num-pres) and CALLERID(name-pres). The following values are valid:

Presentation Allowed, Not Screened.

Presentation Allowed, Passed Screen.

Presentation Allowed, Failed Screen.

Presentation Allowed, Network Number.

Presentation Prohibited, Not Screened.

Presentation Prohibited, Passed Screen.

Presentation Prohibited, Failed Screen.

Presentation Prohibited, Network Number.

Number Unavailable.

Syntax

```
CALLERPRES()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CDR

CDR()

Synopsis

Gets or sets a CDR variable.

Description

All of the CDR field names are read-only, except for `accountcode`, `userfield`, and `amaflags`. You may, however, supply a name not on the above list, and create your own variable, whose value can be changed with this function, and this variable will be stored on the cdr.

For setting CDR values, the For setting CDR values, the `l` flag does not apply to setting the `accountcode`, `userfield`, or `amaflags`.

Raw values for `disposition`:

NO ANSWER

NO ANSWER (NULL record)

FAILED

BUSY

ANSWERED

Raw values for `amaflags`:

OMIT

BILLING

DOCUMENTATION

Example: exten => 1,1,Set(CDR(userfield)=test)

Syntax

```
CDR(name[,options])
```

Arguments

- `name` - CDR field name:
    - `clid` - Caller ID.
    - `lastdata` - Last application arguments.
    - `disposition` - ANSWERED, NO ANSWER, BUSY, FAILED.
    - `src` - Source.
    - `start` - Time the call started.
    - `amaflags` - DOCUMENTATION, BILL, IGNORE, etc.
    - `dst` - Destination.
    - `answer` - Time the call was answered.
    - `accountcode` - The channel's account code.
    - `dcontext` - Destination context.
    - `end` - Time the call ended.
    - `uniqueid` - The channel's unique id.
    - `dstchannel` - Destination channel.
    - `duration` - Duration of the call.
    - `userfield` - The channel's user specified field.
    - `lastapp` - Last application.
    - `billsec` - Duration of the call once it was answered.
    - `channel` - Channel name.
    - `sequence` - CDR sequence number.
- `options`
    - `f` - Returns billsec or duration fields as floating point values.
    - `l` - Uses the most recent CDR on a channel with multiple records
    - `r` - Searches the entire stack of CDRs on the channel.
    - `s` - Skips any CDR's that are marked 'LOCKED' due to forkCDR() calls. (on setting/writing CDR vars only)
    - `u` - Retrieves the raw, unprocessed value. For example, 'start', 'answer', and 'end' will be retrieved as epoch values, when the `u` option is passed, but formatted as YYYY-MM-DD HH:MM:SS otherwise. Similarly, disposition and amaflags will return their raw integral values.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CHANNEL

CHANNEL()

Synopsis

Gets/sets various pieces of information about the channel.

Description

Gets/sets various pieces of information about the channel, additional *item* may be available from the channel driver; see its documentation for details. Any *item* requested that is not available on the current channel will return an empty string.

Syntax

```
CHANNEL(item)
```

Arguments

- `item` - Standard items (provided by all channel technologies) are: **chan_sip** provides the following additional options: **chan_iax2** provides the following additional options: **chan_dahdi** provides the following additional options:
    - `audioreadformat` - R/O format currently being read.
    - `audionativeformat` - R/O format used natively for audio.
    - `audiowriteformat` - R/O format currently being written.
    - `callgroup` - R/W call groups for call pickup.
    - `channeltype` - R/O technology used for channel.
    - `checkhangup` - R/O Whether the channel is hanging up (1/0)
    - `language` - R/W language for sounds played.
    - `musicclass` - R/W class (from musiconhold.conf) for hold music.
    - `name` - The name of the channel
    - `parkinglot` - R/W parkinglot for parking.
    - `rxgain` - R/W set rxgain level on channel drivers that support it.
    - `secure_bridge_signaling` - Whether or not channels bridged to this channel require secure signaling
    - `secure_bridge_media` - Whether or not channels bridged to this channel require secure media
    - `state` - R/O state for channel
    - `tonezone` - R/W zone for indications played
    - `transfercapability` - R/W ISDN Transfer Capability, one of:
        - `SPEECH`
        - `DIGITAL`
        - `RESTRICTED_DIGITAL`
        - `3K1AUDIO`
        - `DIGITAL_W_TONES`
        - `VIDEO`
    - `txgain` - R/W set txgain level on channel drivers that support it.
    - `videonativeformat` - R/O format used natively for video
    - `trace` - R/W whether or not context tracing is enabled, only available **if CHANNEL_TRACE is defined**.
    - `peerip` - R/O Get the IP address of the peer.
    - `recvip` - R/O Get the source IP address of the peer.
    - `from` - R/O Get the URI from the From: header.
    - `uri` - R/O Get the URI from the Contact: header.
    - `useragent` - R/O Get the useragent.
    - `peername` - R/O Get the name of the peer.
    - `t38passthrough` - R/O 1 if T38 is offered or enabled in this channel, otherwise 0
    - `rtpqos` - R/O Get QOS information about the RTP stream This option takes two additional arguments: Argument 1: `audio` Get data about the audio stream `video` Get data about the video stream `text` Get data about the text stream Argument 2: `local_ssrc` Local SSRC (stream ID) `local_lostpackets` Local lost packets `local_jitter` Local calculated jitter `local_maxjitter` Local calculated jitter (maximum) `local_minjitter` Local calculated jitter (minimum) `local_normdevjitter` Local calculated jitter (normal deviation) `local_stdevjitter` Local calculated jitter (standard deviation) `local_count` Number of received packets `remote_ssrc` Remote SSRC (stream ID) `remote_lostpackets` Remote lost packets `remote_jitter` Remote reported jitter `remote_maxjitter` Remote calculated jitter (maximum) `remote_minjitter` Remote calculated jitter (minimum) `remote_normdevjitter` Remote calculated jitter (normal deviation) `remote_stdevjitter` Remote calculated jitter (standard deviation) `remote_count` Number of transmitted packets `rtt` Round trip time `maxrtt` Round trip time (maximum) `minrtt` Round trip time (minimum) `normdevrtt` Round trip time (normal deviation) `stdevrtt` Round trip time (standard deviation) `all` All statistics (in a form suited to logging, but not for parsing)
    - `rtpdest` - R/O Get remote RTP destination information. This option takes one additional argument: Argument 1: `audio` Get audio destination `video` Get video destination `text` Get text destination
    - `dahdi_channel` - R/O DAHDI channel related to this channel.
    - `dahdi_span` - R/O DAHDI span related to this channel.
    - `dahdi_type` - R/O DAHDI channel type, one of:

- analog
- mfc/r2
- pri
- pseudo
- ss7
- `keypad_digits` - R/O PRI Keypad digits that came in with the SETUP message.
- `reversecharge` - R/O PRI Reverse Charging Indication, one of:
  - `-1` - None
  - {{ 1}} - Reverse Charging Requested
- `no_media_path` - R/O PRI Nonzero if the channel has no B channel. The channel is either on hold or a call waiting call.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r314358.

Function_CHANNELS

CHANNELS()

Synopsis

Gets the list of channels, optionally filtering by a regular expression.

Description

Gets the list of channels, optionally filtering by a *regular_expression*. If no argument is provided, all known channels are returned. The *regular_expression* must correspond to the POSIX.2 specification, as shown in **regex(7)**. The list returned will be space-delimited.

Syntax

```
CHANNELS([regular_expression])
```

Arguments

- `regular_expression`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CHECKSIPDOMAIN

CHECKSIPDOMAIN()

Synopsis

Checks if domain is a local domain.

Description

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 in `sip.conf`.

Syntax

```
CHECKSIPDOMAIN(domain)
```

Arguments

- `domain`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CONNECTEDLINE

CONNECTEDLINE()

Synopsis

Gets or sets Connected Line data on the channel.

Description

Gets or sets Connected Line data on the channel.

The allowable values for the *name-charset* field are the following:

Unknown

ISO8859-1

Withdrawn

ISO8859-2

ISO8859-3

ISO8859-4

ISO8859-5

ISO8859-7

ISO10646 Bmp String

ISO10646 UTF-8 String

Syntax

```
CONNECTEDLINE(datatype[,i])
```

Arguments

- `datatype` - The allowable datatypes are:
    - `all`
    - `name`
    - `name-valid`
    - `name-charset`
    - `name-pres`
    - `num`
    - `num-valid`
    - `num-plan`
    - `num-pres`
    - `subaddr`
    - `subaddr-valid`
    - `subaddr-type`
    - `subaddr-odd`
    - `tag`
- `i` - If set, this will prevent the channel from sending out protocol messages because of the value being set

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CSV_QUOTE

CSV_QUOTE()

Synopsis

Quotes a given string for use in a CSV file, escaping embedded quotes as necessary

Description

Example: ${CSV_QUOTE("a,b" 123)} will return """a,b"" 123"

Syntax

```
CSV_QUOTE(string)
```

Arguments

- `string`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_CUT

CUT()

Slices and dices strings, based upon a named delimiter.

Cut out information from a string (*varname*), based upon a named delimiter.

```
CUT(varname,char-delim,range-spec)
```

Arguments

- `varname` - Variable you want cut
- `char-delim` - Delimiter, defaults to -
- `range-spec` - Number of the field you want (1-based offset), may also be specified as a range (with `-`) or group of ranges and fields (with `&`)

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Read from or write to the Asterisk database.

This function will read from or write a value to the Asterisk database. On a read, this function returns the corresponding value from the database, or blank if it does not exist. Reading a database value will also set the variable DB_RESULT. If you wish to find out if an entry exists, use the DB_EXISTS function.

```
DB(family,key)
```

Arguments

- `family`
- `key`

Application_DBdel
Function_DB_DELETE

Application_DBdeltree
Function_DB_EXISTS

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_DB_DELETE

DB_DELETE()

Synopsis

Return a value from the database and delete it.

Description

This function will retrieve a value from the Asterisk database and then remove that key from the database. This function will retrieve a value from the Asterisk database and then remove that key from the database. `None - DB_RESULT` will be set to the key's value if it exists.

Syntax

```
DB_DELETE(family,key)
```

Arguments

- `family`
- `key`

See Also

Application_DBdel
Function_DB
Application_DBdeltree

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_DB_EXISTS

DB_EXISTS()

Synopsis

Check to see if a key exists in the Asterisk database.

Description

This function will check to see if a key exists in the Asterisk database. If it exists, the function will return `1`. If not, it will return `0`. Checking for existence of a database key will also set the variable

DB_RESULT to the key's value if it exists.

Syntax

```
DB_EXISTS(family,key)
```

Arguments

- `family`
- `key`

See Also

Function_DB

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_DEC

DEC()

Synopsis

Decrements the value of a variable, while returning the updated value to the dialplan

Description

Decrements the value of a variable, while returning the updated value to the dialplan

Example: DEC(MyVAR) - Increments MyVar

Note: DEC(${MyVAR}) - Is wrong, as INC expects the variable name, not its value

Syntax

```
DEC(variable)
```

Arguments

- `variable` - The variable name to be manipulated, without the braces.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_DENOISE

DENOISE()

Synopsis

Apply noise reduction to audio on a channel.

Description

The DENOISE function will apply noise reduction to audio on the channel that it is executed on. It is very useful for noisy analog lines, especially when adjusting gains or using AGC. Use `rx` for audio received from the channel and `tx` to apply the filter to the audio being sent to the channel.

Examples:

exten => 1,1,Set(DENOISE(rx)=on)

exten => 1,2,Set(DENOISE(tx)=off)

Syntax

```
DENOISE(channeldirection)
```

Arguments

- `channeldirection` - This can be either `rx` or `tx` the values that can be set to this are either `on` and `off`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_DEVICE_STATE

DEVICE_STATE()

Synopsis

Get or Set a device state.

Description

The DEVICE_STATE function can be used to retrieve the device state from any device state provider. For example:

NoOp(SIP/mypeer has state ${DEVICE_STATE(SIP/mypeer)})

NoOp(Conference number 1234 has state ${DEVICE_STATE(MeetMe:1234)})

The DEVICE_STATE function can also be used to set custom device state from the dialplan. The `Custom:` prefix must be used. For example:

Set(DEVICE_STATE(Custom:lamp1)=BUSY)

Set(DEVICE_STATE(Custom:lamp2)=NOT_INUSE)

You can subscribe to the status of a custom device state using a hint in the dialplan:

exten => 1234,hint,Custom:lamp1

The possible values for both uses of this function are:

UNKNOWN | NOT_INUSE | INUSE | BUSY | INVALID | UNAVAILABLE | RINGING | RINGINUSE | ONHOLD

Syntax

```
DEVICE_STATE(device)
```

Arguments

- `device`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_DIALGROUP

DIALGROUP()

Synopsis

Manages a group of users for dialing.

Description

Presents an interface meant to be used in concert with the Dial application, by presenting a list of channels which should be dialled when referenced.

When DIALGROUP is read from, the argument is interpreted as the particular *group* for which a dial should be attempted. When DIALGROUP is written to with no arguments, the entire list is replaced with the argument specified.

Functionality is similar to a queue, except that when no interfaces are available, execution may continue in the dialplan. This is useful when you want certain people to be the first to answer any calls, with immediate fallback to a queue when the front line people are busy or unavailable, but you still want front line people to log in and out of that group, just like a queue.

Example:

exten => 1,1,Set(DIALGROUP(mygroup,add)=SIP/10)

exten => 1,n,Set(DIALGROUP(mygroup,add)=SIP/20)

exten => 1,n,Dial(${DIALGROUP(mygroup)})

```
DIALGROUP(group[,op])
```

Arguments

- group
- op - The operation name, possible values are:
  add - add a channel name or interface (write-only) del - remove a channel name or interface (write-only)

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_DIALPLAN_EXISTS

DIALPLAN_EXISTS()

Synopsis

Checks the existence of a dialplan target.

Description

This function returns 1 if the target exits. Otherwise, it returns 0.

Syntax

```
DIALPLAN_EXISTS(context[,extension[,priority]])
```

Arguments

- context
- extension
- priority

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_DUNDILOOKUP

DUNDILOOKUP()

Synopsis

Do a DUNDi lookup of a phone number.

Description

This will do a DUNDi lookup of the given phone number.

This function will return the Technology/Resource found in the first result in the DUNDi lookup. If no results were found, the result will be blank.

Syntax

```
DUNDILOOKUP(number[,context[,options]])
```

Arguments

- `number`
- `context` - If not specified the default will be `e164`.
- `options`
    - `b` - Bypass the internal DUNDi cache

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_DUNDIQUERY

DUNDIQUERY()

Synopsis

Initiate a DUNDi query.

Description

This will do a DUNDi lookup of the given phone number.

The result of this function will be a numeric ID that can be used to retrieve the results with the `DUNDIRESULT` function.

Syntax

```
DUNDIQUERY(number[,context[,options]])
```

Arguments

- `number`
- `context` - If not specified the default will be `e164`.
- `options`
    - `b` - Bypass the internal DUNDi cache

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_DUNDIRESULT

DUNDIRESULT()

Synopsis

Retrieve results from a DUNDIQUERY.

Description

This function will retrieve results from a previous use\n" of the `DUNDIQUERY` function.

Syntax

```
DUNDIRESULT(id[,resultnum])
```

Arguments

- `id` - The identifier returned by the `DUNDIQUERY` function.
- `resultnum`
    - `number` - The number of the result that you want to retrieve, this starts at `1`
    - `getnum` - The total number of results that are available.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_ENUMLOOKUP

ENUMLOOKUP()

Synopsis

General or specific querying of NAPTR records for ENUM or ENUM-like DNS pointers.

Description

For more information see `doc/AST.pdf`.

Syntax

```
ENUMLOOKUP(number[,method-type[,options[,record#[,zone-suffix]]]])
```

Arguments

- `number`
- `method-type` - If no *method-type* is given, the default will be `sip`.
- `options`
    - `c` - Returns an integer count of the number of NAPTRs of a certain RR type. Combination of `c` and Method-type of `ALL` will return a count of all NAPTRs for the record.
    - `u` - Returns the full URI and does not strip off the URI-scheme.

- s - Triggers ISN specific rewriting.
- i - Looks for branches into an Infrastructure ENUM tree.
- d - for a direct DNS lookup without any flipping of digits.
- record# - If no *record#* is given, defaults to `1`.
- zone-suffix - If no *zone-suffix* is given, the default will be `e164.arpa`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_ENUMQUERY

ENUMQUERY()

Synopsis

Initiate an ENUM query.

Description

This will do a ENUM lookup of the given phone number.

Syntax

```
ENUMQUERY(number[,method-type[,zone-suffix]])
```

Arguments

- number
- method-type - If no *method-type* is given, the default will be `sip`.
- zone-suffix - If no *zone-suffix* is given, the default will be `e164.arpa`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_ENUMRESULT

ENUMRESULT()

Synopsis

Retrieve results from a ENUMQUERY.

Description

This function will retrieve results from a previous use of the ENUMQUERY function.

Syntax

```
ENUMRESULT(id,resultnum)
```

Arguments

- `id` - The identifier returned by the ENUMQUERY function.
- `resultnum` - The number of the result that you want to retrieve. Results start at `1`. If this argument is specified as `getnum`, then it will return the total number of results that are available.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_ENV

ENV()

Synopsis

Gets or sets the environment variable specified.

Description

Variables starting with `AST_` are reserved to the system and may not be set.

Syntax

```
ENV(varname)
```

Arguments

- `varname` - Environment variable name

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_EVAL

EVAL()

Synopsis

Evaluate stored variables

Description

Using EVAL basically causes a string to be evaluated twice. When a variable or expression is in the dialplan, it will be evaluated at runtime. However, if the results of the evaluation is in fact

another variable or expression, using EVAL will have it evaluated a second time.

Example: If the Example: If the `None` - `MYVAR` contains Example: If the `None` - `OTHERVAR`, then the result of ${EVAL(Example: If the `None` - `MYVAR`)} in the dialplan will be the contents of Example: If the `None` - `OTHERVAR`. Normally just putting Example: If the `None` - `MYVAR` in the dialplan the result would be Example: If the `None` - `OTHERVAR`.

Syntax

```
EVAL(variable)
```

Arguments

- `variable`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_EXCEPTION

EXCEPTION()

Synopsis

Retrieve the details of the current dialplan exception.

Description

Retrieve the details (specified *field*) of the current dialplan exception.

Syntax

```
EXCEPTION(field)
```

Arguments

- `field` - The following fields are available for retrieval:
    - `reason` - INVALID, ERROR, RESPONSETIMEOUT, ABSOLUTETIMEOUT, or custom value set by the RaiseException() application
    - `context` - The context executing when the exception occurred.
    - `exten` - The extension executing when the exception occurred.
    - `priority` - The numeric priority executing when the exception occurred.

See Also

Application_RaiseException

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_EXISTS

EXISTS()

Synopsis

Test the existence of a value.

Description

Returns `1` if exists, `0` otherwise.

Syntax

```
EXISTS(data)
```

Arguments

- `data`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_EXTENSION_STATE

EXTENSION_STATE()

Synopsis

Get an extension's state.

Description

The EXTENSION_STATE function can be used to retrieve the state from any hinted extension. For example:

NoOp(1234@default has state ${EXTENSION_STATE(1234)})

NoOp(4567@home has state ${EXTENSION_STATE(4567@home)})

The possible values returned by this function are:

UNKNOWN | NOT_INUSE | INUSE | BUSY | INVALID | UNAVAILABLE | RINGING | RINGINUSE | HOLDINUSE | ONHOLD

Syntax

```
EXTENSION_STATE(extension[,context])
```

Arguments

- `extension`
- `context` - If it is not specified defaults to `default`.

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_FAXOPT

FAXOPT()

Synopsis

Gets/sets various pieces of information about a fax session.

Description

FAXOPT can be used to override the settings for a FAX session listed in `res_fax.conf`, it can also be used to retreive information about a FAX session that has finished eg. pages/status.

Syntax

```
FAXOPT(item)
```

Arguments

- `item`
    - `ecm` - R/W Error Correction Mode (ECM) enable with 'yes', disable with 'no'.
    - `error` - R/O FAX transmission error code upon failure.
    - `filename` - R/O Filename of the first file of the FAX transmission.
    - `filenames` - R/O Filenames of all of the files in the FAX transmission (comma separated).
    - `headerinfo` - R/W FAX header information.
    - `localstationid` - R/W Local Station Identification.
    - `minrate` - R/W Minimum transfer rate set before transmission.
    - `maxrate` - R/W Maximum transfer rate set before transmission.
    - `modem` - R/W Modem type (v17/v27/v29).
    - `pages` - R/O Number of pages transferred.
    - `rate` - R/O Negotiated transmission rate.
    - `remotestationid` - R/O Remote Station Identification after transmission.
    - `resolution` - R/O Negotiated image resolution after transmission.
    - `sessionid` - R/O Session ID of the FAX transmission.
    - `status` - R/O Result Status of the FAX transmission.
    - `statusstr` - R/O Verbose Result Status of the FAX transmission.

See Also

[Application_ReceiveFax](#)
[Application_SendFax](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_FIELDNUM

FIELDNUM()

Synopsis

Return the 1-based offset of a field in a list

Description

Search the variable named *varname* for the string *value* delimited by *delim* and return a 1-based offset as to its location. If not found or an error occured, return `0`.

The delimiter may be specified as a special or extended ASCII character, by encoding it. The characters `\n`, `\r`, and `\t` are all recognized as the newline, carriage return, and tab characters, respectively. Also, octal and hexadecimal specifications are recognized by the patterns `\0nnn` and `\xHH`, respectively. For example, if you wanted to encode a comma as the delimiter, you could use either `\054` or `\x2C`.

Example: If ${example} contains `ex-amp-le`, then ${FIELDNUM(example,-,amp)} returns 2.

Syntax

```
FIELDNUM(varname,delim,value)
```

Arguments

- `varname`
- `delim`
- `value`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_FIELDQTY

FIELDQTY()

Synopsis

Count the fields with an arbitrary delimiter

Description

The delimiter may be specified as a special or extended ASCII character, by encoding it. The characters `\n`, `\r`, and `\t` are all recognized as the newline, carriage return, and tab characters, respectively. Also, octal and hexadecimal specifications are recognized by the patterns `\0nnn` and `\xHH`, respectively. For example, if you wanted to encode a comma as the delimiter, you could use either `\054` or `\x2C`.

Example: If ${example} contains `ex-amp-le`, then ${FIELDQTY(example,-)} returns 3.

```
FIELDQTY(varname,delim)
```

Arguments

- `varname`
- `delim`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_FILE

FILE()

Read or write text file.

Read and write text file in character and line mode.

Examples:

Read mode (byte):

;reads the entire content of the file.

Set(foo=${FILE(/tmp/test.txt)})

;reads from the 11th byte to the end of the file (i.e. skips the first 10).

Set(foo=${FILE(/tmp/test.txt,10)})

;reads from the 11th to 20th byte in the file (i.e. skip the first 10, then read 10 bytes).

Set(foo=${FILE(/tmp/test.txt,10,10)})

Read mode (line):

; reads the 3rd line of the file.

Set(foo=${FILE(/tmp/test.txt,3,1,l)})

; reads the 3rd and 4th lines of the file.

Set(foo=${FILE(/tmp/test.txt,3,2,l)})

; reads from the third line to the end of the file.

Set(foo=${FILE(/tmp/test.txt,3,,l)})

; reads the last three lines of the file.

Set(foo=${FILE(/tmp/test.txt,-3,,l)})

; reads the 3rd line of a DOS-formatted file.

Set(foo=${FILE(/tmp/test.txt,3,1,l,d)})

Write mode (byte):

; truncate the file and write "bar"

Set(FILE(/tmp/test.txt)=bar)

; Append "bar"

Set(FILE(/tmp/test.txt,,,a)=bar)

; Replace the first byte with "bar" (replaces 1 character with 3)

Set(FILE(/tmp/test.txt,0,1)=bar)

; Replace 10 bytes beginning at the 21st byte of the file with "bar"

Set(FILE(/tmp/test.txt,20,10)=bar)

; Replace all bytes from the 21st with "bar"

Set(FILE(/tmp/test.txt,20)=bar)

; Insert "bar" after the 4th character

Set(FILE(/tmp/test.txt,4,0)=bar)

Write mode (line):

; Replace the first line of the file with "bar"

Set(FILE(/tmp/foo.txt,0,1,l)=bar)

; Replace the last line of the file with "bar"

Set(FILE(/tmp/foo.txt,-1,,l)=bar)

; Append "bar" to the file with a newline

Set(FILE(/tmp/foo.txt,,,al)=bar)

```
FILE(filename[,offset[,length[,options[,format]]]])
```

Arguments

- `filename`
- `offset` - Maybe specified as any number. If negative, *offset* specifies the number of bytes back from the end of the file.
- `length` - If specified, will limit the length of the data read to that size. If negative, trims *length* bytes from the end of the file.
- `options`
    - `l` - Line mode: offset and length are assumed to be measured in lines, instead of byte offsets.
    - `a` - In write mode only, the append option is used to append to the end of the file, instead of overwriting the existing file.
    - `d` - In write mode and line mode only, this option does not automatically append a newline string to the end of a value. This is useful for deleting lines, instead of setting them to blank.
- `format` - The *format* parameter may be used to delimit the type of line terminators in line mode.
    - `u` - Unix newline format.
    - `d` - DOS newline format.
    - `m` - Macintosh newline format.

See Also

[Function_FILE_COUNT_LINE](#)
[Function_FILE_FORMAT](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_FILE_COUNT_LINE

FILE_COUNT_LINE()

Synopsis

Obtains the number of lines of a text file.

Description

Returns the number of lines, or `-1` on error.

Syntax

```
FILE_COUNT_LINE(filename[,format])
```

Arguments

- `filename`
- `format` - Format may be one of the following: If not specified, an attempt will be made to determine the newline format type.If not specified, an attempt will be made to determine the newline format type.
    - `u` - Unix newline format.
    - `d` - DOS newline format.
    - `m` - Macintosh newline format.

Function_FILE
Function_FILE_FORMAT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_FILE_FORMAT

FILE_FORMAT()

Synopsis

Return the newline format of a text file.

Description

Return the line terminator type:

'u' - Unix "\n" format

'd' - DOS "\r\n" format

'm' - Macintosh "\r" format

'x' - Cannot be determined

Syntax

```
FILE_FORMAT(filename)
```

Arguments

- filename

See Also

Function_FILE
Function_FILE_COUNT_LINE

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_FILTER

FILTER()

Synopsis

Filter the string to include only the allowed characters

Permits all characters listed in *allowed-chars*, filtering all others outs. In addition to literally listing the characters, you may also use ranges of characters (delimited by a -

Hexadecimal characters started with a \x (i.e. \x20)

Octal characters started with a \0 (i.e. \040)

Also \t, \n and \r are recognized.

If you want the If you want the - character it needs to be prefixed with a {{}}

Syntax

```
FILTER(allowed-chars,string)
```

Arguments

- allowed-chars
- string

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_FRAME_TRACE

FRAME_TRACE()

Synopsis

View internal ast_frames as they are read and written on a channel.

Description

Examples:

exten => 1,1,Set(FRAME_TRACE(white)=DTMF_BEGIN,DTMF_END); view only DTMF frames.

exten => 1,1,Set(FRAME_TRACE()=DTMF_BEGIN,DTMF_END); view only DTMF frames.

exten => 1,1,Set(FRAME_TRACE(black)=DTMF_BEGIN,DTMF_END); view everything except DTMF frames.

Syntax

```
FRAME_TRACE(filter list type)
```

Arguments

- `filter list type` - A filter can be applied to the trace to limit what frames are viewed. This filter can either be a `white` or `black` list of frame types. When no filter type is present, `white` is used. If no arguments are provided at all, all frames will be output. Below are the different types of frames that can be filtered.
    - `DTMF_BEGIN`
    - `DTMF_END`
    - `VOICE`
    - `VIDEO`
    - `CONTROL`
    - `NULL`
    - `IAX`
    - `TEXT`
    - `IMAGE`
    - `HTML`
    - `CNG`
    - `MODEM`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_GLOBAL

GLOBAL()

Synopsis

Gets or sets the global variable specified.

Description

Set or get the value of a global variable specified in *varname*

Syntax

```
GLOBAL(varname)
```

Arguments

- `varname` - Global variable name

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_GROUP

GROUP()

Gets or sets the channel group.

category can be employed for more fine grained group management. Each channel can only be member of exactly one group per *category*.

```
GROUP([category])
```

Arguments

- category - Category name.

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_GROUP_COUNT

GROUP_COUNT()

Counts the number of channels in the specified group.

Calculates the group count for the specified group, or uses the channel's current group if not specifed (and non-empty).

```
GROUP_COUNT([groupname[,category]])
```

Arguments

- groupname - Group name.
- category - Category name

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_GROUP_LIST

GROUP_LIST()

Gets a list of the groups set on a channel.

Description

Gets a list of the groups set on a channel.

Syntax

```
GROUP_LIST()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_GROUP_MATCH_COUNT

GROUP_MATCH_COUNT()

Synopsis

Counts the number of channels in the groups matching the specified pattern.

Description

Calculates the group count for all groups that match the specified pattern. Note: category matching is applied after matching based on group. Uses standard regular expression matching on both (see regex(7)).

Syntax

```
GROUP_MATCH_COUNT(groupmatch[,category])
```

Arguments

- `groupmatch` - A standard regular expression used to match a group name.
- `category` - A standard regular expression used to match a category name.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_HASH

HASH()

Implementation of a dialplan associative array

Description

In two arguments mode, gets and sets values to corresponding keys within a named associative array. The single-argument mode will only work when assigned to from a function defined by func_odbc

Syntax

```
HASH(hashname[,hashkey])
```

Arguments

- `hashname`
- `hashkey`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_HASHKEYS

HASHKEYS()

Synopsis

Retrieve the keys of the HASH() function.

Description

Returns a comma-delimited list of the current keys of the associative array defined by the HASH() function. Note that if you iterate over the keys of the result, adding keys during iteration will cause the result of the HASHKEYS() function to change.

Syntax

```
HASHKEYS(hashname)
```

Arguments

- `hashname`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_HINT

HINT()

Synopsis

Get the devices set for a dialplan hint.

Description

The HINT function can be used to retrieve the list of devices that are mapped to a dialplan hint. For example:

NoOp(Hint for Extension 1234 is ${HINT(1234)})

Syntax

```
HINT(extension[@context][,options])
```

Arguments

- extension
    - extension
    - context
- options
    - n - Retrieve name on the hint instead of list of devices.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_IAXPEER

IAXPEER()

Synopsis

Gets IAX peer information.

Description

Syntax

```
IAXPEER(peername[,item])
```

Arguments

- peername
    - CURRENTCHANNEL - If *peername* is specified to this value, return the IP address of the endpoint of the current channel
- item - If *peername* is specified, valid items are:
    - ip - (default) The IP address.

- `status` - The peer's status (if `qualify=yes`)
- `mailbox` - The configured mailbox.
- `context` - The configured context.
- `expire` - The epoch time of the next expire.
- `dynamic` - Is it dynamic? (yes/no).
- `callerid_name` - The configured Caller ID name.
- `callerid_num` - The configured Caller ID number.
- `codecs` - The configured codecs.
- `codec[x]` - Preferred codec index number *x* (beginning with `0`)

See Also

Function_SIPPEER

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_IAXVAR

IAXVAR()

Synopsis

Sets or retrieves a remote variable.

Description

Syntax

```
IAXVAR(varname)
```

Arguments

- `varname`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_ICONV

ICONV()

Synopsis

Converts charsets of strings.

Description

Converts string from *in-charset* into *out-charset*. For available charsets, use `iconv -l` on your shell command line.

Due to limitations within the API, ICONV will not currently work with charsets with embedded NULLs. If found, the string will terminate.Due to limitations within the API, ICONV will not currently work with charsets with embedded NULLs. If found, the string will terminate.

Syntax

```
ICONV(in-charset,out-charset,string)
```

Arguments

- `in-charset` - Input charset
- `out-charset` - Output charset
- `string` - String to convert, from *in-charset* to *out-charset*

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_IF

IF()

Synopsis

Check for an expresion.

Description

Returns the data following ? if true, else the data following :

Syntax

```
IF(expresion?[true][:false])
```

Arguments

- `expresion`
- `retvalue`
  - `true`
  - `false`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_IFMODULE

IFMODULE()

Synopsis

Checks if an Asterisk module is loaded in memory.

Description

Checks if a module is loaded. Use the full module name as shown by the list in `module list`. Returns `1` if module exists in memory, otherwise `0`

Syntax

```
IFMODULE(modulename.so)
```

Arguments

- `modulename.so` - Module name complete with `.so`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_IFTIME

IFTIME()

Synopsis

Temporal Conditional.

Description

Returns the data following `?` if true, else the data following `:`

Syntax

```
IFTIME(timespec?[true][:false])
```

Arguments

- `timespec`
- `retvalue`
    - `true`
    - `false`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_IMPORT

IMPORT()

Retrieve the value of a variable from another channel.

Description

Syntax

```
IMPORT(channel,variable)
```

Arguments

- channel
- variable

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_INC

INC()

Synopsis

Increments the value of a variable, while returning the updated value to the dialplan

Description

Increments the value of a variable, while returning the updated value to the dialplan

Example: INC(MyVAR) - Increments MyVar

Note: INC(${MyVAR}) - Is wrong, as INC expects the variable name, not its value

Syntax

```
INC(variable)
```

Arguments

- variable - The variable name to be manipulated, without the braces.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_ISNULL

ISNULL()

Check if a value is NULL.

Description

Returns `1` if NULL or `0` otherwise.

Syntax

```
ISNULL(data)
```

Arguments

- `data`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_JABBER_RECEIVE

JABBER_RECEIVE()

Synopsis

Reads XMPP messages.

Description

Receives a text message on the given *account* from the buddy identified by *jid* and returns the contents.

Example: ${JABBER_RECEIVE(asterisk,bob@domain.com)} returns an XMPP message sent from *bob@domain.com* (or nothing in case of a time out), to the *asterisk* XMPP account configured in jabber.conf.

Syntax

```
JABBER_RECEIVE(account,jid[,timeout])
```

Arguments

- `account` - The local named account to listen on (specified in jabber.conf)
- `jid` - Jabber ID of the buddy to receive message from. It can be a bare JID (username@domain) or a full JID (username@domain/resource).
- `timeout` - In seconds, defaults to `20`.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_JABBER_STATUS

JABBER_STATUS()

Synopsis

Retrieves a buddy's status.

Description

Retrieves the numeric status associated with the buddy identified by *jid*. If the buddy does not exist in the buddylist, returns 7.

Status will be 1-7.

1=Online, 2=Chatty, 3=Away, 4=XAway, 5=DND, 6=Offline

If not in roster variable will be set to 7.

Example: ${JABBER_STATUS(asterisk,bob@domain.com)} returns 1 if *bob@domain.com* is online. *asterisk* is the associated XMPP account configured in jabber.conf.

Syntax

```
JABBER_STATUS(account,jid)
```

Arguments

- `account` - The local named account to listen on (specified in jabber.conf)
- `jid` - Jabber ID of the buddy to receive message from. It can be a bare JID (username@domain) or a full JID (username@domain/resource).

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_KEYPADHASH

KEYPADHASH()

Synopsis

Hash the letters in string into equivalent keypad numbers.

Description

Example: ${KEYPADHASH(Les)} returns "537"

Syntax

```
KEYPADHASH(string)
```

Arguments

- `string`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_LEN

LEN()

Synopsis

Return the length of the string given.

Description

Example: ${LEN(example)} returns 7

Syntax

```
LEN(string)
```

Arguments

- `string`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_LISTFILTER

LISTFILTER()

Synopsis

Remove an item from a list, by name.

Remove *value* from the list contained in the *varname* variable, where the list delimiter is specified by the *delim* parameter. This is very useful for removing a single channel name from a list of channels, for example.

Syntax

```
LISTFILTER(varname,delim,value)
```

Arguments

- varname
- delim
- value

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_LOCAL

LOCAL()

Synopsis

Manage variables local to the gosub stack frame.

Description

Read and write a variable local to the gosub stack frame, once we Return() it will be lost (or it will go back to whatever value it had before the Gosub()).

Syntax

```
LOCAL(varname)
```

Arguments

- varname

See Also

Application_Gosub
Application_GosubIf
Application_Return

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_LOCAL_PEEK

LOCAL_PEEK()

Synopsis

Retrieve variables hidden by the local gosub stack frame.

Description

Read a variable *varname* hidden by *n* levels of gosub stack frames. Note that ${LOCAL_PEEK(0,foo)} is the same as Read a variable `None - foo`, since the value of *n* peeks under 0 levels of stack frames; in other words, 0 is the current level. If *n* exceeds the available number of stack frames, then an empty string is returned.

Syntax

```
LOCAL_PEEK(n,varname)
```

Arguments

- `n`
- `varname`

See Also

Application_Gosub
Application_GosubIf
Application_Return

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_LOCK

LOCK()

Synopsis

Attempt to obtain a named mutex.

Description

Attempts to grab a named lock exclusively, and prevents other channels from obtaining the same lock. LOCK will wait for the lock to become available. Returns `1` if the lock was obtained or `0` on error.

To avoid the possibility of a deadlock, LOCK will only attempt to obtain the lock for 3 seconds if the channel already has another lock.To avoid the possibility of a deadlock, LOCK will only

attempt to obtain the lock for 3 seconds if the channel already has another lock.

```
LOCK(lockname)
```

Arguments

- `lockname`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_MAILBOX_EXISTS

MAILBOX_EXISTS()

Synopsis

Tell if a mailbox is configured.

Description

Returns a boolean of whether the corresponding *mailbox* exists. If *context* is not specified, defaults to the `default` context.

Syntax

```
MAILBOX_EXISTS(mailbox[,context])
```

Arguments

- `mailbox`
- `context`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_MASTER_CHANNEL

MASTER_CHANNEL()

Synopsis

Gets or sets variables on the master channel

Description

Allows access to the channel which created the current channel, if any. If the channel is already a master channel, then accesses local channel variables.

Syntax

```
MASTER_CHANNEL()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_MATH

MATH()

Synopsis

Performs Mathematical Functions.

Description

Performs mathematical functions based on two parameters and an operator. The returned value type is *type*

Example: Set(i=${MATH(123%16,int)}) - sets var i=11

Syntax

```
MATH(expression[,type])
```

Arguments

- `expression` - Is of the form: *number1 op number2* where the possible values for *op* are:
  +,-,/,*,%,<<,>>,^,AND,OR,XOR,<,%gt;,>=,<=,== (and behave as their C equivalents)
- `type` - Wanted type of result: f, float - float(default) i, int - integer h, hex - hex c, char - char

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_MD5

MD5()

Synopsis

Computes an MD5 digest.

Computes an MD5 digest.

```
MD5(data)
```

Arguments

- `data`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_MEETME_INFO

MEETME_INFO()

Query a given conference of various properties.

```
MEETME_INFO(keyword,confno)
```

Arguments

- `keyword` - Options:
    - `lock` - Boolean of whether the corresponding conference is locked.
    - `parties` - Number of parties in a given conference
    - `activity` - Duration of conference in seconds.
    - `dynamic` - Boolean of whether the corresponding conference is dynamic.
- `confno` - Conference number to retrieve information from.

Application_MeetMe
Application_MeetMeCount
Application_MeetMeAdmin
Application_MeetMeChannelAdmin

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_MINIVMACCOUNT

MINIVMACCOUNT()

Synopsis

Gets MiniVoicemail account information.

Description

Syntax

```
MINIVMACCOUNT(account,item)
```

Arguments

- `account`
- `item` - Valid items are:
    - `path` - Path to account mailbox (if account exists, otherwise temporary mailbox).
    - `hasaccount` - 1 is static Minivm account exists, 0 otherwise.
    - `fullname` - Full name of account owner.
    - `email` - Email address used for account.
    - `etemplate` - Email template for account (default template if none is configured).
    - `ptemplate` - Pager template for account (default template if none is configured).
    - `accountcode` - Account code for the voicemail account.
    - `pincode` - Pin code for voicemail account.
    - `timezone` - Time zone for voicemail account.
    - `language` - Language for voicemail account.
    - `<channel variable name>` - Channel variable value (set in configuration for account).

See Also

Application_MinivmRecord
Application_MinivmGreet
Application_MinivmNotify
Application_MinivmDelete
Application_MinivmAccMess
Application_MinivmMWI
Function_MINIVMCOUNTER

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_MINIVMCOUNTER

MINIVMCOUNTER()

Synopsis

Reads or sets counters for MiniVoicemail message.

Description

The operation is atomic and the counter is locked while changing the value. The counters are stored as text files in the minivm account directories. It might be better to use realtime functions if you are using a database to operate your Asterisk.

```
MINIVMCOUNTER(account,name[,operand])
```

Arguments

- `account` - If account is given and it exists, the counter is specific for the account. If account is a domain and the domain directory exists, counters are specific for a domain.
- `name` - The name of the counter is a string, up to 10 characters.
- `operand` - The counters never goes below zero. Valid operands for changing the value of a counter when assigning a value are:
    - `i` - Increment by value.
    - `d` - Decrement by value.
    - `s` - Set to value.

See Also

Application_MinivmRecord
Application_MinivmGreet
Application_MinivmNotify
Application_MinivmDelete
Application_MinivmAccMess
Application_MinivmMWI
Function_MINIVMACCOUNT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_MUTEAUDIO

MUTEAUDIO()

Synopsis

Muting audio streams in the channel

Description

The MUTEAUDIO function can be used to mute inbound (to the PBX) or outbound audio in a call. Example:

MUTEAUDIO(in)=on MUTEAUDIO(in)=off

Syntax

```
MUTEAUDIO(direction)
```

Arguments

- `direction` - Must be one of
    - `in` - Inbound stream (to the PBX)
    - `out` - Outbound stream (from the PBX)
    - `all` - Both streams

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_ODBC

ODBC()

Synopsis

Controls ODBC transaction properties.

Description

The ODBC() function allows setting several properties to influence how a connected database processes transactions.

Syntax

```
ODBC(property[,argument])
```

Arguments

- `property`
  - `transaction` - Gets or sets the active transaction ID. If set, and the transaction ID does not exist and a *database name* is specified as an argument, it will be created.
  - `forcecommit` - Controls whether a transaction will be automatically committed when the channel hangs up. Defaults to false. If a *transaction ID* is specified in the optional argument, the property will be applied to that ID, otherwise to the current active ID.
  - `isolation` - Controls the data isolation on uncommitted transactions. May be one of the following: `read_committed`, `read_uncommitted`, `repeatable_read`, or `serializable`. Defaults to the database setting in `res_odbc.conf` or `read_committed` if not specified. If a *transaction ID* is specified as an optional argument, it will be applied to that ID, otherwise the current active ID.
- `argument`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_ODBC_FETCH

ODBC_FETCH()

Synopsis

Fetch a row from a multirow query.

Description

For queries which are marked as mode=multirow, the original query returns a *result-id* from which results may be fetched. This function implements the actual fetch of the results.

This also sets This also sets `None` - `ODBC_FETCH_STATUS`.

- `ODBC_FETCH_STATUS` -
  - `SUCESS` - If rows are available.
  - `FAILURE` - If no rows are available.

Syntax

```
ODBC_FETCH(result-id)
```

Arguments

- `result-id`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_PASSTHRU

PASSTHRU()

Synopsis

Pass the given argument back as a value.

Description

Literally returns the given *string*. The intent is to permit other dialplan functions which take a variable name as an argument to be able to take a literal string, instead.

Syntax

```
PASSTHRU([string])
```

Arguments

- `string`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_PITCH_SHIFT

PITCH_SHIFT()

Synopsis

Pitch shift both tx and rx audio streams on a channel.

Examples:

exten => 1,1,Set(PITCH_SHIFT(tx)=highest); raises pitch an octave

exten => 1,1,Set(PITCH_SHIFT(rx)=higher) ; raises pitch more

exten => 1,1,Set(PITCH_SHIFT(both)=high) ; raises pitch

exten => 1,1,Set(PITCH_SHIFT(rx)=low) ; lowers pitch

exten => 1,1,Set(PITCH_SHIFT(tx)=lower) ; lowers pitch more

exten => 1,1,Set(PITCH_SHIFT(both)=lowest) ; lowers pitch an octave

exten => 1,1,Set(PITCH_SHIFT(rx)=0.8) ; lowers pitch

exten => 1,1,Set(PITCH_SHIFT(tx)=1.5) ; raises pitch

Syntax

```
PITCH_SHIFT(channel direction)
```

Arguments

- `channel direction` - Direction can be either `rx`, `tx`, or `both`. The direction can either be set to a valid floating point number between 0.1 and 4.0 or one of the enum values listed below. A value of 1.0 has no effect. Greater than 1 raises the pitch. Lower than 1 lowers the pitch. The pitch amount can also be set by the following values
    - `highest`
    - `higher`
    - `high`
    - `low`
    - `lower`
    - `lowest`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_POP

POP()

Synopsis

Removes and returns the last item off of a variable containing delimited text

Description

Example:

```
exten => s,1,Set(array=one,two,three)

exten => s,n,While($["$\{SET(var=$\{POP(array)\})\}" != ""])

exten => s,n,NoOp(var is ${var})

exten => s,n,EndWhile
```

This would iterate over each value in array, right to left, and would result in NoOp(var is three), NoOp(var is two), and NoOp(var is one) being executed.

Syntax

```
POP(varname[,delimiter])
```

Arguments

- varname
- delimiter

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_PP_EACH_EXTENSION

PP_EACH_EXTENSION()

Synopsis

Execute specified template for each extension.

Description

Output the specified template for each extension associated with the specified MAC address.

Syntax

```
PP_EACH_EXTENSION(mac,template)
```

Arguments

- mac
- template

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_PP_EACH_USER

PP_EACH_USER()

Synopsis

Generate a string for each phoneprov user.

Description

Pass in a string, with phoneprov variables you want substituted in the format of %{VARNAME}, and you will get the string rendered for each user in phoneprov excluding ones with MAC address *exclude_mac*. Probably not useful outside of res_phoneprov.

Example: ${PP_EACH_USER(<item><fn>%{DISPLAY_NAME}</fn></item>|${MAC})

Syntax

```
PP_EACH_USER(string,exclude_mac)
```

Arguments

- string
- exclude_mac

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_PUSH

PUSH()

Synopsis

Appends one or more values to the end of a variable containing delimited text

Description

Example: Set(PUSH(array)=one,two,three) would append one, two, and three to the end of the values stored in the variable "array".

Syntax

```
PUSH(varname[,delimiter])
```

Arguments

- varname
- delimiter

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_QUEUE_EXISTS

QUEUE_EXISTS()

Synopsis

Check if a named queue exists on this server

Description

Returns 1 if the specified queue exists, 0 if it does not

Syntax

```
QUEUE_EXISTS([queuename])
```

Arguments

- queuename

See Also

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS
Function_QUEUE_WAITING_COUNT
Function_QUEUE_MEMBER_LIST
Function_QUEUE_MEMBER_PENALTY

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_QUEUE_MEMBER

QUEUE_MEMBER()

Synopsis

Count number of members answering a queue.

Returns the number of members currently associated with the specified *queuename*.

```
QUEUE_MEMBER(queuename,option)
```

Arguments

- `queuename`
- `option`
    - `logged` - Returns the number of logged-in members for the specified queue.
    - `free` - Returns the number of logged-in members for the specified queue that either can take calls or are currently wrapping up after a previous call.
    - `ready` - Returns the number of logged-in members for the specified queue that are immediately available to answer a call.
    - `count` - Returns the total number of members for the specified queue.

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS
Function_QUEUE_WAITING_COUNT
Function_QUEUE_MEMBER_LIST
Function_QUEUE_MEMBER_PENALTY

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_QUEUE_MEMBER_COUNT

QUEUE_MEMBER_COUNT()

Count number of members answering a queue.

Returns the number of members currently associated with the specified *queuename*.

This function has been deprecated in favor of the `QUEUE_MEMBER()` function

Syntax

```
QUEUE_MEMBER_COUNT(queuename)
```

Arguments

- queuename

See Also

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS
Function_QUEUE_WAITING_COUNT
Function_QUEUE_MEMBER_LIST
Function_QUEUE_MEMBER_PENALTY

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_QUEUE_MEMBER_LIST

QUEUE_MEMBER_LIST()

Synopsis

Returns a list of interfaces on a queue.

Description

Returns a comma-separated list of members associated with the specified *queuename*.

Syntax

```
QUEUE_MEMBER_LIST(queuename)
```

Arguments

- queuename

See Also

Application_Queue

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_QUEUE_MEMBER_PENALTY

QUEUE_MEMBER_PENALTY()

Synopsis

Gets or sets queue members penalty.

Description

Gets or sets queue members penalty.

Syntax

```
QUEUE_MEMBER_PENALTY(queuename,interface)
```

Arguments

- `queuename`
- `interface`

See Also

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS
Function_QUEUE_WAITING_COUNT

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_QUEUE_VARIABLES

QUEUE_VARIABLES()

Synopsis

Return Queue information in variables.

Description

Makes the following queue variables available.

Returns `0` if queue is found and setqueuevar is defined, `-1` otherwise.

Syntax

```
QUEUE_VARIABLES(queuename)
```

Arguments

- `queuename`
  - `QUEUEMAX` - Maxmimum number of calls allowed.
  - `QUEUESTRATEGY` - The strategy of the queue.
  - `QUEUECALLS` - Number of calls currently in the queue.
  - `QUEUEHOLDTIME` - Current average hold time.
  - `QUEUECOMPLETED` - Number of completed calls for the queue.
  - `QUEUEABANDONED` - Number of abandoned calls.
  - `QUEUESRVLEVEL` - Queue service level.
  - `QUEUESRVLEVELPERF` - Current service level performance.

See Also

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS
Function_QUEUE_WAITING_COUNT
Function_QUEUE_MEMBER_LIST
Function_QUEUE_MEMBER_PENALTY

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_QUEUE_WAITING_COUNT

QUEUE_WAITING_COUNT()

Synopsis

Count number of calls currently waiting in a queue.

Description

Returns the number of callers currently waiting in the specified *queuename*.

Syntax

```
QUEUE_WAITING_COUNT([queuename])
```

Arguments

- queuename

See Also

Application_Queue
Application_QueueLog
Application_AddQueueMember
Application_RemoveQueueMember
Application_PauseQueueMember
Application_UnpauseQueueMember
Function_QUEUE_VARIABLES
Function_QUEUE_MEMBER
Function_QUEUE_MEMBER_COUNT
Function_QUEUE_EXISTS
Function_QUEUE_WAITING_COUNT
Function_QUEUE_MEMBER_LIST
Function_QUEUE_MEMBER_PENALTY

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_QUOTE

QUOTE()

Synopsis

Quotes a given string, escaping embedded quotes as necessary

Description

Example: ${QUOTE(ab"c"de)} will return "abcde"

```
QUOTE(string)
```

Arguments

- string

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_RAND

RAND()

Choose a random number in a range.

Choose a random number between *min* and *max*. *min* defaults to `0`, if not specified, while *max* defaults to `RAND_MAX` (2147483647 on many systems).

Example: Set(junky=${RAND(1,8)}); Sets junky to a random number between 1 and 8, inclusive.

```
RAND([min[,max]])
```

Arguments

- min
- max

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_REALTIME

REALTIME()

RealTime Read/Write Functions.

This function will read or write values from/to a RealTime repository. REALTIME(....) will read names/values from the repository, and REALTIME(....)= will write a new value/field to the repository. On a read, this function returns a delimited text string. The name/value pairs are delimited by *delim1*, and the name and value are delimited between each other with delim2. If there is no match, NULL will be returned by the function. On a write, this function will always return NULL.

Syntax

```
REALTIME(family,fieldmatch[,value[,delim1|field[,delim2]]])
```

Arguments

- `family`
- `fieldmatch`
- `value`
- `delim1|field` - Use *delim1* with *delim2* on read and *field* without *delim2* on write If we are reading and *delim1* is not specified, defaults to ,
- `delim2` - Parameter only used when reading, if not specified defaults to =

See Also

[Function_REALTIME_STORE](#)
[Function_REALTIME_DESTROY](#)
[Function_REALTIME_FIELD](#)
[Function_REALTIME_HASH](#)

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_REALTIME_DESTROY

REALTIME_DESTROY()

Synopsis

RealTime Destroy Function.

Description

This function acts in the same way as REALTIME(....) does, except that it destroys the matched record in the RT engine.

Syntax

```
REALTIME_DESTROY(family,fieldmatch[,value[,delim1[,delim2]]])
```

Arguments

- `family`
- `fieldmatch`
- `value`
- `delim1`
- `delim2`

See Also

Function_REALTIME
Function_REALTIME_STORE
Function_REALTIME_FIELD
Function_REALTIME_HASH

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_REALTIME_FIELD

REALTIME_FIELD()

Synopsis

RealTime query function.

Description

This function retrieves a single item, *fieldname* from the RT engine, where *fieldmatch* contains the value *value*. When written to, the REALTIME_FIELD() function performs identically to the REALTIME() function.

Syntax

```
REALTIME_FIELD(family,fieldmatch,value,fieldname)
```

Arguments

- `family`
- `fieldmatch`
- `value`
- `fieldname`

See Also

Function_REALTIME
Function_REALTIME_STORE
Function_REALTIME_DESTROY
Function_REALTIME_HASH

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_REALTIME_HASH

REALTIME_HASH()

Synopsis

RealTime query function.

Description

This function retrieves a single record from the RT engine, where *fieldmatch* contains the value *value* and formats the output suitably, such that it can be assigned to the HASH() function. The HASH() function then provides a suitable method for retrieving each field value of the record.

Syntax

```
REALTIME_HASH(family,fieldmatch,value)
```

Arguments

- family
- fieldmatch
- value

See Also

Function_REALTIME
Function_REALTIME_STORE
Function_REALTIME_DESTROY
Function_REALTIME_FIELD

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_REALTIME_STORE

REALTIME_STORE()

Synopsis

RealTime Store Function.

Description

This function will insert a new set of values into the RealTime repository. If RT engine provides an unique ID of the stored record, REALTIME_STORE(...)=.. creates channel variable named RTSTOREID, which contains value of unique ID. Currently, a maximum of 30 field/value pairs is supported.

Syntax

```
REALTIME_STORE(family,field1,fieldN[,...],field30)
```

Arguments

- `family`
- `field1`
- `fieldN`
- `field30`

See Also

Function_REALTIME
Function_REALTIME_DESTROY
Function_REALTIME_FIELD
Function_REALTIME_HASH

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_REDIRECTING

REDIRECTING()

Synopsis

Gets or sets Redirecting data on the channel.

Description

Gets or sets Redirecting data on the channel.

The allowable values for the *reason* field are the following:

Unknown

Call Forwarding Busy

Call Forwarding No Reply

Callee is Unavailable

Time of Day

Do Not Disturb

Call Deflection

Follow Me

Called DTE Out-Of-Order

Callee is Away

Call Forwarding By The Called DTE

Call Forwarding Unconditional

The allowable values for the *xxx-name-charset* field are the following:

Unknown

ISO8859-1

Withdrawn

ISO8859-2

ISO8859-3

ISO8859-4

ISO8859-5

ISO8859-7

ISO10646 Bmp String

ISO10646 UTF-8 String

Syntax

```
REDIRECTING(datatype[,i])
```

Arguments

- `datatype` - The allowable datatypes are:
    - `from-all`
    - `from-name`
    - `from-name-valid`
    - `from-name-charset`
    - `from-name-pres`
    - `from-num`
    - `from-num-valid`
    - `from-num-plan`
    - `from-num-pres`
    - `from-subaddr`
    - `from-subaddr-valid`
    - `from-subaddr-type`
    - `from-subaddr-odd`
    - `from-tag`
    - `to-all`
    - `to-name`
    - `to-name-valid`
    - `to-name-charset`
    - `to-name-pres`
    - `to-num`
    - `to-num-valid`
    - `to-num-plan`
    - `to-num-pres`

- to-subaddr
- to-subaddr-valid
- to-subaddr-type
- to-subaddr-odd
- to-tag
- reason
- count
- `i` - If set, this will prevent the channel from sending out protocol messages because of the value being set

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_REGEX

REGEX()

Synopsis

Check string against a regular expression.

Description

Return `1` on regular expression match or `0` otherwise

Please note that the space following the double quotes separating the regex from the data is optional and if present, is skipped. If a space is desired at the beginning of the data, then put two spaces there; the second will not be skipped.

Syntax

```
REGEX("regular expression",string)
```

Arguments

- `"regular expression"`
- `string`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_REPLACE

REPLACE()

Synopsis

Replace a set of characters in a given string with another character.

Description

Iterates through a string replacing all the *find-chars* with *replace-char*. *replace-char* may be either empty or contain one character. If empty, all *find-chars* will be deleted from the output.

The replacement only occurs in the output. The original variable is not altered.The replacement only occurs in the output. The original variable is not altered.

Syntax

```
REPLACE(varname,find-chars[,replace-char])
```

Arguments

- varname
- find-chars
- replace-char

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SET

SET()

Synopsis

SET assigns a value to a channel variable.

Description

Syntax

```
SET(varname[,value])
```

Arguments

- varname
- value

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SHA1

SHA1()

Synopsis

Computes a SHA1 digest.

Generate a SHA1 digest via the SHA1 algorythm.

Example: Set(sha1hash=${SHA1(junky)})

Sets the asterisk variable sha1hash to the string `60fa5675b9303eb62f99a9cd47f9f5837d18f9a0` which is known as his hash

```
SHA1(data)
```

Arguments

- `data` - Input string

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SHARED

SHARED()

Gets or sets the shared variable specified.

Implements a shared variable area, in which you may share variables between channels.

The variables used in this space are separate from the general namespace of the channel and thus The variables used in this space are separate from the general namespace of the channel and thus `None` - `SHARED(foo)` and The variables used in this space are separate from the general namespace of the channel and thus `None` - `foo` represent two completely different variables, despite sharing the same name.

Finally, realize that there is an inherent race between channels operating at the same time, fiddling with each others' internal variables, which is why this special variable namespace exists; it is to remind you that variables in the SHARED namespace may change at any time, without warning. You should therefore take special care to ensure that when using the SHARED namespace, you retrieve the variable and store it in a regular channel variable before using it in a set of calculations (or you might be surprised by the result).

```
SHARED(varname[,channel])
```

Arguments

- `varname` - Variable name
- `channel` - If not specified will default to current channel. It is the complete channel name: `SIP/12-abcd1234` or the prefix only `SIP/12`.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SHELL

SHELL()

Synopsis

Executes a command as if you were at a shell.

Description

Returns the value from a system command

Example:
```
Set(foo=${SHELL(echo \bar)})
```

When using the SHELL() dialplan function, your \SHELL\ is /bin/sh, which may differ as to the underlying shell, depending upon your production platform. Also keep in mind that if you are using a common path, you should be mindful of race conditions that could result from two calls running SHELL() simultaneously.When using the SHELL() dialplan function, your \SHELL\ is /bin/sh, which may differ as to the underlying shell, depending upon your production platform. Also keep in mind that if you are using a common path, you should be mindful of race conditions that could result from two calls running SHELL() simultaneously.

Syntax

```
SHELL(command)
```

Arguments

- `command` - This is the argument to the function, the command you want to pass to the shell.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SHIFT

SHIFT()

Synopsis

Removes and returns the first item off of a variable containing delimited text

Description

Example:

exten => s,1,Set(array=one,two,three)

exten => s,n,While($["$\{SET(var=$\{SHIFT(array)\})\}" != ""])

exten => s,n,NoOp(var is ${var})

exten => s,n,EndWhile

This would iterate over each value in array, left to right, and would result in NoOp(var is one), NoOp(var is two), and NoOp(var is three) being executed.

Syntax

```
SHIFT(varname[,delimiter])
```

Arguments

- varname
- delimiter

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SIP_HEADER

SIP_HEADER()

Synopsis

Gets the specified SIP header.

Description

Since there are several headers (such as Via) which can occur multiple times, SIP_HEADER takes an optional second argument to specify which header with that name to retrieve. Headers start at offset 1.

Syntax

```
SIP_HEADER(name[,number])
```

Arguments

- `name`
- `number` - If not specified, defaults to `1`.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SIPCHANINFO

SIPCHANINFO()

Synopsis

Gets the specified SIP parameter from the current channel.

Description

Syntax

```
SIPCHANINFO(item)
```

Arguments

- `item`
  - `peerip` - The IP address of the peer.
  - `recvip` - The source IP address of the peer.
  - `from` - The URI from the `From:` header.
  - `uri` - The URI from the `Contact:` header.
  - `useragent` - The useragent.
  - `peername` - The name of the peer.
  - `t38passthrough` - `1` if T38 is offered or enabled in this channel, otherwise `0`.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SIPPEER

SIPPEER()

Synopsis

Gets SIP peer information.

Description

Syntax

```
SIPPEER(peername[,item])
```

Arguments

- `peername`
- `item`
  - `ip` - (default) The ip address.
  - `port` - The port number.
  - `mailbox` - The configured mailbox.
  - `context` - The configured context.
  - `expire` - The epoch time of the next expire.
  - `dynamic` - Is it dynamic? (yes/no).
  - `callerid_name` - The configured Caller ID name.
  - `callerid_num` - The configured Caller ID number.
  - `callgroup` - The configured Callgroup.
  - `pickupgroup` - The configured Pickupgroup.
  - `codecs` - The configured codecs.
  - `status` - Status (if qualify=yes).
  - `regexten` - Registration extension.
  - `limit` - Call limit (call-limit).
  - `busylevel` - Configured call level for signalling busy.
  - `curcalls` - Current amount of calls. Only available if call-limit is set.
  - `language` - Default language for peer.
  - `accountcode` - Account code for this peer.
  - `useragent` - Current user agent id for peer.
  - `maxforwards` - The value used for SIP loop prevention in outbound requests
  - `chanvar[name]` - A channel variable configured with setvar for this peer.
  - `codec[x]` - Preferred codec index number *x* (beginning with zero).

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SMDI_MSG

SMDI_MSG()

Synopsis

Retrieve details about an SMDI message.

Description

This function is used to access details of an SMDI message that was pulled from the incoming SMDI message queue using the SMDI_MSG_RETRIEVE() function.

Syntax

```
SMDI_MSG(message_id,component)
```

Arguments

- `message_id`
- `component` - Valid message components are:
  - `number` - The message desk number
  - `terminal` - The message desk terminal
  - `station` - The forwarding station

- `callerid` - The callerID of the calling party that was forwarded
- `type` - The call type. The value here is the exact character that came in on the SMDI link. Typically, example values are:
  Options:
  - `D` - Direct Calls
  - `A` - Forward All Calls
  - `B` - Forward Busy Calls
  - `N` - Forward No Answer Calls

See Also

Function_SMDI_MSG_RETRIEVE

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SMDI_MSG_RETRIEVE

SMDI_MSG_RETRIEVE()

Synopsis

Retrieve an SMDI message.

Description

This function is used to retrieve an incoming SMDI message. It returns an ID which can be used with the SMDI_MSG() function to access details of the message. Note that this is a destructive function in the sense that once an SMDI message is retrieved using this function, it is no longer in the global SMDI message queue, and can not be accessed by any other Asterisk channels. The timeout for this function is optional, and the default is 3 seconds. When providing a timeout, it should be in milliseconds.

The default search is done on the forwarding station ID. However, if you set one of the search key options in the options field, you can change this behavior.

Syntax

```
SMDI_MSG_RETRIEVE(smdi port,search key[,timeout[,options]])
```

Arguments

- `smdi port`
- `search key`
- `timeout`
- `options`
  - `t` - Instead of searching on the forwarding station, search on the message desk terminal.
  - `n` - Instead of searching on the forwarding station, search on the message desk number.

See Also

Function_SMDI_MSG

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SORT

SORT()

Synopsis

Sorts a list of key/vals into a list of keys, based upon the vals.

Description

Takes a comma-separated list of keys and values, each separated by a colon, and returns a comma-separated list of the keys, sorted by their values. Values will be evaluated as floating-point numbers.

Syntax

```
SORT(key1val1[,key2val2[,...]])
```

Arguments

- keyval
    - key1
    - val1
- keyvaln
    - key2
    - val2

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SPEECH

SPEECH()

Synopsis

Gets information about speech recognition results.

Description

Gets information about speech recognition results.

Syntax

```
SPEECH(argument)
```

Arguments

- argument

- `status` - Returns `1` upon speech object existing, or `0` if not
- `spoke` - Returns `1` if spoker spoke, or `0` if not
- `results` - Returns number of results that were recognized.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SPEECH_ENGINE

SPEECH_ENGINE()

Synopsis

Change a speech engine specific attribute.

Description

Changes a speech engine specific attribute.

Syntax

```
SPEECH_ENGINE(name)
```

Arguments

- `name`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SPEECH_GRAMMAR

SPEECH_GRAMMAR()

Synopsis

Gets the matched grammar of a result if available.

Description

Gets the matched grammar of a result if available.

Syntax

```
SPEECH_GRAMMAR([nbest_number,result_number])
```

Arguments

- nbest_number
- result_number

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SPEECH_RESULTS_TYPE

SPEECH_RESULTS_TYPE()

Synopsis

Sets the type of results that will be returned.

Description

Sets the type of results that will be returned. Valid options are normal or nbest.

Syntax

```
SPEECH_RESULTS_TYPE()
```

Arguments

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SPEECH_SCORE

SPEECH_SCORE()

Synopsis

Gets the confidence score of a result.

Description

Gets the confidence score of a result.

Syntax

```
SPEECH_SCORE([nbest_number,result_number])
```

Arguments

- nbest_number
- result_number

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SPEECH_TEXT

SPEECH_TEXT()

Synopsis

Gets the recognized text of a result.

Description

Gets the recognized text of a result.

Syntax

```
SPEECH_TEXT([nbest_number,result_number])
```

Arguments

- `nbest_number`
- `result_number`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SPRINTF

SPRINTF()

Synopsis

Format a variable according to a format string.

Description

Parses the format string specified and returns a string matching that format. Supports most options found in **sprintf(3)**. Returns a shortened string if a format specifier is not recognized.

Syntax

```
SPRINTF(format,arg1[,arg2[,...][,argN]])
```

Arguments

- `format`
- `arg1`

- arg2
- argN

```
sprintf(3)
```

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SQL_ESC

SQL_ESC()

Synopsis

Escapes single ticks for use in SQL statements.

Description

Used in SQL templates to escape data which may contain single ticks ' which are otherwise used to delimit data.

Example: SELECT foo FROM bar WHERE baz='${SQL_ESC(${ARG1})}'

Syntax

```
SQL_ESC(string)
```

Arguments

- string

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SRVQUERY

SRVQUERY()

Synopsis

Initiate an SRV query.

Description

This will do an SRV lookup of the given service.

Syntax

```
SRVQUERY(service)
```

Arguments

- `service` - The service for which to look up SRV records. An example would be something like `_sip._udp.example.com`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SRVRESULT

SRVRESULT()

Synopsis

Retrieve results from an SRVQUERY.

Description

This function will retrieve results from a previous use of the SRVQUERY function.

Syntax

```
SRVRESULT(id,resultnum)
```

Arguments

- `id` - The identifier returned by the SRVQUERY function.
- `resultnum` - The number of the result that you want to retrieve. Results start at `1`. If this argument is specified as `getnum`, then it will return the total number of results that are available.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_STAT

STAT()

Synopsis

Does a check on the specified file.

Description

Syntax

```
STAT(flag,filename)
```

Arguments

- `flag` - Flag may be one of the following: d - Checks if the file is a directory. e - Checks if the file exists. f - Checks if the file is a regular file. m - Returns the file mode (in octal) s - Returns the size (in bytes) of the file A - Returns the epoch at which the file was last accessed. C - Returns the epoch at which the inode was last changed. M - Returns the epoch at which the file was last modified.
- `filename`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_STRFTIME

STRFTIME()

Synopsis

Returns the current date/time in the specified format.

Description

STRFTIME supports all of the same formats as the underlying C function **strftime(3)**. It also supports the following format:
`%[n]q` - fractions of a second, with leading zeros.

Example:
`%3q` will give milliseconds and `%1q` will give tenths of a second. The default is set at milliseconds (n=3). The common case is to use it in combination with %S, as in `%S.%3q`.

Syntax

```
STRFTIME([epoch[,timezone[,format]]])
```

Arguments

- `epoch`
- `timezone`
- `format`

See Also

`strftime(3)`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_STRPTIME

STRPTIME()

Returns the epoch of the arbitrary date/time string structured as described by the format.

Description

This is useful for converting a date into `EPOCH` time, possibly to pass to an application like SayUnixTime or to calculate the difference between the two date strings

Example: ${STRPTIME(2006-03-01 07:30:35,America/Chicago,%Y-%m-%d %H:%M:%S)} returns 1141219835

Syntax

```
STRPTIME(datetime,timezone,format)
```

Arguments

- `datetime`
- `timezone`
- `format`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_SYSINFO

SYSINFO()

Synopsis

Returns system information specified by parameter.

Description

Returns information from a given parameter.

Syntax

```
SYSINFO(parameter)
```

Arguments

- `parameter`
  - `loadavg` - System load average from past minute.
  - `numcalls` - Number of active calls currently in progress.
  - `uptime` - System uptime in hours. This parameter is dependant upon operating system.This parameter is dependant upon operating system.
  - `totalram` - Total usable main memory size in KiB. This parameter is dependant upon operating system.This parameter is

- `freeram` - Available memory size in KiB. This parameter is dependant upon operating system.This parameter is dependant upon operating system.
- `bufferram` - Memory used by buffers in KiB. This parameter is dependant upon operating system.This parameter is dependant upon operating system.
- `totalswap` - Total swap space still available in KiB. This parameter is dependant upon operating system.This parameter is dependant upon operating system.
- `freeswap` - Free swap space still available in KiB. This parameter is dependant upon operating system.This parameter is dependant upon operating system.
- `numprocs` - Number of current processes. This parameter is dependant upon operating system.This parameter is dependant upon operating system.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_TESTTIME

TESTTIME()

Synopsis

Sets a time to be used with the channel to test logical conditions.

Description

To test dialplan timing conditions at times other than the current time, use this function to set an alternate date and time. For example, you may wish to evaluate whether a location will correctly identify to callers that the area is closed on Christmas Day, when Christmas would otherwise fall on a day when the office is normally open.

Syntax

```
TESTTIME(date time[,zone])
```

Arguments

- `date` - Date in ISO 8601 format
- `time` - Time in HH:MM:SS format (24-hour time)
- `zone` - Timezone name

See Also

Application_GotoIfTime

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_TIMEOUT

TIMEOUT()

Synopsis

Gets or sets timeouts on the channel. Timeout values are in seconds.

The timeouts that can be manipulated are:
`absolute` : The absolute maximum amount of time permitted for a call. Setting of 0 disables the timeout. `digit` : 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 would be passed to the `i` extension, or if it doesn't exist the call would be terminated). The default timeout is 5 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 would be terminated. The default timeout is 10 seconds.

Syntax

```
TIMEOUT(timeouttype)
```

Arguments

- `timeouttype` - The timeout that will be manipulated. The possible timeout types are:
  `absolute`, `digit` or `response`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_TOLOWER

TOLOWER()

Synopsis

Convert string to all lowercase letters.

Description

Example: ${TOLOWER(Example)} returns "example"

Syntax

```
TOLOWER(string)
```

Arguments

- `string`

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_TOUPPER

TOUPPER()

Synopsis

Convert string to all uppercase letters.

Description

Example: ${TOUPPER(Example)} returns "EXAMPLE"

Syntax

```
TOUPPER(string)
```

Arguments

- string

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_TRYLOCK

TRYLOCK()

Synopsis

Attempt to obtain a named mutex.

Description

Attempts to grab a named lock exclusively, and prevents other channels from obtaining the same lock. Returns 1 if the lock was available or 0 otherwise.

Syntax

```
TRYLOCK(lockname)
```

Arguments

- lockname

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_TXTCIDNAME

TXTCIDNAME()

Synopsis

TXTCIDNAME looks up a caller name via DNS.

Description

This function looks up the given phone number in DNS to retrieve the caller id name. The result will either be blank or be the value found in the TXT record in DNS.

Syntax

```
TXTCIDNAME(number[,zone-suffix])
```

Arguments

- `number`
- `zone-suffix` - If no *zone-suffix* is given, the default will be `e164.arpa`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_UNLOCK

UNLOCK()

Synopsis

Unlocks a named mutex.

Description

Unlocks a previously locked mutex. Returns `1` if the channel had a lock or `0` otherwise.

It is generally unnecessary to unlock in a hangup routine, as any locks held are automatically freed when the channel is destroyed.It is generally unnecessary to unlock in a hangup routine, as any locks held are automatically freed when the channel is destroyed.

Syntax

```
UNLOCK(lockname)
```

Arguments

- `lockname`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_UNSHIFT

UNSHIFT()

Synopsis

Inserts one or more values to the beginning of a variable containing delimited text

Description

Example: Set(UNSHIFT(array)=one,two,three) would insert one, two, and three before the values stored in the variable "array".

Syntax

```
UNSHIFT(varname[,delimiter])
```

Arguments

- `varname`
- `delimiter`

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_URIDECODE

URIDECODE()

Synopsis

Decodes a URI-encoded string according to RFC 2396.

Description

Returns the decoded URI-encoded *data* string.

```
URIDECODE(data)
```

Arguments

- `data` - Input string to be decoded.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_URIENCODE

URIENCODE()

Synopsis

Encodes a string to URI-safe encoding according to RFC 2396.

Description

Returns the encoded string defined in *data*.

Syntax

```
URIENCODE(data)
```

Arguments

- `data` - Input string to be encoded.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_VALID_EXTEN

VALID_EXTEN()

Synopsis

Determine whether an extension exists or not.

Description

Returns a true value if the indicated *context*, *extension*, and *priority* exist.

Syntax

```
VALID_EXTEN([context,extension[,priority]])
```

Arguments

- `context` - Defaults to the current context
- `extension`
- `priority` - Priority defaults to `1`.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_VERSION

VERSION()

Synopsis

Return the Version info for this Asterisk.

Description

If there are no arguments, return the version of Asterisk in this format: SVN-branch-1.4-r44830M

Example: Set(junky=${VERSION()};

Sets junky to the string `SVN-branch-1.6-r74830M`, or possibly, `SVN-trunk-r45126M`.

Syntax

```
VERSION([info])
```

Arguments

- `info` - The possible values are:
  - `ASTERISK_VERSION_NUM` - A string of digits is returned (right now fixed at 999999).
  - `BUILD_USER` - The string representing the user's name whose account was used to configure Asterisk, is returned.
  - `BUILD_HOSTNAME` - The string representing the name of the host on which Asterisk was configured, is returned.
  - `BUILD_MACHINE` - The string representing the type of machine on which Asterisk was configured, is returned.
  - `BUILD_OS` - The string representing the OS of the machine on which Asterisk was configured, is returned.
  - `BUILD_DATE` - The string representing the date on which Asterisk was configured, is returned.
  - `BUILD_KERNEL` - The string representing the kernel version of the machine on which Asterisk was configured, is returned.

See Also

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Function_VMCOUNT

VMCOUNT()

Count the voicemails in a specified mailbox.

Count the number of voicemails in a specified mailbox, you could also specify the *context* and the mailbox *folder*.

Example:
```
exten => s,1,Set(foo=${VMCOUNT(125)})
```

```
VMCOUNT(vmbox[@context][,folder])
```

Arguments

- `vmbox`
    - `vmbox`
    - `context` - If not specified, defaults to `default`.
- `folder` - If not specified, defaults to `INBOX`

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Set the TX or RX volume of a channel.

The VOLUME function can be used to increase or decrease the `tx` or `rx` gain of any channel.

For example:

Set(VOLUME(TX)=3)

Set(VOLUME(RX)=2)

Set(VOLUME(TX,p)=3)

Set(VOLUME(RX,p)=3>

```
VOLUME(direction[,options])
```

Arguments

- `direction` - Must be `TX` or `RX`.
- `options`
    - `p` - Enable DTMF volume control

Import Version

This documentation was imported from Asterisk version SVN-branch-1.8-r311874.

Configuration and Operation

Here be the top-level page for all of the Asterisk Reference Information as found in the doc/ and doc/tex subdirectories of the Asterisk source.

It's been there all along, but now it's here, in an easy to view format (no need to install 800MB of dependancies in Debian just to convert .tex into PDF), that's also searchable. Hoo-ray!

Asterisk Calendaring

The Asterisk Calendaring API aims to be a generic interface for integrating Asterisk with various calendaring technologies. The goal is to be able to support reading and writing of calendar events as well as allowing notification of pending events through the Asterisk dialplan.

There are three calendaring modules that ship with Asterisk that provide support for iCalendar, CalDAV, and Microsoft Exchange Server calendars. All three modules support event notification. Both CalDAV and Exchange support reading and writing calendars, while iCalendar is a read-only format.

Configuring Asterisk Calendaring

All asterisk calendaring modules are configured through calender.conf. Each calendar module can define its own set of required parameters in addition to the parameters available to all calendar types. An effort has been made to keep all options the same in all calendaring modules, but some options will diverge over time as features are added to each module.
An example calendar.conf might look like:

```
[calendar_joe]
type = ical
url = https://example.com/home/jdoe/Calendar
user = jdoe
secret = mysecret
refresh = 15
timeframe = 600
autoreminder = 10
channel = SIP/joe
context = calendar_event_notify
extension = s
waittime = 30
```

Module-independent settings

The settings related to calendar event notification are handled by the core calendaring API. These settings are:

- autoreminder - This allows the overriding of any alarms that may or may not be set for a calendar event. It is specified in minutes.

- refresh - How often to refresh the calendar data; specified in minutes.

- timeframe - How far into the future each calendar refresh should look. This is the amount of data that will be visible to queries from the dialplan. This setting should always be greater than or equal to the refresh setting or events may be missed. It is specified in minutes.

- channel - The channel that should be used for making the notification attempt.

- waittime - How long to wait, in seconds, for the channel to answer a notification attempt. There are two ways to specify how to handle a notification. One option is providing a context and extension, while the other is providing an application and the arguments to that application. One (and only one) of these options should be provided.

- context - The context of the extension to connect to the notification channel

- extension - The extension to connect to the notification. Note that the priority will always be 1.

- app - The dialplan application to execute upon the answer of a notification

- appdata - The data to pass to the notification dialplan application

Module-dependent settings

Connection-related options are specific to each module. Currently, all modules take a url, user, and secret for configuration and no other module-specific settings have been implemented. At this time, no support for HTTP redirects has been implemented, so it is important to specify the correct URL-paying attention to any trailing slashes that may be necessary.

Calendaring Dialplan Functions

Read functions

The simplest dialplan query is the CALENDAR_BUSY query. It takes a single option, the name of the calendar defined, and returns "1" for busy (including tentatively busy) and "0" for not busy.

For more information about a calendar event, a combination of CALENDAR_QUERY and CALENDAR_QUERY_RESULT is used. CALENDAR_QUERY takes the calendar name and optionally a start and end time in "unix time" (seconds from unix epoch). It returns an id that can be passed to CALENDAR_QUERY_RESULT along with a field name to return the data in that field. If multiple events are returned in the query, the number of the event in the list can be specified as well. The available fields to return are:

- summary - A short summary of the event

- description - The full description of the event

- organizer - Who organized the event

- location - Where the event is located

- calendar - The name of the calendar from calendar.conf

- uid - The unique identifier associated with the event

- start - The start of the event in seconds since Unix epoch

- end - The end of the event in seconds since Unix epoch

- busystate - The busy state 0=Free, 1=Tentative, 2=Busy

- attendees - A comma separated list of attendees as stored in the event and may include prefixes such as "mailto:".

When an event notification is sent to the dial plan, the CALENDAR_EVENT function may be used to return the information about the event that is causing the notification. The fields that can be returned are the same as those from CALENDAR_QUERY_RESULT.

Write functions

To write an event to a calendar, the CALENDAR_WRITE function is used. This function takes a calendar name and also uses the same fields as CALENDAR_QUERY_RESULT. As a write function, it takes a set of comma-separated values that are in the same order as the specified fields. For example:

```
CALENDAR_WRITE(mycalendar,summary,organizer,start,end,busystate)=
"My event","mailto:jdoe@example.com",228383580,228383640,1)
```

Calendaring Dialplan Examples

Office hours

A common business PBX scenario is would be executing dialplan logic based on when the business is open and the phones staffed. If the business is closed for holidays, it is sometimes desirable to play a message to the caller stating why the business is closed.

The standard way to do this in asterisk has been doing a series of GotoIfTime statements or time-based include statements. Either way can be tedious and requires someone with access to edit asterisk config files.

With calendaring, the adminstrator only needs to set up a calendar that contains the various holidays or even recurring events specifying the office hours. A custom greeting filename could even be contained in the description field for playback. For example:

```
[incoming]
exten => 5555551212,1,Answer
exten =>
5555551212,n,GotoIf(${CALENDAR_BUSY(officehours)}?closed:attendant,s,1

exten =>
5555551212,n(closed),Set(id=${CALENDAR_QUERY(office,${EPOCH},${EPOCH})

exten =>
5555551212,n,Set(soundfile=${CALENDAR_QUERY_RESULT(${id},description)}
=> 5555551212,n,Playback($[${ISNULL(soundfile)} ? generic-closed ::
${soundfile}])
exten => 5555551212,n,Hangup
```

Meeting reminders

One useful application of Asterisk Calendaring is the ability to execute dialplan logic based on an event notification. Most calendaring technologies allow a user to set an alarm for an event. If these alarms are set on a calendar that Asterisk is monitoring and the calendar is set up for event notification via calendar.conf, then Asterisk will execute notify the specified channel at the time of the alarm. If an overrided notification time is set with the autoreminder setting, then the notification would happen at that time instead.

The following example demonstrates the set up for a simple event notification that plays back a generic message followed by the time of the upcoming meeting. calendar.conf.

```
[calendar_joe]
type = ical
url = https://example.com/home/jdoe/Calendar
user = jdoe
secret = mysecret
refresh = 15
timeframe = 600
autoreminder = 10
channel = SIP/joe
context = calendar_event_notify
extension = s
waittime = 30
```

extensions.conf :

```
[calendar_event_notify]
exten => s,1,Answer
exten => s,n,Playback(you-have-a-meeting-at)
exten => s,n,SayUnixTime(${CALENDAR_EVENT(start)})
exten => s,n,Hangup
```

Writing an event

Both CalDAV and Exchange calendar servers support creating new events. The following
example demonstrates writing a log of a call to a calendar.

```
[incoming]
exten => 6000,1,Set(start=${EPOCH})
exten => 6000,n,Dial(SIP/joe)
exten => h,1,Set(end=${EPOCH})
exten => h,n,Set(CALENDAR_WRITE(calendar_joe,summary,start,end)=Call
from ${CALLERID(all)},${start},${end})
```

Asterisk Channel Drivers

All about Asterisk and its Channel Drivers

Inter-Asterisk eXchange protocol, version 2 (IAX2)

Why IAX2?

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

- Interoperability with NAT/PAT/Masquerade firewalls
- IAX seamlessly interoperates through all sorts of NAT and PAT and other firewalls, including the ability to place and receive calls, and
  transfer calls to other stations.
- High performance, low overhead protocol
- When running on low-bandwidth connections, or when running large numbers of calls, optimized bandwidth utilization is imperative. IAX
  uses only 4 bytes of overhead
- Internationalization support
- IAX transmits language information, so that remote PBX content can be delivered in the native language of the calling party.
- Remote dialplan polling
- IAX allows a PBX or IP phone to poll the availability of a number from a remote server. This allows PBX dialplans to be centralized.
- Flexible authentication
- IAX supports cleartext, md5, and RSA authentication, providing flexible security models for outgoing calls and registration services.
- Multimedia protocol
- IAX supports the transmission of voice, video, images, text, HTML, DTMF, and URL's. Voice menus can be presented in both audibly and
  visually.
- Call statistic gathering

- IAX gathers statistics about network performance (including latency and jitter, as well as providing end-to-end latency measurement.
- Call parameter communication
- Caller*ID, requested extension, requested context, etc are all communicated through the call.
- Single socket design
- IAX's single socket design allows up to 32768 calls to be multiplexed.

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

Introduction to IAX2

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

IAX2 Configuration

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

IAX2 Jitterbuffer

The new jitterbuffer

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

PLC

The new jitterbuffer detects packet loss. PLC is done to try to recreate these lost packets in the codec decoding stage, as the encoded audio is translated to slinear. PLC is also used to mask jitterbuffer growth.
This facility is enabled by default in iLBC and speex, as it has no additional cost. This facility can be enabled in adpcm, alaw, g726, gsm, lpc10, and ulaw by setting genericplc = true in the [plc] section of codecs.conf.

Trunktimestamps

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

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

Communication with Asterisk v1.0.x systems

You can set up communication with v1.0.x systems with the new jitterbuffer, but you can't use trunks with trunktimestamps in this communication.
If you are connecting to an Asterisk server with earlier versions of the software (1.0.x), do not

enable both jitterbuffer and trunking for the involved peers/users in order to be able to communicate. Earlier systems will not support trunktimestamps.

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

Testing and monitoring

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

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

The stats shown are:

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

Reporting problems

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

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

- Bad timestamps: If whatever is generating timestamps to be sent to you generates nonsensical timestamps, it can confuse the jitterbuffer. In particular, discontinuities in timestamps will really upset it: Things like timestamps sequences which go 0, 20, 40, 60, 80, 34000, 34020, 34040, 34060... It's going to think you've got about 34 seconds of jitter in this case, etc.. The right solution to this is to find out what's causing the sender to send us such nonsense, and fix that. But we should also figure out how to make the receiver more robust in cases like this.
  chan_iax2 will actually help fix this a bit if it's more than 3 seconds or so, but at some point we should try to think of a better way to detect this kind of thing and resynchronize.

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

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

mISDN

Introduction to mISDN

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

mISDN Features

- NT and TE mode
- PP and PMP mode
- BRI and PRI (with BNE1 and BN2E1 Cards)
- Hardware bridging
- DTMF detection in HW+mISDNdsp
- Display messages on phones (on those that support it)
- app_SendText
- HOLD/RETRIEVE/TRANSFER on ISDN phones :)
- Allow/restrict user number presentation
- Volume control
- Crypting with mISDNdsp (Blowfish)
- Data (HDLC) callthrough
- Data calling (with app_ptyfork +pppd)
- Echo cancellation
- Call deflection
- Some others

mISDN Fast Installation Guide

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

You can download latest stable releases from http://www.misdn.org/downloads/

Just fetch the newest head of the GIT (mISDN project moved from CVS) In details this process described here: http://www.misdn.org/index.php/GIT
then compile and install both with:

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

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

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

Now you can compile chan_misdn, just by making Asterisk:

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

That's all!

Follow the instructions in the mISDN Package for how to load the Kernel Modules. Also install process described in http://www.misdn.org/index.php/Installing_mISDN

mISDN Pre-Requisites

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

You should use Kernels = 2.6.9

mISDN Configuration

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

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

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

misdn.conf: [general] subsection

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

misdn.conf: [default] subsection

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

misdn.conf: user/port subsections

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

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

Dial and Options String

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

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

The Optionsstring looks Like:

```
:<optchar><optarg>:<optchar><optarg>...
```

The ":" character is the delimiter. The available options are:

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

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

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

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

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

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

- Box 1:

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

- Box 2:

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

mISDN CLI Commands

At the Asterisk cli you can try to type in:

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

Now you should see the misdn cli commands:

- clean -> pid (cleans a broken call, use with care, leads often to a segmentation fault)
- send -> display (sends a Text Message to a Asterisk channel, this channel must be an misdn channel)
- set -> debug (sets debug level)
- show ->
  - config (shows the configuration options)
  - channels (shows the current active misdn channels)
  - channel (shows details about the given misdn channels)
  - stacks (shows the current ports, their protocols and states)
  - fullstacks (shows the current active and inactive misdn channels)
- restart -> port (restarts given port (L2 Restart)) - reload (reloads misdn.conf)

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

Then you use it like this:

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

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

mISDN Variables

mISDN Exports/Imports a few Variables:

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

mISDN Debugging and Bug Reports

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

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

mISDN Examples

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

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

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

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

mISDN Known Problems

- Q: I cannot hear any tone after a successful CONNECT to the other end.
- A: You forgot to load mISDNdsp, which is now needed by chan_misdn for switching and DTMF tone detection.

Local Channel

Introduction to Local Channels

In Asterisk, Local channels are a method used to treat an extension in the dialplan as if it were an external device. In essense, Asterisk will send the call back into the dialplan as the destination of the call, versus sending the call to a device.

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

Local Channel Examples

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

Trivial Local Channel Example

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Delay Dialing Devices Example

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

The dialplan for this would look something like the following:

```
[devices]
exten => 201,1,Verbose(2,Call desk phone and cellphone but with
delay)
exten =>
201,n,Dial(Local/deskphone-201@extensions&Local/cellphone-201@extensio

exten => 201,n,Voicemail(201@default,${IF($[${DIALSTATUS} =
BUSY]?b:u)})
exten => 201,n,Hangup()

[extensions]
; Dial the desk phone
exten => deskphone-201,1,Verbose(2,Dialing desk phone of extension
201)
exten => deskphone-201,n,Dial(SIP/0004f2040001) ; SIP device with
MAC address
                                          ; of 0004f2040001
; Dial the cellphone
exten => cellphone-201,1,Verbose(2,Dialing cellphone of extension
201)
exten => cellphone-201,n,Verbose(2,-- Waiting 6 seconds before
dialing)
exten => cellphone-201,n,Wait(6)
exten => cellphone-201,n,Dial(DAHDI/g0/14165551212)
```

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

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

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

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

Dialing Destinations with Different Information

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

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

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

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

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

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

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

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

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

Because each of the channels is independent from the other, you could perform any other call manipulation you need. Perhaps the 1-416-555-1212 number is a cell phone and you know you can only ring that device for 18 seconds before the voicemail would pick up. You could then limit the length of time the external number is dialed, but still allow the internal device to be dialed for a longer period of time.

Using Callfiles and Local Channels

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

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

Our callfile will simply look like the following:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

We answer the call.

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

We then start playing back the files.

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

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

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

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

Understanding when to use (slash)n

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

This following dialplan will not perform as expected.

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

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

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

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

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

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

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

...into the following:

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

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

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

1. SIP device PHONE_A calls Asterisk via a SIP INVITE
2. Asterisk accepts the INVITE and then starts processing dialplan logic in the [internal] context
3. Our dialplan calls Dial(Local/2@services) - notice no /n
4. The Local channel then executes dialplan at extension 2 within the [services] context
5. Extension 2 within [services] then performs Dial() to PHONE_B with the line: Dial(SIP/PHONE_B,,L(60000:45000:15000))
6. SIP/PHONE_B then answers the call
7. Even though the L option was given when dialing the SIP device, the L information is stored in the channel that is doing the Dial() which is the Local channel, and not the endpoint SIP channel.
8. The Local channel in the middle, containing the information for tracking the time allowance of the call, is then optimized out of the call path, losing all information about when to terminate the call.
9. SIP/PHONE_A and SIP/PHONE_B then continue talking indefinitely.

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

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

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

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

Local Channel Modifiers

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

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

- 'j' - Adding "/j" at the end of the string allows you to use the generic jitterbuffer on incoming calls going to Asterisk applications. For example, this would allow you to use a jitterbuffer for an incoming SIP call to Voicemail by putting a Local channel in the middle. The 'j' option must be used in conjunction with the 'n' option to make sure that the Local channel does not get optimized out of the call. This option is available starting in the Asterisk 1.6.0 branch.

- 'm' - Using the "/m" option will cause the Local channel to forward music on hold (MoH) start and stop requests. Normally the Local channel acts on them and it is started or stopped on the Local channel itself. This options allows those requests to be forwarded through the Local channel.
This option is available starting in the Asterisk 1.4 branch.

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

This option is available starting in the Asterisk 1.6.0 branch.

Mobile Channel

chan_mobile pages

Introduction to the Mobile Channel

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

Mobile Channel Features

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

Mobile Channel Requirements

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

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

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

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

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

Mobile Channel Concepts

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

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

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

Configuring chan_mobile

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

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

Using chan_mobile

chan_mobile.so must be loaded either by loading it using the Asterisk CLI, or by adding it to /etc/asterisk/modules.conf
Search for your bluetooth devices using the CLI command 'mobile search'. Be patient with this command as it will take 8 - 10 seconds to do the discovery. This requires a free adapter. Headsets will generally have to be put into 'pairing' mode before they will show up here. This will return something like the following :-

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

This is a list of all bluetooth devices seen and whether or not they are usable with chan_mobile. The Address field contains the 'bd address' of the device. This is like an ethernet mac address. The Name field is whatever is configured into the device as its name. The Usable field tells you

whether or not the device supports the Bluetooth Handsfree Profile or Headset profile. The Type field tells you whether the device is usable as a Phone line (FXO) or a headset (FXS) The Port field is the number to put in the configuration file.

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

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

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

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

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

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

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

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

```
; Calls via LGTU5500
exten => _9X.,1,Dial(Mobile/LGTU550/${EXTEN:1},45)
exten => _9X.,n,Hangup
```

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

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

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

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

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

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

Mobile Channel Dialplan Hints

chan_mobile supports 'device status' so you can do somthing like

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

MobileStatus Application

chan_mobile also registers an application named MobileStatus. You can use this in your Dialplan to determine the 'state' of a device.
For example, suppose you wanted to call dave's extension, but only if he was in the office. You could test to see if his mobile phone was attached to Asterisk, if it is dial his extension, otherwise dial his mobile phone.

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

MobileStatus sets the value of the given variable to :-

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

Mobile Channel DTMF Debouncing

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

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

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

Mobile Channel SMS Sending and Receiving

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

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

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

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

The above will just print the message on the console.

If you use res_jabber, you could do something like this :-

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

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

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

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

Mobile Channel Debugging

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

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

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

This will log a bunch of debug messages indicating what the phone is doing, importantly the

rfcomm conversation between Asterisk and the phone. This can be used to sort out what your phone is doing and make chan_mobile support it.

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

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

Asterisk Configuration

The top-level page for all things related to Asterisk configuration

General Configuration Information

The top-level page for general (typical) Asterisk configuration information.

Configuration Parser

Introduction

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

General syntax

Asterisk configuration files are defined as follows:

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

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

```
[section]
label1 = value1
label2 = value2
object => name
label3 = value3
label2 = value4
object2 => name2
```

In this syntax, we create objects with the settings defined above the object creation. Note that settings are inherited from the top, so in the example above object2 has inherited the setting for "label1" from the first object.
For template configurations, the syntax for defining a section is changed to:

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

The options field is used to define templates, refer to templates and hide templates. Any object can be used as a template.
No whitespace is allowed between the closing "]" and the parenthesis "(".

Comments

All lines that starts with semi-colon ";" is treated as comments and is not parsed.
The ";~~" is a marker for a multi-line comment. Everything after that marker will be treated as a comment until the end-marker~~";" is found. Parsing begins directly after the end-marker.

```
;This is a comment
label = value
;-- This is
a comment -;
;- Comment --; exten=> 1000,1,dial(SIP/lisa)
```

Including other files

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

```
#include myusers.conf
```

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

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

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

The exec directive is used like this:

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

Adding to an existing section

```
[section]
label = value

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

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

Defining a template-only section

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

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

Using templates (or other configuration sections)

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

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

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

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

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

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

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

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

Additional Examples
(in top-level sip.conf)

```
[defaults]
type=friend
nat=yes
qualify=on
dtmfmode=rfc2833
disallow=all
allow=alaw
#include accounts/*/sip.conf
```

(in accounts/customer1/sip.conf)

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

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

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

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

The asterisk.conf file

Asterisk Main Configuration File

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

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

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

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

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

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

; Where spool directories are located
; Voicemail, monitor, dictation and other apps will create files
here
; and outgoing call files (used with pbx_spool) must be placed here
```

```
astspooldir => /var/spool/asterisk

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

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

[options]
;Under "options" you can enter configuration options
;that you also can set with command line options
; Verbosity level for logging (-v) verbose = 0
; Debug: "No" or value (1-4)
debug = 3

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

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

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

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

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

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

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

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

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

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

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

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

; Language Options
documentation_language = en | es | ru

; These options have no command line equivalent

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

; Build transcode paths via SLINEAR
transcode_via_sln = yes | no

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

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

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

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

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

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

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

; Should language code be last component of sound file name or
first?
; when off, sound files are searched as <path>/<lang>/<file>
; when on, sound files are search as <lang>/<path>/<file>
; (only affects relative paths for sound files)
```

```
languageprefix = yes | no

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

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

[files]
; Changing the following lines may compromise your security
; Asterisk.ctl is the pipe that is used to connect the remote CLI
; (asterisk -r) to Asterisk. Changing these settings change the
; permissions and ownership of this file.
; The file is created when Asterisk starts, in the "astrundir"
above.
;astctlpermissions = 0660
```

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

CLI Prompt

Changing the CLI Prompt

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

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

- %d - Date (year-month-date)
- %s - Asterisk system name (from asterisk.conf)
- %h - Full hostname
- %H - Short hostname

- %t - Time
- %u - Username
- %g - Groupname
- %% - Percent sign

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

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

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

The Asterisk Dialplan

The Asterisk dialplan

The Asterisk dialplan is divided into contexts. A context is simply a group of extensions. For each "line" that should be able to be called, an extension must be added to a context. Then, you configure the calling "line" to have access to this context.

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

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

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

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

command in the CLI.

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

Example dialplan

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

Special extensions

There are some extensions with important meanings:

- s - What to do when an extension context is entered (unless overridden by the low level channel interface) This is used in macros, and some special cases. "s" is not a generic catch-all wildcard extension.
- i - What to do if an invalid extension is entered
- h - The hangup extension, executed at hangup
- t - What to do if nothing is entered in the requisite amount of time.
- T - This is the extension that is executed when the 'absolute' timeout is reached. See "core show function TIMEOUT" for more information on setting timeouts.
- e - This extension will substitute as a catchall for any of the 'i', 't', or 'T' extensions, if any of them do not exist and catching the error in a single routine is desired. The function EXCEPTION may be used to query the type of exception or the location where it occurred.

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

IP Quality of Service

Introduction

Asterisk supports different QoS settings at the application level for various protocols on both signaling and media. The Type of Service (TOS) byte can be set on outgoing IP packets for various protocols. The TOS byte is used by the network to provide some level of Quality of Service (QoS) even if the network is congested with other traffic.

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

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

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

Table 2.1: Channel Driver QoS Settings

	Signaling	Audio	Video	Text

chan_sip	+	+	+	+
chan_skinny	+	+	+	
chan_mgcp	+	+		
chan_unistm	+	+		
chan_h323		+		
chan_iax2	+			

Table 2.2: Other ToS Settings

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

IP TOS values

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

The tos parameters also take numeric values.*

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

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

802.1p CoS values

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

Recommended values

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

Table 2.3: Recommended QoS Settings

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

IAX2

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

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

SIP

In sip.conf, there are four parameters that control the TOS settings: "tos_sip", "tos_audio", "tos_video" and "tos_text". tos_sip controls what TOS SIP call signaling packets are set to. tos_audio, tos_video and tos_text control what TOS values are used for RTP audio, video, and text packets, respectively.

There are four parameters to control 802.1p CoS: "cos_sip", "cos_audio", "cos_video" and "cos_text". The behavior of these parameters is the same as for the SIP TOS settings described above.

Other RTP channels

chan_mgcp, chan_h323, chan_skinny and chan_unistim also support TOS and CoS via setting tos and cos parameters in their corresponding configuration files. Naming style and behavior are the same as for chan_sip.

Reference

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

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

IANA Assignments, DSCP registry Differentiated Services Field Codepoints
http://www.iana.org/assignments/dscp-registry
To get the most out of setting the TOS on packets generated by Asterisk, you will need to ensure that your network handles packets with a TOS properly. For Cisco devices, see the previously mentioned "Enterprise QoS Solution Reference Network Design Guide". For Linux systems see the "Linux Advanced Routing & Traffic Control HOWTO" at http://www.lartc.org/.
For more information on Quality of Service for VoIP networks see the "Enterprise QoS Solution Reference Network Design Guide" version 3.3 from Cisco at:
http://www.cisco.com/application/pdf/en/us/guest/netsol/ns432/c649/ccmigration_09186a008049b

MP3 Support

MP3 Music On Hold

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

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

ICES

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

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

Database Support Configuration

Top-level page for information about Database support.

Realtime Database Configuration

Introduction

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

The benefits of this architecture are many, both from a code management standpoint and from an installation perspective.
The ARA is designed to be independent of storage. Currently, most drivers are based on SQL, but the architecture should be able to handle other storage methods in the future, like LDAP.

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

1. ODBC: Support for UnixODBC, integrated into Asterisk The UnixODBC subsystem supports many different databases, please check www.unixodbc.org for more information.
2. MySQL: Native support for MySQL, integrated into Asterisk
3. PostgreSQL: Native support for Postgres, integrated into Asterisk

Two modes: Static and Realtime

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

The ARA static mode is used to load configuration files. For the Asterisk modules that read configurations, there's no difference between a static file in the file system, like extensions.conf, and a configuration loaded from a database.
You just have to always make sure the var_metric values are properly set and ordered as you expect in your database server if you're using the static mode with ARA (either sequentially or with the same var_metric value for everybody).

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

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

Realtime SIP friends

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

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

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

Realtime H.323 friends

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

New function in the dial plan: The Realtime Switch

The realtime switch is more than a port of functionality in v1.0 to the new architecture, this is a new feature of Asterisk based on the ARA. The realtime switch lets your Asterisk server do database lookups of extensions in realtime from your dial plan. You can have many Asterisk servers sharing a dynamically updated dial plan in real time with this solution.
Note that this switch does NOT support Caller ID matching, only extension name or pattern matching.

Capabilities

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

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

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

Configuration in extconfig.conf

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

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

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

The options following the realtime driver identified depends on the driver.

Defined well-known family names are:

- sippeers, sipusers - SIP peers and users
- sipregs - SIP registrations
- iaxpeers, iaxusers - IAX2 peers and users
- voicemail - Voicemail accounts
- extensions - Realtime extensions (switch)
- meetme - MeetMe conference rooms
- queues - Queues
- queue_members - Queue members
- musiconhold - Music On Hold classes
- queue_log - Queue logging

Voicemail storage with the support of ODBC described in ODBC Voicemail Storage.

Limitations

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

FreeTDS supported with connection pooling

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

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

```
Threading = 2
```

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

FreeTDS

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

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

SIP Realtime, MySQL table structure

Here is the table structure used by MySQL for Realtime SIP friends

```
#
# Table structure for table `sipfriends`
#

CREATE TABLE IF NOT EXISTS `sipfriends` (
        `id` int(11) NOT NULL AUTO_INCREMENT,
        `name` varchar(10) NOT NULL,
        `ipaddr` varchar(15) DEFAULT NULL,
        `port` int(5) DEFAULT NULL,
        `regseconds` int(11) DEFAULT NULL,
        `defaultuser` varchar(10) DEFAULT NULL,
        `fullcontact` varchar(35) DEFAULT NULL,
        `regserver` varchar(20) DEFAULT NULL,
        `useragent` varchar(20) DEFAULT NULL,
        `lastms` int(11) DEFAULT NULL,
        `host` varchar(40) DEFAULT NULL,
        `type` enum('friend','user','peer') DEFAULT NULL,
        `context` varchar(40) DEFAULT NULL,
        `permit` varchar(40) DEFAULT NULL,
        `deny` varchar(40) DEFAULT NULL,
        `secret` varchar(40) DEFAULT NULL,
        `md5secret` varchar(40) DEFAULT NULL,
        `remotesecret` varchar(40) DEFAULT NULL,
        `transport` enum('udp','tcp','udp,tcp','tcp,udp') DEFAULT
NULL,
        `dtmfmode` enum('rfc2833','info','shortinfo','inband','auto')
DEFAULT NULL,
        `directmedia` enum('yes','no','nonat','update') DEFAULT NULL,
        `nat` enum('yes','no','never','route') DEFAULT NULL,
        `callgroup` varchar(40) DEFAULT NULL,
        `pickupgroup` varchar(40) DEFAULT NULL,
        `language` varchar(40) DEFAULT NULL,
        `allow` varchar(40) DEFAULT NULL,
        `disallow` varchar(40) DEFAULT NULL,
        `insecure` varchar(40) DEFAULT NULL,
        `trustrpid` enum('yes','no') DEFAULT NULL,
        `progressinband` enum('yes','no','never') DEFAULT NULL,
        `promiscredir` enum('yes','no') DEFAULT NULL,
        `useclientcode` enum('yes','no') DEFAULT NULL,
        `accountcode` varchar(40) DEFAULT NULL,
        `setvar` varchar(40) DEFAULT NULL,
```

```
      `callerid` varchar(40) DEFAULT NULL,
      `amaflags` varchar(40) DEFAULT NULL,
      `callcounter` enum('yes','no') DEFAULT NULL,
      `busylevel` int(11) DEFAULT NULL,
      `allowoverlap` enum('yes','no') DEFAULT NULL,
      `allowsubscribe` enum('yes','no') DEFAULT NULL,
      `videosupport` enum('yes','no') DEFAULT NULL,
      `maxcallbitrate` int(11) DEFAULT NULL,
      `rfc2833compensate` enum('yes','no') DEFAULT NULL,
      `mailbox` varchar(40) DEFAULT NULL,
      `session-timers` enum('accept','refuse','originate') DEFAULT
NULL,
      `session-expires` int(11) DEFAULT NULL,
      `session-minse` int(11) DEFAULT NULL,
      `session-refresher` enum('uac','uas') DEFAULT NULL,
      `t38pt_usertpsource` varchar(40) DEFAULT NULL,
      `regexten` varchar(40) DEFAULT NULL,
      `fromdomain` varchar(40) DEFAULT NULL,
      `fromuser` varchar(40) DEFAULT NULL,
      `qualify` varchar(40) DEFAULT NULL,
      `defaultip` varchar(40) DEFAULT NULL,
      `rtptimeout` int(11) DEFAULT NULL,
      `rtpholdtimeout` int(11) DEFAULT NULL,
      `sendrpid` enum('yes','no') DEFAULT NULL,
      `outboundproxy` varchar(40) DEFAULT NULL,
      `callbackextension` varchar(40) DEFAULT NULL,
      `registertrying` enum('yes','no') DEFAULT NULL,
      `timert1` int(11) DEFAULT NULL,
      `timerb` int(11) DEFAULT NULL,
      `qualifyfreq` int(11) DEFAULT NULL,
      `constantssrc` enum('yes','no') DEFAULT NULL,
      `contactpermit` varchar(40) DEFAULT NULL,
      `contactdeny` varchar(40) DEFAULT NULL,
      `usereqphone` enum('yes','no') DEFAULT NULL,
      `textsupport` enum('yes','no') DEFAULT NULL,
      `faxdetect` enum('yes','no') DEFAULT NULL,
      `buggymwi` enum('yes','no') DEFAULT NULL,
      `auth` varchar(40) DEFAULT NULL,
      `fullname` varchar(40) DEFAULT NULL,
      `trunkname` varchar(40) DEFAULT NULL,
      `cid_number` varchar(40) DEFAULT NULL,
      `callingpres`
enum('allowed_not_screened','allowed_passed_screen','allowed_failed_sc
DEFAULT NULL,
      `mohinterpret` varchar(40) DEFAULT NULL,
      `mohsuggest` varchar(40) DEFAULT NULL,
```

```
`parkinglot` varchar(40) DEFAULT NULL,
`hasvoicemail` enum('yes','no') DEFAULT NULL,
`subscribemwi` enum('yes','no') DEFAULT NULL,
`vmexten` varchar(40) DEFAULT NULL,
`autoframing` enum('yes','no') DEFAULT NULL,
`rtpkeepalive` int(11) DEFAULT NULL,
`call-limit` int(11) DEFAULT NULL,
`g726nonstandard` enum('yes','no') DEFAULT NULL,
`ignoresdpversion` enum('yes','no') DEFAULT NULL,
`allowtransfer` enum('yes','no') DEFAULT NULL,
`dynamic` enum('yes','no') DEFAULT NULL,
PRIMARY KEY (`id`),
UNIQUE KEY `name` (`name`),
KEY `ipaddr` (`ipaddr`,`port`),
```

```
        KEY `host` (`host`,`port`)
) ENGINE=MyISAM;
```

Privacy Configuration

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

FTC Don't Call List

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

Fighting Autodialers

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

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

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

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

Fighting Empty Caller ID

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

Null callerid's are a fact of life, and could be a friend with an unlisted number, or some charity looking for a handout. The PrivacyManager application can help here. It will ask the caller to enter a 10-digit phone number. They get 3 tries(configurable), and this is configurable, with control being passed to next priority where you can check the channelvariable PRIVACYMGRSTATUS. If the callerid was valid this variable will have the value SUCCESS, otherwise it will have the value FAILED.

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

PrivacyManager can be told about a context where you can have patterns that describe valid

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

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

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

Using Welcome Menus for Privacy

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

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

Example usage of Zapateller and PrivacyManager

```
[homeline]
exten => s,1,Answer
exten => s,2,SetVar,repeatcount=0
exten => s,3,Zapateller,nocallerid
exten => s,4,PrivacyManager
;; do this if they don't enter a number to Privacy Manager
exten => s,5,GotoIf($[ "${PRIVACYMGRSTATUS}" = "FAILED" ]?s,105)
exten => s,6,GotoIf($[ "${CALLERID(num)}" = "7773334444" &
"${CALLERID(name)}" : "Privacy Manager" ]?callerid-liar,s,1:s,7)
exten => s,7,Dial(SIP/yourphone)
exten => s,105,Background(tt-allbusy)
exten => s,106,Background(tt-somethingwrong)
exten => s,107,Background(tt-monkeysintro)
exten => s,108,Background(tt-monkeys)
exten => s,109,Background(tt-weasels)
exten => s,110,Hangup
```

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

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

In nearly all cases now, the telemarketers/charity-seekers that usually get thru to my main intro,

hang up. I guess they can see it's pointless, or the average telemarketer/charity-seeker is instructed not to enter options when encountering such systems. Don't know.

Making life difficult for telemarketers

I have developed an elaborate script to torture Telemarketers, and entertain friends.

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

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

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

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

Using Call Screening

Another option is to use call screening in the Dial command. It has two main privacy modes, one that remembers the CID of the caller, and how the callee wants the call handled, and the other, which does not have a "memory".
Turning on these modes in the dial command results in this sequence of events, when someone calls you at an extension:

The caller calls the Asterisk system, and at some point, selects an option or enters an extension number that would dial your extension.

Before ringing your extension, the caller is asked to supply an introduction. The application asks them: "After the tone, say your name". They are allowed 4 seconds of introduction.

After that, they are told "Hang on, we will attempt to connect you to your party. Depending on your dial options, they will hear ringing indications, or get music on hold. I suggest music on hold.

Your extension is then dialed. When (and if) you pick up, you are told that a caller presenting themselves as their recorded intro is played is calling, and you have options, like being connected, sending them to voicemail, torture, etc.
You make your selection, and the call is handled as you chose.

There are some variations, and these will be explained in due course.

To use these options, set your Dial to something like:

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

or:

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

or:

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

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

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

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

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

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

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

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

Call Screening Options

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

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

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

Screening Calls with Recorded Introductions

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

Introductions

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

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

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

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

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

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

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

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

```
[home-introduction]
exten => s,1,Background(intro-options) ;; Script:
;; To hear your Introduction, dial 1.
;; to record a new introduction, dial 2.
;; to return to the main menu, dial 3.
;; to hear what this is all about, dial 4.
exten => 1,1,Playback,priv-callerintros/${CALLERID(num)}
```

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

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

```
;; "John, from the Park County Morgue. You stab 'em, we slab 'em!".
;; Just one caution: the kids will hear what you record every time
;; you call. So watch your language!
;; I will begin recording after the tone.
;; When you are done, hit the # key. Gather your thoughts and get
;; ready. Remember, the # key will end the recording, and play back
;; your intro. Good Luck, and Thank you!"
exten => 1,2,Goto(2,1)
exten => 2,1,Background(intro-start)
;; OK, here we go! After the beep, please give your introduction.
exten => 2,2,Background(beep)
exten => 2,3,Record(priv-callerintros/${CALLERID(num)}:gsm,4)
exten => 2,4,Background(priv-callerintros/${CALLERID(num)})
exten => 2,5,Goto(home-introduction,s,1)
exten => t,1,Goto(s,1)
```

```
exten => i,1,Background(invalid)
exten => i,2,Goto(s,1)
exten => o,1,Goto(s,1)
```

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

Asterisk Extension Language (AEL)

Top-level page for all things AEL

Introduction to AEL

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

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

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

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

1. The first and most obvious is the AEL syntax itself. A BNF is provided near the end of this document.
2. The second syntax is the Expression Syntax, which is normally handled by Asterisk extension engine, as expressions enclosed in $[...]. The right hand side of assignments are wrapped in $[...] by AEL, and so are the if and while expressions, among others.
3. The third syntax is the Variable Reference Syntax, the stuff enclosed in ${..} curly braces. It's a bit more involved than just putting a variable name in there. You can include one of dozens of 'functions', and their arguments, and there are even some string manipulation notation in there.
4. The last syntax that underlies AEL, and is not used directly in AEL, is the Extension Language Syntax. The extension language is what you see in extensions.conf, and AEL compiles the higher level AEL language into extensions and priorities, and passes them via function calls into Asterisk.
   Embedded in this language is the Application/AGI commands, of which one application call per step, or priority can be made. You can think of this as a "macro assembler" language, that AEL will compile into.

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

AEL and Asterisk in a Nutshell

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

AEL about Contexts

Contexts are a grouping of extensions.

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

AEL about Extensions and priorities

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

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

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

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

AEL about Macros

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

AEL about Applications

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

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

Getting Started with AEL

The AEL parser (res_ael.so) is completely separate from the module that parses extensions.conf (pbx_config.so). To use AEL, the only thing that has to be done is the module res_ael.so must be loaded by Asterisk. This will be done automatically if using 'autoload=yes' in /etc/asterisk/modules.conf. When the module is loaded, it will look for 'extensions.ael' in

/etc/asterisk/. extensions.conf and extensions.ael can be used in conjunction with each other if that is what is desired. Some users may want to keep extensions.conf for the features that are configured in the 'general' section of extensions.conf.

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

```
*CLI ael reload
```

AEL Debugging

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

- Enable AEL contexts debug

```
*CLI> ael debug contexts
```

- Enable AEL macros debug

```
*CLI> ael debug macros
```

- Enable AEL read debug

```
*CLI> ael debug read
```

- Enable AEL tokens debug

```
*CLI> ael debug tokens
```

- Disable AEL debug messages

```
*CLI> ael no debug
```

> ✅ If things are going wrong in your dialplan, you can use the following facilities to debug your file:
>
> 1. The messages log in /var/log/asterisk. (from the checks done at load time).
> 2. The "show dialplan" command in asterisk
> 3. The standalone executable, "aelparse" built in the utils/ dir in the source.

About "aelparse"

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

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

aelparse has two optional arguments:

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

General Notes about AEL Syntax

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

You can just as easily say,

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

as you can say:

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

or:

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

or:

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

AEL Keywords

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

The following are keywords in the AEL language:

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

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

AEL Procedural Interface and Internals

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

This tree is then handed to the semantic check routine.

Then the tree is handed to the compiler.

After that, it is freed from memory.

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

AEL version 2 BNF

(hopefully, something close to bnf).

First, some basic objects

```
------------------------
<word> a lexical token consisting of characters matching this
pattern:
[-a-zA-Z0-9"_/.\<\>\*\+!$#\[\]][-a-zA-Z0-9"_/.!\*\+\<\>\{\}$#\[\]]*
    <word3-list> a concatenation of up to 3 <word>s.
    <collected-word> all characters encountered until the character
that follows the <collected-word> in the grammar.
    -------------------------
    <file> :== <objects>

    <objects> :== <object>
```

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

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

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

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

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


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

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

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

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

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

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

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

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

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

<word3-list> :== <word>
         | <word> <word>
         | <word> <word> <word>
<switch_head> :== 'switch' '(' <collected-word> ')' '{'

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

```
            | <ifTime_head> <statement> 'else' <statement>
            | ';'
<target> :== <word>
            | <word> '|' <word>
            | <word> '|' <word> '|' <word>
            | 'default' '|' <word> '|' <word>
            | <word> ',' <word>
            | <word> ',' <word> ',' <word>
            | 'default' ',' <word> ',' <word>

<jumptarget> :== <word>
                | <word> ',' <word>
                | <word> ',' <word> '@' <word>
                | <word> '@' <word>
                | <word> ',' <word> '@' 'default'
                | <word> '@' 'default'
<macro_call> :== <word> '(' <eval_arglist> ')'
        | <word> '(' ')'

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

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

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

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

   <macro_statement> :== <statement>
            | 'catch' <word> '{' <statements> '}'
   <switches> :== 'switches' '{' <switchlist> '}'
            | 'switches' '{' '}'
   <eswitches> :== 'eswitches' '{' <switchlist> '}'
            | 'eswitches' '{' '}'
```

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

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

AEL Example Usages

Example usages of AEL

AEL Comments

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

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

AEL Context

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

```
context default {
}
```

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

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

AEL Extensions

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

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

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

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

```
context default {
    regexten _5XXX => NoOp(it's a pattern!);
}

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

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

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

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

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

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

AEL Includes

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

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

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

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

AEL including other files

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

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

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

AEL Dialplan Switches

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

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

AEL Ignorepat

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

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

AEL Variables

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

Global variables are set in their own block.

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

Variables can be set within extensions as well.

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

NOTE: AEL wraps the right hand side of an assignment with $[ ] to allow expressions to be used

If this is unwanted, you can protect the right hand side from being wrapped by using the Set() application. Read the README.variables about the requirements and behavior of $[ ] expressions.
NOTE: These things are wrapped up in a $[ ] expression: The while() test; the if() test; the middle expression in the for(x; y; z) statement (the y expression); Assignments - the right hand side, so a = b - Set(a=$[b])

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

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

You can declare variables in Macros, as so:

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

AEL Local Variables

In 1.2, and 1.4, ALL VARIABLES are CHANNEL variables, including the function arguments and associated ARG1, ARG2, etc variables. Sorry.
In trunk (1.6 and higher), we have made all arguments local variables to a macro call. They will not affect channel variables of the same name. This includes the ARG1, ARG2, etc variables.

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

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

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

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

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

AEL Conditionals

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

```
context conditional {
    _8XXX => {
        Dial(SIP/${EXTEN});
        if ("${DIALSTATUS}" = "BUSY")
        {
            NoOp(yessir);
            Voicemail(${EXTEN},b);
        }
        else
            Voicemail(${EXTEN},u);
        ifTime (14:00-25:00,sat-sun,,)
            Voicemail(${EXTEN},b);
        else
        {
            Voicemail(${EXTEN},u);
            NoOp(hi, there!);
        }
        random(51) NoOp(This should appear 51% of the time);
        random( 60 )
        {
                NoOp( This should appear 60% of the time );
        }
        else
        {
                random(75)
                {
                    NoOp( This should appear 30% of the time! );
                }
                else
                {
                    NoOp( This should appear 10% of the time! );
```

```
                }
        }
    }
    _777X => {
        switch (${EXTEN}) {
            case 7771:
                NoOp(You called 7771!);
                break;
            case 7772:
                NoOp(You called 7772!);
                break;
            case 7773:
                NoOp(You called 7773!);
                // fall thru-
            pattern 777[4-9]:
                NoOp(You called 777 something!);
            default: NoOp(In the default clause!);
```

```
                }
        }
}
```

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

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

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

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

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

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

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

AEL goto, jump, and labels

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

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

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

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

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

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

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

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

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

AEL Macros

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

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

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

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

AEL Loops

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

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

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

AEL Break, Continue, and Return

Three keywords:

1. break
2. continue
3. return

are included in the syntax to provide flow of control to loops, and switches.

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

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

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

AEL Examples

```
context demo {
    s => {
        Wait(1);
        Answer();
        TIMEOUT(digit)=5;
        TIMEOUT(response)=10;
    restart:
        Background(demo-congrats);
    instructions:
        for (x=0; ${x} < 3; x=${x} + 1) {
            Background(demo-instruct);
            WaitExten();
        }
    }
    2 => {
        Background(demo-moreinfo);
        goto s,instructions;
    }
    3 => {
        LANGUAGE()=fr;
        goto s,restart;
    }
    500 => {
        Playback(demo-abouttotry);
        Dial(IAX2/guest@misery.digium.com);
        Playback(demo-nogo);
        goto s,instructions;
    }
    600 => {
        Playback(demo-echotest);
        Echo();
        Playback(demo-echodone);
        goto s,instructions;
    }
    # => {
        hangup:
            Playback(demo-thanks);
            Hangup();
    }
    t => goto #,hangup;
    i => Playback(invalid);
}
```

AEL Semantic Checks

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

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

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

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

Differences with the original version of AEL

1. The $[...] expressions have been enhanced to include the ==, , and && operators. These operators are exactly equivalent to the =, , and & operators, respectively. Why? So the C, Java, C++ hackers feel at home here.
2. It is more free-form. The newline character means very little, and is pulled out of the white-space only for line numbers in error messages.
3. It generates more error messages - by this I mean that any difference between the input and the grammar are reported, by file, line number, and column.
4. It checks the contents of $[ ] expressions (or what will end up being $[ ] expressions!) for syntax errors. It also does matching paren/bracket counts.
5. It runs several semantic checks after the parsing is over, but before the compiling begins, see the list above.
6. It handles #include "filepath" directives. - ALMOST anywhere, in fact. You could easily include a file in a context, in an extension, or at the root level. Files can be included in files that are included in files, down to 50 levels of hierarchy...
7. Local Goto's inside Switch statements automatically have the extension of the location of the switch statement appended to them.
8. A pretty printer function is available within pbx_ael.so.
9. In the utils directory, two standalone programs are supplied for debugging AEL files. One is called "aelparse", and it reads in the /etc/asterisk/extensions.ael file, and shows the results of syntax and semantic checking on stdout, and also shows the results of compilation to stdout. The other is "aelparse1", which uses the original ael compiler to do the same work, reading in "/etc/asterisk/extensions.ael", using the original 'pbx_ael.so' instead.
10. AEL supports the "jump" statement, and the "pattern" statement in switch constructs. Hopefully these will be documented in the AEL README.
11. Added the "return" keyword, which will jump to the end of an extension/Macro.
12. Added the ifTime (time rangedays of weekdays of monthmonths) [else] construct, which executes much like an if () statement, but the decision is based on the current time, and the time spec provided in the ifTime. See the example above. (Note: all the other time-dependent Applications can be used via ifTime)
13. Added the optional time spec to the contexts in the includes construct. See examples above.
14. You don't have to wrap a single "true" statement in curly braces, as in the original AEL. An "else" is attached to the closest if. As usual, be careful about nested if statements! When in doubt, use curlies!
15. Added the syntax [regexten] [hint(channel)] to precede an extension declaration. See examples above, under "Extension". The regexten keyword will cause the priorities in the extension to begin with 2 instead of 1. The hint keyword will cause its arguments to be inserted in the extension under the hint priority. They are both optional, of course, but the order is fixed at the moment- the regexten must come before the hint, if they are both present.
16. Empty case/default/pattern statements will "fall thru" as expected. (0.6)
17. A trailing label in an extension, will automatically have a NoOp() added, to make sure the label exists in the extension on Asterisk. (0.6)
18. (0.9) the semicolon is no longer required after a closing brace! (i.e. "];" === "}". You can have them there if you like, but they are not

necessary. Someday they may be rejected as a syntax error, maybe.

19. (0.9) the // comments are not recognized and removed in the spots where expressions are gathered, nor in application call arguments. You may have to move a comment if you get errors in existing files.
20. (0.10) the random statement has been added. Syntax: random (expr) lucky-statement [else unlucky-statement]. The probability of the lucky-statement getting executed is expr, which should evaluate to an integer between 0 and 100. If the lucky-statement isn't so lucky this time around, then the unlucky-statement gets executed, if it is present.

AEL Hints and Bugs

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

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

The Full Power of AEL

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

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

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

Asterisk Manager Interface (AMI)

What is the Asterisk Manager Interface, or AMI? Read on...

The Asterisk Manager TCP IP API

The manager is a client/server model over TCP. With the manager interface, you'll be able to control the PBX, originate calls, check mailbox status, monitor channels and queues as well as execute Asterisk commands.

AMI is the standard management interface into your Asterisk server. You configure AMI in manager.conf. By default, AMI is available on TCP port 5038 if you enable it in manager.conf.

AMI receive commands, called "actions". These generate a "response" from Asterisk. Asterisk will also send "Events" containing various information messages about changes within Asterisk. Some actions generate an initial response and data in the form list of events. This format is

created to make sure that extensive reports do not block the manager interface fully.

Management users are configured in the configuration file manager.conf and are given permissions for read and write, where write represents their ability to perform this class of "action", and read represents their ability to receive this class of "event".
If you develop AMI applications, treat the headers in Actions, Events and Responses as local to that particular message. There is no cross-message standardization of headers.

If you develop applications, please try to reuse existing manager headers and their interpretation. If you are unsure, discuss on the asterisk-dev mailing list.

Manager subscribes to extension status reports from all channels, to be able to generate events when an extension or device changes state. The level of details in these events may depend on the channel and device configuration. Please check each channel configuration file for more information. (in sip.conf, check the section on subscriptions and call limits)

AMI Command Syntax

Management communication consists of tags of the form "header: value", terminated with an empty newline (\r\n) in the style of SMTP, HTTP, and other headers.

The first tag MUST be one of the following:

- Action: An action requested by the CLIENT to the Asterisk SERVER. Only one "Action" may be outstanding at any time.
- Response: A response to an action from the Asterisk SERVER to the CLIENT.
- Event: An event reported by the Asterisk SERVER to the CLIENT

AMI Manager Commands

To see all of the available manager commands, use the "manager show commands" CLI command.

You can get more information about a manager command with the "manager show command command" CLI command in Asterisk.

AMI Examples

- Login - Log a user into the manager interface.

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

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

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

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

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

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

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

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

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

Ensuring all modules are loaded with AMI

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

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

Device Status Reports with AMI

blank

Some Standard AMI Headers

- Account: – Account Code (Status)
- AccountCode: – Account Code (cdr_manager)
- ACL: <Y | N> – Does ACL exist for object ?
- Action: <action> – Request or notification of a particular action
- Address-IP: – IPaddress
- Address-Port: – IP port number
- Agent: <string> – Agent name
- AMAflags: – AMA flag (cdr_manager, sippeers)
- AnswerTime: – Time of answer (cdr_manager)
- Append: <bool> – CDR userfield Append flag
- Application: – Application to use
- Async: – Whether or not to use fast setup
- AuthType: – Authentication type (for login or challenge) "md5"
- BillableSeconds: – Billable seconds for call (cdr_manager)
- CallerID: – Caller id (name and number in Originate & cdr_manager)
- CallerID: – CallerID number Number or "<unknown>" or "unknown" (should change to "<unknown>" in app_queue)
- CallerID1: – Channel 1 CallerID (Link event)
- CallerID2: – Channel 2 CallerID (Link event)
- CallerIDName: – CallerID name Name or "<unknown>" or "unknown" (should change to "<unknown>" in app_queue)
- Callgroup: – Call group for peer/user

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

- RegExpiry: – SIP registry expiry
- Reason: – Originate reason code
- Seconds: – Seconds (Status)
- Secret: <password> – Authentication secret (for login)
- SecretExist: <Y | N> – Whether secret exists
- Shutdown: – "Uncleanly", "Cleanly"
- SIP-AuthInsecure:
- SIP-FromDomain: – Peer FromDomain
- SIP-FromUser: – Peer FromUser
- SIP-NatSupport:
- SIPLastMsg:
- Source: – Source of call (dial event, cdr_manager)
- SrcUniqueID: – UniqueID of source (dial event)
- StartTime: – Start time of call (cdr_manager)
- State: – Channel state
- State: <1 | 0> – Mute flag
- Status: – Registration status (Registry events SIP)
- Status: – Extension status (Extensionstate)
- Status: – Peer status (if monitored) ** Will change name ** "unknown", "lagged", "ok"
- Status: <num> – Queue Status
- Status: – DND status (DNDState)
- Time: <sec> – Roundtrip time (latency)
- Timeout: – Parking timeout time
- Timeout: – Timeout for call setup (Originate)
- Timeout: <seconds> – Timeout for call
- Uniqueid: – Channel Unique ID
- Uniqueid1: – Channel 1 Unique ID (Link event)
- Uniqueid2: – Channel 2 Unique ID (Link event)
- User: – Username (SIP registry)
- UserField: – CDR userfield (cdr_manager)
- Val: – Value to set/read in ASTdb
- Variable: – Variable AND value to set (multiple separated with | in Originate)
- Variable: <name> – For channel variables
- Value: <value> – Value to set
- VoiceMailbox: – VM Mailbox in SIPpeers
- Waiting: – Count of mailbox messages (mailboxstatus)

⚠ Please try to re-use existing headers to simplify manager message parsing in clients.*

Read the CODING-GUIDELINES if you develop new manager commands or events.

Asynchronous Javascript Asterisk Manger (AJAM)

AJAM is a new technology which allows web browsers or other HTTP enabled applications and web pages to directly access the Asterisk Manger Interface (AMI) via HTTP. Setting up your server to process AJAM involves a few steps:

Setting up the Asterisk HTTP server

1. Uncomment the line "enabled=yes" in /etc/asterisk/http.conf to enable Asterisk's builtin micro HTTP server.
2. If you want Asterisk to actually deliver simple HTML pages, CSS, javascript, etc. you should uncomment "enablestatic=yes"
3. Adjust your "bindaddr" and "bindport" settings as appropriate for your desired accessibility
4. Adjust your "prefix" if appropriate, which must be the beginning of any URI on the server to match. The default is "asterisk" and the rest of these instructions assume that value.

Allow Manager Access via HTTP

1. Make sure you have both "enabled = yes" and "webenabled = yes" setup in /etc/asterisk/manager.conf
2. You may also use "httptimeout" to set a default timeout for HTTP connections.
3. Make sure you have a manager username/secret

Once those configurations are complete you can reload or restart Asterisk and you should be able to point your web browser to specific URI's which will allow you to access various web

functions. A complete list can be found by typing "http show status" at the Asterisk CLI.
examples:

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

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

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

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

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

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

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

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

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

> ⚠️ For the demo, there is no need for **any external web server.**

Integration with other web servers

Asterisk's micro HTTP server is **not designed to replace a general purpose web server** and it is intentionally created to provide only the minimal interfaces required. Even without the addition of an external web server, one can use Asterisk's interfaces to implement screen pops and similar tools pulling data from other web servers using iframes, div's etc. If you want to integrate CGI's, databases, PHP, etc. you will likely need to use a more traditional web server like Apache and link in your Asterisk micro HTTP server with something like this:

ProxyPass /asterisk http://localhost:8088/asterisk

Asterisk Queues

Pardon, but the dialplan in this tutorial will be expressed in AEL, the new Asterisk Extension Language. If you are not used to its syntax, we hope you will find it to some degree intuitive. If not, there are documents explaining its syntax and constructs.

Configuring Call Queues

Top-level for configuring call queues

Using queues.conf

First of all, set up call queues in queue.conf
Here is an example:

queues.conf

```
; Cool Digium Queues
[general]
persistentmembers = yes

; General sales queue
[sales-general]
music=default
context=sales
strategy=ringall
joinempty=strict
leavewhenempty=strict

; Customer service queue
[customerservice]
music=default
context=customerservice
strategy=ringall
joinempty=strict
leavewhenempty=strict

; Support dispatch queue
[dispatch]
music=default
context=dispatch
strategy=ringall
joinempty=strict
leavewhenempty=strict
```

In the above, we have defined 3 separate calling queues: sales-general, customerservice, and dispatch.

Please note that the sales-general queue specifies a context of "sales", and that customerservice specifies the context of "customerservice", and the dispatch queue specifies the context "dispatch". These three contexts must be defined somewhere in your dialplan. We will show them after the main menu below.

In the [general] section, specifying the persistentmembers=yes, will cause the agent lists to be stored in astdb, and recalled on startup.

The strategy=ringall will cause all agents to be dialed together, the first to answer is then assigned the incoming call.

"joinempty" set to "strict" will keep incoming callers from being placed in queues where there are no agents to take calls. The Queue() application will return, and the dial plan can determine what to do next.

If there are calls queued, and the last agent logs out, the remaining incoming callers will immediately be removed from the queue, and the Queue() call will return, IF the "leavewhenempty" is set to "strict".

Routing Incoming Calls to Queues

Then in extensions.ael, you can do these things:

The Main Menu
At Digium, incoming callers are sent to the "mainmenu" context, where they are greeted, and directed to the numbers they choose...

```
context mainmenu {
    includes {
        digium;
        queues-loginout;
    }
    0 => goto dispatch,s,1;
    2 => goto sales,s,1;
    3 => goto customerservice,s,1;
    4 => goto dispatch,s,1;
    s => {
        Ringing();
        Wait(1);
        Set(attempts=0);
        Answer();
        Wait(1);
        Background(digium/ThankYouForCallingDigium);
        Background(digium/YourOpenSourceTelecommunicationsSupplier);

        WaitExten(0.3);
    repeat:
        Set(attempts=$[${attempts} + 1]);

 Background(digium/IfYouKnowYourPartysExtensionYouMayDialItAtAnyTime);

        WaitExten(0.1);
        Background(digium/Otherwise);
        WaitExten(0.1);
```

```
        Background(digium/ForSalesPleasePress2);
        WaitExten(0.2);
        Background(digium/ForCustomerServicePleasePress3);
        WaitExten(0.2);
        Background(digium/ForAllOtherDepartmentsPleasePress4);
        WaitExten(0.2);

Background(digium/ToSpeakWithAnOperatorPleasePress0AtAnyTime);
        if( ${attempts} < 2 ) {
            WaitExten(0.3);
            Background(digium/ToHearTheseOptionsRepeatedPleaseHold);
        }
        WaitExten(5);
        if( ${attempts} < 2 ) goto repeat;
        Background(digium/YouHaveMadeNoSelection);
        Background(digium/ThisCallWillBeEnded);
        Background(goodbye);
```

```
        Hangup();
    }
}
```

The Contexts referenced from the queues.conf file

```
context sales {
    0 => goto dispatch,s,1;
    8 => Voicemail(${SALESVM});
    s => {
        Ringing();
        Wait(2);

Background(digium/ThankYouForContactingTheDigiumSalesDepartment);
        WaitExten(0.3);

Background(digium/PleaseHoldAndYourCallWillBeAnsweredByOurNextAvailabl

        WaitExten(0.3);

Background(digium/AtAnyTimeYouMayPress0ToSpeakWithAnOperatorOr8ToLeave

        Set(CALLERID(name)=Sales);
        Queue(sales-general,t);
        Set(CALLERID(name)=EmptySalQ);
        goto dispatch,s,1;
        Playback(goodbye);
        Hangup();
    }
}
```

Please note that there is only one attempt to queue a call in the sales queue. All sales agents that are logged in will be rung.

```
context customerservice {
    0 => {
        SetCIDName(CSVTrans);
        goto dispatch|s|1;
    }
    8 => Voicemail(${CUSTSERVVM});
    s => {
        Ringing();
        Wait(2);
        Background(digium/ThankYouForCallingDigiumCustomerService);
        WaitExten(0.3);
        notracking:
Background(digium/PleaseWaitForTheNextAvailableCustomerServiceRepresen

        WaitExten(0.3);

Background(digium/AtAnyTimeYouMayPress0ToSpeakWithAnOperatorOr8ToLeave

        Set(CALLERID(name)=Cust Svc);
        Set(QUEUE_MAX_PENALTY=10);
        Queue(customerservice,t);
        Set(QUEUE_MAX_PENALTY=0);
        Queue(customerservice,t);
        Set(CALLERID(name)=EmptyCSVQ);
        goto dispatch,s,1;

Background(digium/NoCustomerServiceRepresentativesAreAvailableAtThisTi

Background(digium/PleaseLeaveAMessageInTheCustomerServiceVoiceMailBox)

        Voicemail(${CUSTSERVVM});
        Playback(goodbye);
        Hangup();
    }
}
```

Note that calls coming into customerservice will first be try to queue calls to those agents with a QUEUE_MAX_PENALTY of 10, and if none are available, then all agents are rung.

```
context dispatch {
    s => {
        Ringing();
        Wait(2);
        Background(digium/ThankYouForCallingDigium);
        WaitExten(0.3);

Background(digium/YourCallWillBeAnsweredByOurNextAvailableOperator);

        Background(digium/PleaseHold);
        Set(QUEUE_MAX_PENALTY=10);
        Queue(dispatch|t);
        Set(QUEUE_MAX_PENALTY=20);
        Queue(dispatch|t);
        Set(QUEUE_MAX_PENALTY=0);
        Queue(dispatch|t);
        Background(digium/NoOneIsAvailableToTakeYourCall);

Background(digium/PleaseLeaveAMessageInOurGeneralVoiceMailBox);
        Voicemail(${DISPATCHVM});
        Playback(goodbye);
        Hangup();
    }
}
```

And in the dispatch context, first agents of priority 10 are tried, then 20, and if none are available, all agents are tried.

Notice that a common pattern is followed in each of the three queue contexts:

First, you set QUEUE_MAX_PENALTY to a value, then you call Queue(queue-name,option,...) (see the Queue application documetation for details)

In the above, note that the "t" option is specified, and this allows the agent picking up the incoming call the luxury of transferring the call to other parties.

The purpose of specifying the QUEUE_MAX_PENALTY is to develop a set of priorities amongst agents. By the above usage, agents with lower number priorities will be given the calls first, and then, if no-one picks up the call, the QUEUE_MAX_PENALTY will be incremented, and the queue tried
again. Hopefully, along the line, someone will pick up the call, and the Queue application will end with a hangup.

The final attempt to queue in most of our examples sets the QUEUE_MAX_PENALTY to zero, which means to try all available agents.

Assigning Agents to Queues

In this example dialplan, we want to be able to add and remove agents to handle incoming calls, as they feel they are available. As they log in, they are added to the queue's agent list, and as they log out, they are removed. If no agents are available, the queue command will terminate, and it is the duty of the dialplan to do something appropriate, be it sending the incoming caller to voicemail, or trying the queue again with a higher QUEUE_MAX_PENALTY.

Because a single agent can make themselves available to more than one queue, the process of joining multiple queues can be handled automatically by the dialplan.

Agents Log In and Out

```
context queues-loginout {
    6092 => {
        Answer();
        Read(AGENT_NUMBER,agent-enternum);
        VMAuthenticate(${AGENT_NUMBER}@default,s);
        Set(queue-announce-success=1);
        goto queues-manip,I${AGENT_NUMBER},1;
    }
    6093 => {
        Answer();
        Read(AGENT_NUMBER,agent-enternum);
        Set(queue-announce-success=1);
        goto queues-manip,O${AGENT_NUMBER},1;
    }
}
```

In the above contexts, the agents dial 6092 to log into their queues, and they dial 6093 to log out of their queues. The agent is prompted for their agent number, and if they are logging in, their passcode, and then they are transferred to the proper extension in the queues-manip context. The queues-manip context does all the actual work:

```
context queues-manip {
    // Raquel Squelch
    _[IO]6121 => {
        &queue-addremove(dispatch,10,${EXTEN});
        &queue-success(${EXTEN});
    }
    // Brittanica Spears
    _[IO]6165 => {
        &queue-addremove(dispatch,20,${EXTEN});
        &queue-success(${EXTEN});
    }
    // Rock Hudson
    _[IO]6170 => {
        &queue-addremove(sales-general,10,${EXTEN});
        &queue-addremove(customerservice,20,${EXTEN});
        &queue-addremove(dispatch,30,${EXTEN});
        &queue-success(${EXTEN});
    }
    // Saline Dye-on
    _[IO]6070 => {
        &queue-addremove(sales-general,20,${EXTEN});
        &queue-addremove(customerservice,30,${EXTEN});
        &queue-addremove(dispatch,30,${EXTEN});
        &queue-success(${EXTEN});
    }
}
```

In the above extensions, note that the queue-addremove macro is used to actually add or remove the agent from the applicable queue, with the applicable priority level. Note that agents with a priority level of 10 will be called before agents with levels of 20 or 30.

In the above example, Raquel will be dialed first in the dispatch queue, if she has logged in. If she is not, then the second call of Queue() with priority of 20 will dial Brittanica if she is present, otherwise the third call of Queue() with MAX_PENALTY of 0 will dial Rock and Saline simultaneously.

Also note that Rock will be among the first to be called in the sales-general queue, and among the last in the dispatch queue. As you can see in main menu, the callerID is set in the main menu so they can tell which queue incoming calls are coming from.

The call to queue-success() gives some feedback to the agent as they log in and out, that the process has completed.

```
macro queue-success(exten) {
    if( ${queue-announce-success} > 0 ) {
        switch(${exten:0:1}) {
            case I:
                Playback(agent-loginok);
                Hangup();
                break;
            case O:
                Playback(agent-loggedoff);
                Hangup();
                break;
        }
    }
}
```

The queue-addremove macro is defined in this manner:

```
macro queue-addremove(queuename,penalty,exten) {
    switch(${exten:0:1}) {
        case I: // Login

AddQueueMember(${queuename},Local/${exten:1}@agents,${penalty});
            break;
        case O: // Logout
            RemoveQueueMember(${queuename},Local/${exten:1}@agents);

            break;
        case P: // Pause
            PauseQueueMember(${queuename},Local/${exten:1}@agents);
            break;
        case U: // Unpause

UnpauseQueueMember(${queuename},Local/${exten:1}@agents);
            break;
        default: // Invalid
            Playback(invalid);
            break;
    }
}
```

Basically, it uses the first character of the exten variable, to determine the proper actions to take. In the above dial plan code, only the cases I or O are used, which correspond to the Login and Logout actions.

Controlling the way Queues Call Agents

Notice in the above, that the commands to manipulate agents in queues have "@agents" in their arguments. This is a reference to the agents context:

```
context agents {
    // General sales queue
    8010 => {
        Set(QUEUE_MAX_PENALTY=10);
        Queue(sales-general,t);
        Set(QUEUE_MAX_PENALTY=0);
        Queue(sales-general,t);
        Set(CALLERID(name)=EmptySalQ);
        goto dispatch,s,1;
    }
    // Customer Service queue
    8011 => {
        Set(QUEUE_MAX_PENALTY=10);
        Queue(customerservice,t);
        Set(QUEUE_MAX_PENALTY=0);
        Queue(customerservice,t);
        Set(CALLERID(name)=EMptyCSVQ);
        goto dispatch,s,1;
    }
    8013 => {
        Dial(iax2/sweatshop/9456@from-ecstacy);
        Set(CALLERID(name)=EmptySupQ);
        Set(QUEUE_MAX_PENALTY=10);
        Queue(support-dispatch,t);
        Set(QUEUE_MAX_PENALTY=20);
        Queue(support-dispatch,t);
        Set(QUEUE_MAX_PENALTY=0); // means no max
        Queue(support-dispatch,t);
        goto dispatch,s,1;
    }
    6121 => &callagent(${RAQUEL},${EXTEN});
    6165 => &callagent(${SPEARS},${EXTEN});
    6170 => &callagent(${ROCK},${EXTEN});
    6070 => &callagent(${SALINE},${EXTEN});
}
```

In the above, the variables ${RAQUEL}, etc stand for actual devices to ring that person's phone (like DAHDI/37).

The 8010, 8011, and 8013 extensions are purely for transferring incoming callers to queues. For instance, a customer service agent might want to transfer the caller to talk to sales. The agent only has to transfer to extension 8010, in this case.

Here is the callagent macro, note that if a person in the queue is called, but does not answer, then they are automatically removed from the queue.

```
macro callagent(device,exten) {
    if( ${GROUP_COUNT(${exten}@agents)}=0 ) {
        Set(OUTBOUND_GROUP_ONCE=${exten}@agents);
        Dial(${device},300,t);
        switch(${DIALSTATUS}) {
            case BUSY:
                Busy();
                break;
            case NOANSWER:
                Set(queue-announce-success=0);
                goto queues-manip,O${exten},1;
            default:
                Hangup();
                break;
        }
    }
    else {
        Busy();
    }
}
```

In the callagent macro above, the ${exten} will be 6121, or 6165, etc, which is the extension of the agent.

The use of the GROUP_COUNT, and OUTBOUND_GROUP follow this line of thinking. Incoming calls can be queued to ring all agents in the current priority. If some of those agents are already talking, they would get bothersome call-waiting tones. To avoid this inconvenience, when an agent gets a call, the OUTBOUND_GROUP assigns that conversation to the group specified, for instance 6171@agents. The ${GROUP_COUNT()} variable on a subsequent call should return "1" for that group. If GROUP_COUNT returns 1, then the busy() is returned without actually trying to dial the agent.

Queue Pre-Acknowledgement Messages

If you would like to have a pre acknowledge message with option to reject the message you can use the following dialplan Macro as a base with the 'M' dial argument.

```
[macro-screen]
exten=>s,1,Wait(.25)
exten=>s,2,Read(ACCEPT,screen-callee-options,1)
exten=>s,3,Gotoif($[${ACCEPT} = 1] ?50)
exten=>s,4,Gotoif($[${ACCEPT} = 2] ?30)
exten=>s,5,Gotoif($[${ACCEPT} = 3] ?40)
exten=>s,6,Gotoif($[${ACCEPT} = 4] ?30:30)
exten=>s,30,Set(MACRO_RESULT=CONTINUE)
exten=>s,40,Read(TEXTEN,custom/screen-exten,)
exten=>s,41,Gotoif($[${LEN(${TEXTEN})} = 3]?42:45)
exten=>s,42,Set(MACRO_RESULT=GOTO:from-internal^${TEXTEN}^1)
exten=>s,45,Gotoif($[${TEXTEN} = 0] ?46:4)
exten=>s,46,Set(MACRO_RESULT=CONTINUE)
exten=>s,50,Playback(after-the-tone)
exten=>s,51,Playback(connected)
exten=>s,52,Playback(beep)
```

Queue Caveats

In the above examples, some of the possible error checking has been omitted, to reduce clutter and make the examples clearer.

Queue Logs

In order to properly manage ACD queues, it is important to be able to keep track of details of call setups and teardowns in much greater detail than traditional call detail records provide. In order to support this, extensive and detailed tracing of every queued call is stored in the queue log, located (by default) in /var/log/asterisk/queue_log.

These are the events (and associated information) in the queue log:

- ABANDON(positionorigpositionwaittime) - The caller abandoned their position in the queue. The position is the caller's position in the queue when they hungup, the origposition is the original position the caller was when they first entered the queue, and the waittime is how long the call had been waiting in the queue at the time of disconnect.

- AGENTDUMP - The agent dumped the caller while listening to the queue announcement.

- AGENTLOGIN(channel) - The agent logged in. The channel is recorded.

- AGENTCALLBACKLOGIN(exten@context) - The callback agent logged in. The login extension and context is recorded.

- AGENTLOGOFF(channellogintime) - The agent logged off. The channel is recorded, along with the total time the agent was logged in.

- AGENTCALLBACKLOGOFF(exten@contextlogintimereason) - The callback agent logged off. The last login extension and context is recorded, along with the total time the agent was logged in, and the reason for the logoff if it was not a normal logoff (e.g., Autologoff, Chanunavail)

- COMPLETEAGENT(holdtimecalltimeorigposition) - The caller was connected to an agent, and the call was terminated normally by the agent. The caller's hold time and the length of the call are both recorded. The caller's original position in the queue is recorded in origposition.

- COMPLETECALLER(holdtimecalltimeorigposition) - The caller was connected to an agent, and the call was terminated normally by the

caller. The caller's hold time and the length of the call are both recorded. The caller's original position in the queue is recorded in origposition.

- CONFIGRELOAD - The configuration has been reloaded (e.g. with asterisk -rx reload)

- CONNECT(holdtimebridgedchanneluniqueidringtime) - The caller was connected to an agent. Hold time represents the amount of time the caller was on hold. The bridged channel unique ID contains the unique ID of the queue member channel that is taking the call. This is useful when trying to link recording filenames to a particular call in the queue. Ringtime represents the time the queue members phone was ringing prior to being answered.

- ENTERQUEUE(urlcallerid) - A call has entered the queue. URL (if specified) and Caller*ID are placed in the log.

- EXITEMPTY(positionorigpositionwaittime) - The caller was exited from the queue forcefully because the queue had no reachable members and it's configured to do that to callers when there are no reachable members. The position is the caller's position in the queue when they hungup, the origposition is the original position the caller was when they first entered the queue, and the waittime is how long the call had been waiting in the queue at the time of disconnect.

- EXITWITHKEY(keypositionorigpositionwaittime) - The caller elected to use a menu key to exit the queue. The key and the caller's position in the queue are recorded. The caller's entry position and amoutn of time waited is also recorded.

- EXITWITHTIMEOUT(positionorigpositionwaittime) - The caller was on hold too long and the timeout expired. The position in the queue when the timeout occurred, the entry position, and the amount of time waited are logged.

- QUEUESTART - The queueing system has been started for the first time this session.

- RINGNOANSWER(ringtime) - After trying for ringtime ms to connect to the available queue member, the attempt ended without the member picking up the call. Bad queue member!

- SYSCOMPAT - A call was answered by an agent, but the call was dropped because the channels were not compatible.

- TRANSFER(extensioncontextholdtimecalltimeorigposition) - Caller was transferred to a different extension. Context and extension are recorded. The caller's hold time and the length of the call are both recorded, as is the caller's entry position at the time of the transfer. PLEASE remember that transfers performed by SIP UA's by way of a reinvite may not always be caught by Asterisk and trigger off this event. The only way to be 100% sure that you will get this event when a transfer is performed by a queue member is to use the built-in transfer functionality of Asterisk.

Asterisk Security Framework

Attacks on Voice over IP networks are becoming increasingly more common. It has become clear that we must do something within Asterisk to help mitigate these attacks.

Through a number of discussions with groups of developers in the Asterisk community, the general consensus is that the best thing that we can do within Asterisk is to build a framework which recognizes and reports events that could potentially have security implications. Each channel driver has a different concept of what is an "event", and then each administrator has different thresholds of what is a "bad" event and what is a restorative event. The process of acting upon this information is left to an external program to correlate and then take action - block traffic, modify dialing rules, etc. It was decided that embedding actions inside of Asterisk was inappropriate, as the complexity of construction of such rule sets is difficult and there was no agreement on where rules should be enabled or how they should be processed. The addition of a major section of code to handle rule expiration and severity interpretation was significant. As a final determining factor, there are external programs and services which already parse log files and act in concert with packet filters or external devices to protect or alter network security models for IP connected hosts.

Security Framework Overview

This section discusses the architecture of the Asterisk modifications being proposed.

There are two main components that we propose for the initial implementation of the security framework:

- Security Event Generation
- Security Event Logger

Security Event Generation

The ast_event API is used for the generation of security events. That way, the events are in an easily interpretable format within Asterisk to make it easy to write modules that do things with them. There are also some helper data structures and functions to aid Asterisk modules in reporting these security events with the proper contents.

The next section of this document contains the current list of security events being proposed. Each security event type has some required pieces of information and some other optional pieces of information.

Subscribing to security events from within Asterisk can be done by subscribing to events of type AST_EVENT_SECURITY. These events have an information element, AST_EVENT_IE_SECURITY_EVENT, which identifies the security event sub-type (from the list described in the next section). The result of the information elements in the events contain the required and optional meta data associated with the event sub-type.

Asterisk Security Event Logger

In addition to the infrastructure for generating the events, one module that is a consumer of these events has been implemented.

Asterisk trunk was recently updated to include support for dynamic logger levels. This module takes advantage of this functionality to create a custom "security" logger level. Then, when this module is in use, logger.conf can be configured to put security events into a file

security_log => security

The content of this file is a well defined and easily interpretable format for external scripts to read and act upon. The definition for the format of the log file is described later in this chapter.

Security Events to Log

```
(-) required
(+) optional

Invalid Account ID
  (-) Local address family/IP address/port/transport
  (-) Remote address family/IP address/port/transport
  (-) Service (SIP, AMI, IAX2, ...)
  (-) System Name
  (+) Module
```

```
   (+) Account ID (username, etc)
   (+) Session ID (CallID, etc)
   (+) Session timestamp (required if Session ID present)
   (-) Event timestamp (sub-second precision)
Failed ACL match
   -> everything from invalid account ID
   (+) Name of ACL (when we have named ACLs)

Invalid Challenge/Response
   -> everything from invalid account ID
   (-) Challenge
   (-) Response
   (-) Expected Response
Invalid Password
   -> everything from invalid account ID

Successful Authentication
   -> informational event
   -> everything from invalid account ID

Invalid formatting of Request
   -> everything from invalid account ID
   -> account ID optional
   (-) Request Type
   (+) Request parameters
Session Limit Reached (such as a call limit)
   -> everything from invalid account ID

Memory Limit Reached
   -> everything from invalid account ID
Maximum Load Average Reached
   -> everything from invalid account ID

Request Not Allowed
   -> everything from invalid account ID
   (-) Request Type
   (+) Request parameters
Request Not Supported
   -> everything from invalid account ID
   (-) Request Type

Authentication Method Not Allowed
   -> everything from invalid account ID
   (-) Authentication Method attempted
In dialog message from unexpected host
```

```
    -> everything from invalid account ID
    (-) expected host
```

Security Log File Format

The beginning of each line in the log file is the same as it is for other logger levels within Asterisk.

```
[Feb 11 07:57:03] SECURITY[23736] res_security_log.c: <...>
```

The part of the log entry identified by \<...\> is where the security event content resides. The security event content is a comma separated list of key value pairs. The key is the information element type, and the value is a quoted string that contains the associated meta data for that information element. Any embedded quotes within the content are escaped with a backslash.

INFORMATION_ELEMENT_1="IE1 content",INFORMATION_ELEMENT_2="IE2 content"

The following table includes potential information elements and what the associated content looks like:

- IE: SecurityEvent
  Content: This is the security event sub-type.
  Values: FailedACL, InvalidAccountID, SessionLimit, MemoryLimit, LoadAverageLimit, RequestNotSupported, RequestNotAllowed, AuthMethodNotAllowed, ReqBadFormat, UnexpectedAddress, ChallengeResponseFailed, InvalidPassword

- IE: EventVersion
  Content: This is a numeric value that indicates when updates are made to the content of the event.
  Values: Monotonically increasing integer, starting at 1

- IE: Service
  Content: This is the Asterisk service that generated the event.
  Values: TEST, SIP, AMI

- IE: Module
  Content: This is the Asterisk module that generated the event.
  Values: chan_sip

- IE: AccountID
  Content: This is a string used to identify the account associated with the event. In most cases, this would be a username.

- IE: SessionID
  Content: This is a string used to identify the session associated with the event. The format of the session identifier is specific to the service. In the case of SIP, this would be the Call-ID.

- IE: SessionTV
  Content: The time that the session associated with the SessionID started.
  Values: <seconds><microseconds> since epoch

- IE: ACLName
  Content: This is a string that identifies which named ACL is associated with this event.

- IE: LocalAddress
  Content: This is the local address that was contacted for the related event.
  Values: <Address Family>/<Transport>/<Address>/<Port>
  Examples: -> IPV4/UDP/192.168.1.1/5060 -> IPV4/TCP/192.168.1.1/5038

- IE: RemoteAddress
  Content: This is the remote address associated with the event.

Examples: -> IPV4/UDP/192.168.1.2/5060 -> IPV4/TCP/192.168.1.2/5038

- IE: ExpectedAddress
  Content: This is the address that was expected to be the remote address.
  Examples: -> IPV4/UDP/192.168.1.2/5060 -> IPV4/TCP/192.168.1.2/5038

- IE: EventTV
  Content: This is the timestamp of when the event occurred.
  Values: <seconds><microseconds> since epoch

- IE: RequestType
  Content: This is a service specific string that represents the invalid request

- IE: RequestParams
  Content: This is a service specific string that represents relevant parameters given with a request that was considered invalid.

- IE: AuthMethod
  Content: This is a service specific string that represents an authentication method that was used or requested.

- IE: Challenge
  Content: This is a service specific string that represents the challenge provided to a user attempting challenge/response authentication.

- IE: Response
  Content: This is a service specific string that represents the response received from a user attempting challenge/response authentication.

- IE: ExpectedResponse
  Content: This is a service specific string that represents the response that was expected to be received from a user attempting challenge/response authentication.

Asterisk Sounds Packages

Asterisk utilizes a variety of sound prompts that are available in several file formats and languages. Multiple languages and formats can be installed on the same system, and Asterisk will utilize prompts from languages installed, and will automatically pick the least CPU intensive format that is available on the system (based on codecs in use, in additional to the codec and format modules installed and available).

In addition to the prompts available with Asterisk, you can create your own sets of prompts and utilize them as well. This document will tell you how the prompts available for Asterisk are created so that the prompts you create can be as close and consistent in the quality and volume levels as those shipped with Asterisk.

Getting the Sounds Tools

The sounds tools are available in the publicly accessible repotools repository. You can check these tools out with Subversion via the following command:

```
# svn co http://svn.asterisk.org/svn/repotools
```

The sound tools are available in the subdirectory sound_tools/ which contains the following directories:

- audiofilter
- makeg722
- scripts

About the Sounds Tools

The following sections will describe the sound tools in more detail and explain what they are used for in the sounds package creation process.

audiofilter

The audiofilter application is used to "tune" the sound files in such a way that they sound good when being used while in a compressed format. The values in the scripts for creating the sound files supplied in repotools is essentially a high-pass filter that drops out audio below 100Hz (or so).

(There is an ITU specification that states for 8KHz audio that is being compressed frequencies below a certain threshold should be removed because they make the resulting compressed audio sound worse than it should.)

The audiofilter application is used by the 'converter' script located in the scripts subdirectory of repotools/sound_tools. The values being passed to the audiofilter application is as follows:

```
audiofilter -n 0.86916 -1.73829 0.86916 -d 1.00000 -1.74152 0.77536
```

The two options -n and -d are 'numerator' and 'denominator'. Per the author, Jean-Marc Valin, "These values are filter coefficients (-n means numerator, -d is denominator) expressed in the z-transform domain. There represent an elliptic filter that I designed with Octave such that 'the result sounds good'."

makeg722

The makeg722 application is used by the 'converters' script to generate the G.722 sound files that are shipped with Asterisk. It starts with the RAW sound files and then converts them to G.722.

scripts

The scripts folder is where all the magic happens. These are the scripts that the Asterisk open source team use to build the packaged audio files for the various formats that are distributed with Asterisk.

- chkcore - used to check that the contents of core-sounds-lang.txt are in sync
- chkextra - same as above, but checks the extra sound files
- mkcore - script used to generate the core sounds packages
- mkextra - script used to generate the extra sounds packages
- mkmoh - script used to generate the music on hold packages
- converters - script used to convert the master files to various formats

Call Completion Supplementary Services (CCSS)

Introduction

A new feature for Asterisk 1.8 is Call Completion Supplementary Services. This document aims to explain the system and how to use it. In addition, this document examines some potential

troublesome points which administrators may come across during their deployment of the feature.

What is CCSS?

Call Completion Supplementary Services (often abbreviated "CCSS" or simply "CC") allow for a caller to let Asterisk automatically alert him when a called party has become available, given that a previous call to that party failed for some reason. The two services offered are Call Completion on Busy Subscriber (CCBS) and Call Completion on No Response (CCNR). To illustrate, let's say that Alice attempts to call Bob. Bob is currently on a phone call with Carol, though, so Alice hears a busy signal. In this situation, assuming that Asterisk has been configured to allow for such activity, Alice would be able to request CCBS. Once Bob has finished his phone call, Alice will be alerted. Alice can then attempt to call Bob again.

CCSS Glossary

In this document, we will use some terms which may require clarification. Most of these terms are specific to Asterisk, and are by no means standard.

- CCBS: Call Completion on Busy Subscriber. When a call fails because the recipient's phone is busy, the caller will have the opportunity to request CCBS. When the recipient's phone is no longer busy, the caller will be alerted. The means by which the caller is alerted is dependent upon the type of agent used by the caller.

- CCNR: Call Completion on No Response. When a call fails because the recipient does not answer the phone, the caller will have the opportun- ity to request CCNR. When the recipient's phone becomes busy and then is no longer busy, the caller will be alerted. The means by which the caller is alerted is dependent upon the type of the agent used by the caller.

- Agent: The agent is the entity within Asterisk that communicates with and acts on behalf of the calling party.

- Monitor: The monitor is the entity within Asterisk that communicates with and monitors the status of the called party.

- Generic Agent: A generic agent is an agent that uses protocol-agnostic methods to communicate with the caller. Generic agents should only be used for phones, and never should be used for "trunks."

- Generic Monitor: A generic monitor is a monitor that uses protocol- agnostic methods to monitor the status of the called party. Like with generic agents, generic monitors should only be used for phones.

- Native Agent: The opposite of a generic agent. A native agent uses protocol-specific messages to communicate with the calling party. Native agents may be used for both phones and trunks, but it must be known ahead of time that the device with which Asterisk is communica- ting supports the necessary signaling.

- Native Monitor: The opposite of a generic monitor. A native monitor uses protocol-specific messages to subscribe to and receive notifica- tion of the status of the called party. Native monitors may be used for both phones and trunks, but it must be known ahead of time that the device with which Asterisk is communicating supports the necessary signaling.

- Offer: An offer of CC refers to the notification received by the caller that he may request CC.

- Request: When the caller decides that he would like to subscribe to CC, he will make a request for CC. Furthermore, the term may refer to any outstanding requests made by callers.

- Recall: When the caller attempts to call the recipient after being alerted that the recipient is available, this action is referred to as a "recall."

The Call Completion Process

The Initial Call

The only requirement for the use of CC is to configure an agent for the caller and a monitor for at

least one recipient of the call. This is controlled using the cc_agent_policy for the caller and the cc_monitor_policy for the recipient. For more information about these configuration settings, see configs/samples/ccss.conf.sample. If the agent for the caller is set to something other than "never" and at least one recipient has his monitor set to something other than "never," then CC will be offered to the caller at the end of the call.

Once the initial call has been hung up, the configured cc_offer_timer for the caller will be started. If the caller wishes to request CC for the previous call, he must do so before the timer expires.

Requesting CC

Requesting CC is done differently depending on the type of agent the caller is using.

With generic agents, the CallCompletionRequest application must be called in order to request CC. There are two different ways in which this may be called. It may either be called before the caller hangs up during the initial call, or the caller may hang up from the initial call and dial an extension which calls the CallCompletionRequest application. If the second method is used, then the caller will have until the cc_offer_timer expires to request CC.

With native agents, the method for requesting CC is dependent upon the technology being used, coupled with the make of equipment. It may be possible to request CC using a programmable key on a phone or by clicking a button on a console. If you are using equipment which can natively support CC but do not know the means by which to request it, then contact the equipment manufacturer for more information.

Cancelling CC

CC may be canceled after it has been requested. The method by which this is accomplished differs based on the type of agent the calling party uses.

When using a generic agent, the dialplan application CallRequestCancel is used to cancel CC. When using a native monitor, the method by which CC is cancelled depends on the protocol used. Likely, this will be done using a button on a phone.

Keep in mind that if CC is cancelled, it cannot be un-cancelled.

Monitoring the Called Party

Once the caller has requested CC, then Asterisk's job is to monitor the progress of the called parties. It is at this point that Asterisk allocates the necessary resources to monitor the called parties.

A generic monitor uses Asterisk's device state subsystem in order to determine when the called party has become available. For both CCBS and CCNR, Asterisk simply waits for the phone's state to change to a "not in use" state from a different state. Once this happens, then Asterisk will consider the called party to be available and will alert the caller.

A native monitor relies on the network to send a protocol-specific message when the called party has become available. When Asterisk receives such a message, it will consider the called party to be available and will alert the caller.

Note that since a single caller may dial multiple parties, a monitor is used for each called party. It is within reason that different called parties will use different types of monitors for the same CC request.

Alerting the Caller

Once Asterisk has determined that the called party has become available the time comes for Asterisk to alert the caller that the called party has become available. The method by which this is done differs based on the type of agent in use.

If a generic agent is used, then Asterisk will originate a call to the calling party. Upon answering the call, if a callback macro has been configured, then that macro will be executed on the calling party's channel. After the macro has completed, an outbound call will be issued to the parties involved in the original call.

If a native agent is used, then Asterisk will send an appropriate notification message to the calling party to alert it that it may now attempt its recall. How this is presented to the caller is dependent upon the protocol and equipment that the caller is using. It is possible that the calling party's phone will ring and a recall will be triggered upon answering the phone, or it may be that the user has a specific button that he may press to initiate a recall.

If the Caller is unavailable

When the called party has become available, it is possible that when Asterisk attempts to alert the calling party of the called party's availability, the calling party itself will have become unavailable. If this is the case, then Asterisk will suspend monitoring of the called party and will instead monitor the availability of the calling party. The monitoring procedure for the calling party is the same as is used in the section "Monitoring the Called Party." In other words, the method by which the calling party is monitored is dependent upon the type of agent used by the caller.

Once Asterisk has determined that the calling party has become available again, Asterisk will then move back to the process used in the section "Monitoring the Called Party."

The CC recall

The calling party will make its recall to the same extension that was dialed. Asterisk will provide a channel variable, CC_INTERFACES, to be used as an argument to the Dial application for CC recalls. It is strongly recommended that you use this channel variable during a CC recall. Listed are two reasons:

1. The dialplan may be written in such a way that the dialed destintations are dynamically generated. With such a dialplan, it cannot be guaranteed that the same interfaces will be recalled.
2. For calling destinations with native CC monitors, it may be necessary to dial a special string in order to notify the channel driver that the number being dialed is actually part of a CC recall.

> ⚠️ Even if your call gets routed through local channels, the CC_INTERFACES variable will be populated with the appropriate values for that specific extension.

When the called parties are dialed, it is expected that a called party will answer, since Asterisk had previously determined that the party was available. However, it is possible that the called party may choose not to respond to the call, or he could have become busy again. In such a

situation, the calling party must re-request CC if he wishes to still be alerted when the calling party has become available.

Call Completion Info and Tips

- Be aware when using a generic agent that the max_cc_agents configuration parameter is ignored. The main driving reason for this is that the mechanism for cancelling CC when using a generic agent would become much more potentially confusing to execute. By limiting a calling party to having a single request, there is only ever a single request to be cancelled, making the process simple.

- Keep in mind that no matter what CC agent type is being used, a CC request can only be made for the latest call issued.

- If available timers are running on multiple called parties, it is possible that one of the timers may expire before the others do. If such a situation occurs, then the interface on which the timer expired will cease to be monitored. If, though, one of the other called parties becomes available before his available timer expires, the called party whose available timer had previously expired will still be included in the CC_INTERFACES channel variable on the recall.
- It is strongly recommended that lots of thought is placed into the settings of the CC timers. Our general recommendation is that timers for phones should be set shorter than those for trunks. The reason for this is that it makes it less likely for a link in the middle of a network to cause CC to fail.

- CC can potentially be a memory hog if used irresponsibly. The following are recommendations to help curb the amount of resources required by the CC engine. First, limit the maximum number of CC requests in the system using the cc_max_requests option in ccss.conf. Second, set the cc_offer_timer low for your callers. Since it is likely that most calls will not result in a CC request, it is a good idea to set this value to something low so that information for calls does not stick around in memory for long. The final thing that can be done is to conditionally set the cc_agent_policy to "never" using the CALLCOMPLETION dialplan function. By doing this, no CC information will be kept around after the call completes.

- It is possible to request CCNR on answered calls. The reason for this is that it is impossible to know whether a call that is answered has actually been answered by a person or by something such as voicemail or some other IVR.

- Not all channel drivers have had the ability to set CC config parameters in their configuration files added yet. At the time of this writing (2009 Oct), only chan_sip has had this ability added, with short-term plans to add this to chan_dahdi as well. It is possible to set CC configuration parameters for other channel types, though. For these channel types, the setting of the parameters can only be accomplished using the CALLCOMPLETION dialplan function.

- It is documented in many places that generic agents and monitors can only be used for phones. In most cases, however, Asterisk has no way of distinguishing between a phone and a trunk itself. The result is that Asterisk will happily let you violate the advice given and allow you to set up a trunk with a generic monitor or agent. While this will not cause anything catastrophic to occur, the behavior will most definitely not be what you want.

- At the time of this writing (2009 Oct), Asterisk is the only known SIP stack to write an implementation of draft-ietf-bliss-call-completion-04. As a result, it is recommended that for your SIP phones, use a generic agent and monitor. For SIP trunks, you will only be able to use CC if the other end is terminated by another Asterisk server running version 1.8 or later.

- Native SIP CC will only work if the xml2 development library is installed. This is because we use libxml2 in order to parse PIDF bodies of PUBLISH messages received. If, at configure time, Asterisk cannot detect that the necessary library is installed, then native CC in SIP will be disabled. Attempts to set a channel or SIP peer to use native CC will be changed to having CC being disabled instead.

- If the Dial application is called multiple times by a single extension, CC will only be offered to the caller for the parties called by the first instantiation of Dial.

- If a phone forwards a call, then CC may only be requested for the phone that executed the call forward. CC may not be requested for the phone to which the call was forwarded.
- CC is currently only supported by the Dial application. Queue, Followme, and Page do not support CC because it is not particularly useful for those applications.

- Generic CC relies heavily on accurate device state reporting. In particular, when using SIP phones it is vital to be sure that device state is updated properly when using them. In order to facilitate proper device state handling, be sure to set callcounter=yes for all peers and to set limitonpeers=yes in the general section of sip.conf

- When using SIP CC (i.e. native CC over SIP), it is important that your minexpiry and maxexpiry values allow for available timers to run as little or as long as they are configured. When an Asterisk server requests call completion over SIP, it sends a SUBSCRIBE message with an Expires header set to the number of seconds that the available timer should run. If the Asterisk server that receives this SUBSCRIBE has a maxexpiry set lower than what is in the received Expires header, then the available timer will only run for maxexpiry seconds.

- CC support for ETSI PTP and Q.SIG requires CallerID information to match CC requests with CC offers. For Q.SIG, depending upon the options negotiated when CC is requested, the CallerID information needs to be callable as well.

- As with all Asterisk components, CC is not perfect. If you should find a bug or wish to enhance the feature, please open an issue on https://issues.asterisk.org. If writing an enhancement, please be sure to include a patch for the enhancement, or else the issue will be closed.

Generic Call Completion Example

The following is an incredibly bare-bones example sip.conf and dialplan to show basic usage of generic call completion. It is likely that if you have a more complex setup, you will need to make use of items like the CALLCOMPLETION dialplan function or the CC_INTERFACES channel variable.
First, let's establish a very simple sip.conf to use for this

<table>
<tr><td align="center">sip.conf</td></tr>
</table>

```
[Mark]
context=phone_calls
cc_agent_policy=generic
cc_monitor_policy=generic ;We will accept defaults for the rest of
the cc parameters
;We also are not concerned with other SIP details for this
;example

[Richard]
context=phone_calls
cc_agent_policy=generic
cc_monitor_policy=generic
```

Now, let's write a simple dialplan

<table>
<tr><td align="center">extensions.conf</td></tr>
</table>

```
[phone_calls]
exten => 1000,1,Dial(SIP/Mark,20)
exten => 1000,n,Hangup
exten => 2000,1,Dial(SIP/Richard,20)
exten => 2000,n,Hangup
exten => 30,1,CallCompletionRequest
exten => 30,n,Hangup
exten => 31,1,CallCompletionCancel
exten => 31,n,Hangup
```

Scenario 1: Mark picks up his phone and dials Richard by dialing 2000. Richard is currently on a call, so Mark hears a busy signal. Mark then hangs up, picks up the phone and dials 30 to call the CallCompletionRequest application. After some time, Richard finishes his call and hangs up.

Mark is automatically called back by Asterisk. When Mark picks up his phone, Asterisk will dial extension 2000 for him.

Scenario 2: Richard picks up his phone and dials Mark by dialing 1000. Mark has stepped away from his desk, and so he is unable to answer the phone within the 20 second dial timeout. Richard hangs up, picks the phone back up and then dials 30 to request call completion. Mark gets back to his desk and dials somebody's number. When Mark finishes the call, Asterisk detects that Mark's phone has had some activity and has become available again and rings Richard's phone. Once Richard picks up, Asterisk automatically dials exteision 1000 for him.

Scenario 3: Much like scenario 1, Mark calls Richard and Richard is busy. Mark hangs up, picks the phone back up and then dials 30 to request call completion. After a little while, Mark realizes he doesn't actually need to talk to Richard, so he dials 31 to cancel call completion. When Richard becomes free, Mark will not automatically be redialed by Asterisk.

Scenario 4: Richard calls Mark, but Mark is busy. About thirty seconds later, Richard decides that he should perhaps request call completion. However, since Richard's phone has the default cc_offer_timer of 20 seconds, he has run out of time to request call completion. He instead must attempt to dial Mark again manually. If Mark is still busy, Richard can attempt to request call completion on this second call instead.

Scenario 5: Mark calls Richard, and Richard is busy. Mark requests call completion. Richard does not finish his current call for another 2 hours (7200 seconds). Since Mark has the default ccbs_available_timer of 4800 seconds set, Mark will not be automatically recalled by Asterisk when Richard finishes his call.

Scenario 6: Mark calls Richard, and Richard does not respond within the 20 second dial timeout. Mark requests call completion. Richard does not use his phone again for another 4 hours (144000 seconds). Since Mark has the default ccnr_available_timer of 7200 seconds set, Mark will not be automatically recalled by Asterisk when Richard finishes his call.

Call Detail Records (CDR)

Top-level page for all things CDR

CDR Applications

- SetAccount - Set account code for billing
- SetAMAFlags - Sets AMA flags
- NoCDR - Make sure no CDR is saved for a specific call
- ResetCDR - Reset CDR
- ForkCDR - Save current CDR and start a new CDR for this call
- Authenticate - Authenticates and sets the account code
- SetCDRUserField - Set CDR user field
- AppendCDRUserField - Append data to CDR User field

For more information, use the "core show application application" command. You can set default account codes and AMA flags for devices in channel configuration files, like sip.conf, iax.conf etc.

CDR Fields

- accountcode: What account number to use, (string, 20 characters)
- src: Caller*ID number (string, 80 characters)

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

In some cases, uniqueid is appended:

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

> ⚠ If you use IAX2 channels for your calls, and allow 'full' transfers (not media-only transfers), then when the calls is transferred the server in the middle will no longer be involved in the signaling path, and thus will not generate accurate CDRs for that call. If you can, use media-only transfers with IAX2 to avoid this problem, or turn off transfers completely (although this can result in a media latency increase since the media packets have to traverse the middle server(s) in the call).

CDR Variables

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

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

In addition, you can set your own extra variables by using Set(CDR(name)=value). These variables can be output into a text-format CDR by using the cdr_custom CDR driver; see the cdr_custom.conf.sample file in the configs directory for an example of how to do this.

CDR Storage Backends

Top-level page for information about storage backends for Asterisk's CDR engine.

MSSQL CDR Backend

sterisk can currently store CDRs into an MSSQL database in two different ways: cdr_odbc or cdr_tds

Call Data Records can be stored using unixODBC (which requires the FreeTDS package) [cdr_odbc] or directly by using just the FreeTDS package [cdr_tds] The following provide some examples known to get asterisk working with mssql.

> ⚠ Only choose one db connector.

Compile, configure, and install the latest unixODBC package:

```
tar -zxvf unixODBC-2.2.9.tar.gz && cd unixODBC-2.2.9 && ./configure
--sysconfdir=/etc --prefix=/usr --disable-gui && make && make
install
```

Compile, configure, and install the latest FreeTDS package:

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

Compile, or recompile, asterisk so that it will now add support for cdr_odbc.

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

Setup odbc configuration files.

These are working examples from my system. You will need to modify for your setup. You are not required to store usernames or passwords here.

/etc/odbcinst.ini

```
[FreeTDS]
Description = FreeTDS ODBC driver for MSSQL
Driver = /usr/lib/libtdsodbc.so
Setup = /usr/lib/libtdsS.so
FileUsage = 1
```

/etc/odbc.ini

```
[MSSQL-asterisk]
description = Asterisk ODBC for MSSQL
driver = FreeTDS
server = 192.168.1.25
port = 1433
database = voipdb
tds_version = 7.0
language = us_english
```

> ⛔ Only install one database connector. Do not confuse asterisk by using both ODBC (cdr_odbc) and FreeTDS (cdr_tds). This command will erase the contents of cdr_tds.conf
>
> ```
> [ -f /etc/asterisk/cdr_tds.conf ] > /etc/asterisk/cdr_tds.conf
> ```

> ⚠️ unixODBC requires the freeTDS package, but asterisk does not call freeTDS directly.

Now set up cdr_odbc configuration files.

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

/etc/asterisk/cdr_odbc.conf

```
[global]
dsn=MSSQL-asterisk
username=voipdbuser
password=voipdbpass
loguniqueid=yes
```

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

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

Start asterisk in verbose mode.

You should see that asterisk logs a connection to the database and will now record every call to the database when it's complete.

TDS, using cdr_tds

Compile, configure, and install the latest FreeTDS package:

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

Compile, or recompile, asterisk so that it will now add support for cdr_tds.

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

> 🚫 Only install one database connector. Do not confuse asterisk by using both ODBC (cdr_odbc) and FreeTDS (cdr_tds). This command will erase the contents of cdr_odbc.conf
>
> ```
> [ -f /etc/asterisk/cdr_odbc.conf ] > /etc/asterisk/cdr_odbc.conf
> ```

Setup cdr_tds configuration files.

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

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

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

```
CREATE TABLE cdr (
        [accountcode] [varchar] (20) NULL ,
        [src] [varchar] (80) NULL ,
        [dst] [varchar] (80) NULL ,
        [dcontext] [varchar] (80) NULL ,
        [clid] [varchar] (80) NULL ,
        [channel] [varchar] (80) NULL ,
        [dstchannel] [varchar] (80) NULL ,
        [lastapp] [varchar] (80) NULL ,
        [lastdata] [varchar] (80) NULL ,
        [start] [datetime] NULL ,
        [answer] [datetime] NULL ,
        [end] [datetime] NULL ,
        [duration] [int] NULL ,
        [billsec] [int] NULL ,
        [disposition] [varchar] (20) NULL ,
        [amaflags] [varchar] (16) NULL ,
        [uniqueid] [varchar] (150) NULL ,
        [userfield] [varchar] (256) NULL
)
```

Start asterisk in verbose mode.

You should see that asterisk logs a connection to the database and will now record every call to the database when it's complete.

MySQL CDR Backend
ODBC

Using MySQL for CDR records is supported by using ODBC and the cdr_odbc module.

Native

Alternatively, there is a native MySQL CDR module.

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

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

PostgreSQL CDR Backend

If you want to go directly to postgresql database, and have the cdr_pgsql.so compiled you can use the following sample setup. On Debian, before compiling asterisk, just install libpqxx-dev. Other distros will likely have a similiar package.
Once you have the compile done, copy the sample cdr_pgsql.conf file or create your own.

Here is a sample:

/etc/asterisk/cdr_pgsql.conf

```
; Sample Asterisk config file for CDR logging to PostgresSQL
[global]
hostname=localhost
port=5432
dbname=asterisk
password=password
user=postgres
table=cdr
```

Now create a table in postgresql for your cdrs

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

SQLite 2 CDR Backend

SQLite version 2 is supported in cdr_sqlite.

SQLite 3 CDR Backend

SQLite version 3 is supported in cdr_sqlite3_custom.

RADIUS CDR Backend

What is needed

- FreeRADIUS server
- Radiusclient-ng library
- Asterisk PBX

Installation of the Radiusclient library

Download the sources

From http://developer.berlios.de/projects/radiusclient-ng/

Untar the source tarball:

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

Compile and install the library:

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

Configuration of the Radiusclient library

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

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

```
authserver localhost
```

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

```
acctserver localhost
```

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

File "servers"

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

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

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

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

```
File "dictionary"
```

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

file /usr/local/etc/radiusclient-ng/dictionary the line:

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

Install FreeRADIUS Server (Version 1.1.1)

Download sources tarball from:

http://freeradius.org/

Untar, configure, build, and install the server:

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

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

Configuration of the FreeRADIUS Server

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

File "clients.conf"

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

Example:

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

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

File "dictionary"

> ⚠️ As of version 1.1.2, the dictionary.digium file ships with FreeRADIUS.

The following procedure brings the dictionary.digium file to previous versions of FreeRADIUS.

File /usr/local/etc/raddb/dictionary contains the dictionary of FreeRADIUS server. You have to add the same dictionary file (dictionary.digium), which you added to the dictionary of

radiusclient-ng library. You can include it into the main file, adding the following line at the end of file /usr/local/etc/raddb/dictionary:

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

That will include the same new attribute definitions that are used in radiusclient-ng library so the client and server will understand each other.

Asterisk Accounting Configuration

Compilation and installation:

The module will be compiled as long as the radiusclient-ng library has been detected on your system.

By default FreeRADIUS server will log all accounting requests into /usr/local/var/log/radius/radacct directory in form of plain text files. The server will create one file for each hostname in the directory. The following example shows how the log files look like.

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

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

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

Logged Values

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

Calling using Google

Prerequisites

Asterisk communicates with Google Voice and Google Talk using the chan_gtalk Channel Driver and the res_jabber Resource module. Before proceeding, please ensure that both are compiled and part of your installation. Compilation of res_jabber and chan_gtalk for use with Google Talk /

Voice are dependant on the iksemel library files as well as the OpenSSL development libraries presence on your system.

Calling using Google Voice or via the Google Talk web client requires the use of Asterisk 1.8.1.1 or greater. The 1.6.x versions of Asterisk only support calls made using the legacy GoogleTalk external client.

For basic calling between Google Talk web clients, you need a Google Mail account.

For calling to and from the PSTN, you will need a Google Voice account.

In your Google account, you'll want to change the Chat setting from the default of "Automatically allow people that I communicate with often to chat with me and see when I'm online" to the second option of "Only allow people that I've explicitly approved to chat with me and see when I'm online."

IPv6 is currently not supported. Use of IPv4 is required.

Google Voice can now be used with Google Apps accounts.

Gtalk configuration

The chan_gtalk Channel Driver is configured with the gtalk.conf configuration file, typically located in /etc/asterisk. What follows is the minimum required configuration for successful operation.

Minimum Gtalk Configuration

```
[general]
context=local
allowguests=yes
bindaddr=0.0.0.0
externip=216.208.246.1

[guest]
disallow=all
allow=ulaw
context=local
connection=asterisk
```

This general section of this configuration specifies several items.

1. That calls will terminate to or originate from the **local** context; context=local
2. That guest calls are allowd; allowguests=yes
3. That RTP sessions will be bound to a local address (an IPv4 address must be present); bindaddr=0.0.0.0
4. (optional) That your external (the one outside of your NAT) IP address is defined; externip=216.208.246.1

The guest section of this configuration specifies several items.

1. That no codecs are allowed; disallow=all
2. That the ulaw codec is explicitly allowed; allow=ulaw

3. That calls received by the guest user will be terminated into the **local** context; context=local
4. That the Jabber connection used for guest calls is called "asterisk;" connection=asterisk

Jabber Configuration

The res_jabber Resource is configured with the jabber.conf configuration file, typically located in /etc/asterisk. What follows is the minimum required configuration for successful operation.

Minimum Jabber Configuration

```
[general]
autoregister=yes

[asterisk]
type=client
serverhost=talk.google.com
username=your_google_username@gmail.com/Talk
secret=your_google_password
port=5222
usetls=yes
usesasl=yes
statusmessage="I am an Asterisk Server"
timeout=100
```

The general section of this configuration specifies several items.

1. Debug mode is enabled, so that XMPP messages can be seen on the Asterisk CLI; debug=yes
2. Automated buddy pruning is disabled, otherwise buddies will be automatically removed from your list; autoprune=no
3. Automated registration of users from the buddy list is enabled; autoregister=yes

The asterisk section of this configuration specifies several items.

1. The type is set to client, as we're connecting to Google as a service; type=client
2. The serverhost is Google's talk server; serverhost=talk.google.com
3. Our username is configured as your_google_username@gmail.com/resource, where our resource is "Talk;" username=your_google_username@gmail.com/Talk
4. Our password is configured using the secret option; secret=your_google_password
5. Google's talk service operates on port 5222; port=5222
6. TLS encryption is required by Google; usetls=yes
7. Simple Authentication and Security Layer (SASL) is used by Google; usesasl=yes
8. We set a status message so other Google chat users can see that we're an Asterisk server; statusmessage="I am an Asterisk Server"
9. We set a timeout for receiving message from Google that allows for plenty of time in the event of network delay; timeout=100

Phone configuration

Now, let's place a phone into the same context as the Google calls. The configuration of a SIP device for this purpose would, in sip.conf, typically located in /etc/asterisk, look something like:

```
[malcolm]
type=peer
secret=my_secure_password
host=dynamic
context=local
```

Dialplan configuration

Incoming calls

Next, let's configure our dialplan to receive an incoming call from Google and route it to the SIP phone we created. To do this, our dialplan, extensions.conf, typically located in /etc/asterisk, would look like:

```
exten => s,1,Answer()
exten => s,n,Wait(2)
exten => s,n,SendDTMF(1)
exten => s,n,Dial(SIP/malcolm,20)
```

> ⚠ Note that you might have to adjust the "Wait" time from 2 (in seconds) to something larger, like 8, depending on the current mood of Google. Otherwise, your incoming calls might not be successfully picked up.

This example uses the "s" unmatched extension, because Google does not forward any DID when it sends the call to Asterisk.

In this example, we're Answering the call, Waiting 2 seconds, sending the DTMF "1" back to Google, and **then** dialing the call.
We do this, because inbound calls from Google enable, even if it's disabled in your Google Voice control panel, call screening.
Without this SendDTMF event, you'll have to confirm with Google whether or not you want to answer the call.

> ✔ **Using Google's voicemail**
> Another method for accomplishing the sending of the DTMF event is to use the D dial option. The D option tells Asterisk to send a specified DTMF string after the called party has answered. DTMF events specified before a colon are sent to the **called** party. DTMF events specified after a colon are sent to the **calling** party.
>
> In this example then, one does not need to actually answer the call first. This means that if the called party doesn't answer, Google will resort to sending the call to one's Google Voice voicemail box, instead of leaving it at Asterisk.
>
> ```
> exten => s,1,Dial(SIP/malcolm,20,D(:1))
> ```

```
+15555551212@voice.google.com/srvres-MTAuMjE4LjIuMTk3Ojk4MzM=
```

Your VoIP client (SIPDroid) might not like this, so let's simplify that Caller ID a bit, and make it more presentable for your phone's display. Here's the example that we'll step through:

```
exten => s,1,Set(crazygooglecid=${CALLERID(name)})
exten => s,n,Set(stripcrazysuffix=${CUT(crazygooglecid,@,1)})
exten => s,n,Set(CALLERID(all)=${stripcrazysuffix})
exten => s,n,Dial(SIP/malcolm,20,D(:1))
```

First, we set a variable called **crazygooglecid** to be equal to the name field of the CALLERID function. Next, we use the CUT function to grab everything that's before the @ symbol, and save it in a new variable called **stripcrazysuffix.** We'll set this new variable to the CALLERID that we're going to use for our Dial. Finally, we'll actually Dial our internal destination.

Outgoing calls

Outgoing calls to Google Talk users take the form of:

```
exten => 100,1,Dial(gtalk/asterisk/mybuddy@gmail.com)
```

Where the technology is "gtalk," the dialing peer is "asterisk" as defined in jabber.conf, and the dial string is the Google account name.

Outgoing calls made to Google Voice take the form of:

```
exten =>
_1XXXXXXXXXX,1,Dial(gtalk/asterisk/+${EXTEN}@voice.google.com)
```

Where the technology is "gtalk," the dialing peer is "asterisk" as defined in jabber.conf, and the dial string is a full E.164 number (plus character followed by country code, followed by the rest of the digits).

Interactive Voice with Text Response (IVTR)

Because the Google Talk web client provides both audio and text interface, you can use it to provide a text-based way of traversing Interactive Voice Response (IVR) menus. This is necessary since the client does not have any DTMF inputs.

In the following dialplan example, we will answer the call, wait a bit, send some text that will show

up in the caller's Google Talk client, play back a prompt, capture the caller's text-based response, and then dial the appropriate SIP device.

```
exten => s,1,Answer()
exten => s,n,SendText("If you know the extension of the party you
wish to reach, dial it now.")
exten => s,n,Background(if-u-know-ext-dial)
exten =>
s,n,Set(OPTION=${JABBER_RECEIVE(asterisk,${CALLERID(name)::15},5)})
exten => s,n,Dial(SIP/${OPTION},20)
```

Note that with the JABBER_RECEIVE function, we're receiving the text from **asterisk** which we defined earlier in this page as our connection to Google. We're also specifying with **${CALLERID(name)::15}** that we want to strip off the last 15 characters from the CallerID name string - which is the number of characters that Google is appending, as of this writing, to represent an internal call ID number, and that we want to wait **5** seconds for a response.

Webinar

A Webinar was conducted on Tuesday, March 24, 2011 detailing Asterisk configuration for calling using Google as well as several usage cases. A copy of the slides, in PDF format, is available here - Google Calling Webinar - Public.pdf

Channel Event Logging (CEL)

Top-level page for all things CEL

CEL Design Goals

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

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

The design philosophy of CEL is to generate event data that can grouped together to form a

billing record. This may not be a simple task, but we hope to provide a few different examples that could be used as a basis for those involved in this effort.

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

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

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

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

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

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

```
"EV_CHAN_START","2007-05-09
12:46:16","fxs.52","152","","","","s","extension","Zap/52-1","","","D(
;;; 152 takes the phone off-hook
"EV_APP_START","2007-05-09
12:46:18","fxs.52","152","152","","","151","extension","Zap/52-1","Dia
;;; 152 finishes dialing 151
"EV_CHAN_START","2007-05-09
12:46:18","fxs.51","151","","","","s","extension","Zap/51-1","","","D(
;;; 151 channel created, starts ringing
(151 is ringing)
"EV_ANSWER","2007-05-09
12:46:19","","151","152","","","151","extension","Zap/51-1","AppDial",
Line)","DOCUMENTATION","","1178736378.4","","" ;;; 151 answers
"EV_ANSWER","2007-05-09
12:46:19","fxs.52","152","152","","","151","extension","Zap/52-1","Dia
;;; so does 152 (???)
"EV_BRIDGE_START","2007-05-09
12:46:20","fxs.52","152","152","","","151","extension","Zap/52-1","Dia
;;; 152 and 151 are bridged
(151 and 152 are conversing)
"EV_BRIDGE_END","2007-05-09
12:46:25","fxs.52","152","152","","","151","extension","Zap/52-1","Dia
;;; after 5 seconds, the bridge ends (152 dials #700?)
```

```
"EV_BRIDGE_START","2007-05-09
12:46:25","fxs.52","152","152","","","151","extension","Zap/52-1","Dia
;;; extraneous 0-second bridge?
"EV_BRIDGE_END","2007-05-09
12:46:25","fxs.52","152","152","","","151","extension","Zap/52-1","Dia
;;;
"EV_PARK_START","2007-05-09
12:46:27","","151","152","","","","extension","Zap/51-1","Parked
Call","","DOCUMENTATION","","1178736378.4","","" ;;; 151 is parked
"EV_HANGUP","2007-05-09
12:46:29","fxs.52","152","152","","","h","extension","Zap/52-1","","",
,"","" ;;; 152 hangs up 2 sec later
"EV_CHAN_END","2007-05-09
12:46:29","fxs.52","152","152","","","h","extension","Zap/52-1","","",
;;; 152's channel goes away
(151 is parked and listening to MOH! now, 150 picks up, and dials
701)
"EV_CHAN_START","2007-05-09
12:47:08","fxs.50","150","","","","s","extension","Zap/50-1","","","DC
;;; 150 picks up the phone, dials 701
"EV_PARK_END","2007-05-09
12:47:11","","151","152","","","","extension","Zap/51-1","Parked
Call","","DOCUMENTATION","","1178736378.4","","" ;;; 151's park
comes to end
"EV_ANSWER","2007-05-09
12:47:11","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;; 150 gets answer (twice)
"EV_ANSWER","2007-05-09
12:47:12","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;;
"EV_BRIDGE_START","2007-05-09
12:47:12","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;; bridge begins between 150 and recently parked 151 (150 and 151
are conversing, then 151 hits flash)
"EV_CHAN_START","2007-05-09
12:47:51","fxs.51","151","","","","s","extension","Zap/51-2","","","DC
;;; 39 seconds later, 51-2 channel is created. (151 flashes hook)
"EV_HOOKFLASH","2007-05-09
12:47:51","","151","152","","","","extension","Zap/51-1","Bridged
Call","Zap/50-1","DOCUMENTATION","","1178736378.4","","Zap/51-2" ;;;
a marker to record that 151 flashed the hook
"EV_BRIDGE_END","2007-05-09
12:47:51","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;; bridge ends between 150 and 151
"EV_BRIDGE_START","2007-05-09
12:47:51","fxs.50","150","150","","","701","extension","Zap/50-1","Par
```

```
;;; 0-second bridge from 150 to ? 150 gets no sound at all
"EV_BRIDGE_END","2007-05-09
12:47:51","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;;
"EV_BRIDGE_START","2007-05-09
12:47:51","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;; bridge start on 150
(151 has dialtone after hitting flash; dials 152)
"EV_APP_START","2007-05-09
12:47:55","fxs.51","151","151","","","152","extension","Zap/51-2","Dia
;;; 151-2 dials 152 after 4 seconds
"EV_CHAN_START","2007-05-09
12:47:55","fxs.52","152","","","","s","extension","Zap/52-1","","","DC
,"","" ;;; 152 channel created to ring 152.
(152 ringing)
"EV_ANSWER","2007-05-09
12:47:58","","152","151","","","152","extension","Zap/52-1","AppDial",
Line)","DOCUMENTATION","","1178736475.7","","" ;;; 3 seconds later,
152 answers
"EV_ANSWER","2007-05-09
12:47:58","fxs.51","151","151","","","152","extension","Zap/51-2","Dia
;;; ... and 151-2 also answers
"EV_BRIDGE_START","2007-05-09
12:47:59","fxs.51","151","151","","","152","extension","Zap/51-2","Dia
;;; 1 second later, bridge formed betw. 151-2 and 151 (152 answers,
151 and 152 convering; 150 is listening to silence; 151 hits flash
again... to start a 3way)
"EV_3WAY_START","2007-05-09
12:48:58","","151","152","","","","extension","Zap/51-1","Bridged
Call","Zap/50-1","DOCUMENTATION","","1178736378.4","","Zap/51-2" ;;;
another hook-flash to begin a 3-way conference
"EV_BRIDGE_END","2007-05-09
12:48:58","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;; - almost 1 minute later, the bridge ends (151 flashes hook
again)
"EV_BRIDGE_START","2007-05-09
12:48:58","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;; 0-second bridge at 150. (3 way conf formed)
"EV_BRIDGE_END","2007-05-09
12:48:58","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;;
"EV_BRIDGE_START","2007-05-09
12:48:58","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;; bridge starts for 150
(3way now, then 151 hangs up.)
"EV_BRIDGE_END","2007-05-09
```

```
12:49:26","fxs.50","150","150","","","701","extension","Zap/50-1","Par
;;; 28 seconds later, bridge ends
"EV_HANGUP","2007-05-09
12:49:26","","151","152","","","","extension","Zap/51-1","Bridged
Call","Zap/50-1","DOCUMENTATION","","1178736378.4","","" ;;; 151
hangs up, leaves 150 and 152 connected
"EV_CHAN_END","2007-05-09
12:49:26","","151","152","","","","extension","Zap/51-1","Bridged
Call","Zap/50-1","DOCUMENTATION","","1178736378.4","","" ;;; 151
channel ends
"EV_CHAN_END","2007-05-09
12:49:26","fxs.51","151","151","","","h","extension","Zap/51-2ZOMBIE",
;;; 152-2 channel ends (zombie)
(just 150 and 152 now)
"EV_BRIDGE_END","2007-05-09
12:50:13","fxs.50","150","150","","","152","extension","Zap/50-1","Dia
;;; 47 sec later, the bridge from 150 to 152 ends
"EV_HANGUP","2007-05-09
12:50:13","","152","151","","","","extension","Zap/52-1","Bridged
Call","Zap/50-1","DOCUMENTATION","","1178736475.7","","" ;;; 152
hangs up
"EV_CHAN_END","2007-05-09
12:50:13","","152","151","","","","extension","Zap/52-1","Bridged
Call","Zap/50-1","DOCUMENTATION","","1178736475.7","","" ;;; 152
channel ends
"EV_HANGUP","2007-05-09
12:50:13","fxs.50","150","150","","","h","extension","Zap/50-1","","",
;;; 150 hangs up
"EV_CHAN_END","2007-05-09
```

```
12:50:13","fxs.50","150","150","","","h","extension","Zap/50-1","","",
;;; 150 ends
```

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

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

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

Event: Newstate
Privilege: call,all
Channel: Zap/52-1
State: Ring
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801102.5
Event: Newexten
Privilege: call,all
Channel: Zap/52-1
Context: extension
Extension: 151
Priority: 1
Application: Set
AppData: CDR(myvar)=zingo
Uniqueid: 1178801102.5
Event: Newexten
```

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

Event: Newchannel
Privilege: call,all
Channel: Zap/51-1
State: Rsrvd
CallerIDNum: 151
CallerIDName: fxs.51
Uniqueid: 1178801108.6
Event: Newstate
Privilege: call,all
Channel: Zap/51-1
State: Ringing
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801108.6

Event: Dial
Privilege: call,all
SubEvent: Begin
Source: Zap/52-1
Destination: Zap/51-1
CallerIDNum: 152
CallerIDName: fxs.52
SrcUniqueID: 1178801102.5
DestUniqueID: 1178801108.6
Event: Newcallerid
Privilege: call,all
Channel: Zap/51-1
CallerIDNum: 151
CallerIDName: <Unknown>
Uniqueid: 1178801108.6
CID-CallingPres: 0 (Presentation Allowed, Not Screened)

Event: Newstate
Privilege: call,all
Channel: Zap/52-1
State: Ringing
CallerIDNum: 152
```

```
CallerIDName: fxs.52
Uniqueid: 1178801102.5
Event: Newstate
Privilege: call,all
Channel: Zap/51-1
State: Up
CallerIDNum: 151
CallerIDName: <unknown>
Uniqueid: 1178801108.6
Event: Newstate
Privilege: call,all
Channel: Zap/52-1
State: Up
CallerIDNum: 152
CallerIDName: fxs.52
Uniqueid: 1178801102.5

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

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

```
Uniqueid1: 1178801102.5
Uniqueid2: 1178801108.6
CallerID1: 152
CallerID2: 151

Event: ParkedCall
Privilege: call,all
Exten: 701
Channel: Zap/51-1
From: Zap/52-1
Timeout: 45
CallerIDNum: 151
CallerIDName: <unknown>
Event: Dial
Privilege: call,all
SubEvent: End
Channel: Zap/52-1
DialStatus: ANSWER

Event: Newexten
Privilege: call,all
Channel: Zap/52-1
Context: extension
Extension: h
Priority: 1
Application: Goto
AppData: label1
Uniqueid: 1178801102.5
Event: Newexten
Privilege: call,all
Channel: Zap/52-1
Context: extension
Extension: h
Priority: 4
Application: Goto
AppData: label2
Uniqueid: 1178801102.5

Event: Newexten
Privilege: call,all
Channel: Zap/52-1
Context: extension
Extension: h
Priority: 2
Application: NoOp
AppData: In Hangup! myvar is zingo and accountcode is billsec is 26
```

```
and duration is 40 and end is 2007-05-10 06:45:42.
Uniqueid: 1178801102.5
Event: Newexten
Privilege: call,all
Channel: Zap/52-1
Context: extension
Extension: h
Priority: 3
Application: Goto
AppData: label3
Uniqueid: 1178801102.5

Event: Newexten
Privilege: call,all
Channel: Zap/52-1
Context: extension
Extension: h
Priority: 5
Application: NoOp
AppData: More Hangup message after hopping around"
Uniqueid: 1178801102.5
Event: Hangup
Privilege: call,all
Channel: Zap/52-1
Uniqueid: 1178801102.5
Cause: 16
Cause-txt: Normal Clearing

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

Event: Newcallerid
Privilege: call,all
Channel: Zap/50-1
```

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

Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: 701
Priority: 1
Application: ParkedCall
AppData: 701
Uniqueid: 1178801162.7
Event: UnParkedCall
Privilege: call,all
Exten: 701
Channel: Zap/51-1
From: Zap/50-1
CallerIDNum: 151
CallerIDName: <unknown>
Event: Newstate
Privilege: call,all
Channel: Zap/50-1
State: Up
CallerIDNum: 150
CallerIDName: fxs.50
Uniqueid: 1178801162.7

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

```
Channel: Zap/51-2
State: Rsrvd
CallerIDNum: 151
CallerIDName: fxs.51
Uniqueid: 1178801218.8

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

Event: Unlink
Privilege: call,all
Channel1: Zap/50-1
Channel2: Zap/51-1
Uniqueid1: 1178801162.7
Uniqueid2: 1178801108.6
CallerID1: 150
CallerID2: 151
Event: Link
Privilege: call,all
Channel1: Zap/50-1
Channel2: Zap/51-1
Uniqueid1: 1178801162.7
Uniqueid2: 1178801108.6
CallerID1: 150
CallerID2: 151
Event: Newcallerid
Privilege: call,all
Channel: Zap/51-2
CallerIDNum: 151
CallerIDName: fxs.51
Uniqueid: 1178801218.8
CID-CallingPres: 0 (Presentation Allowed, Not Screened)
```

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

Event: Newexten
Privilege: call,all
Channel: Zap/51-2
Context: extension
Extension: 152
Priority: 1
Application: Set
AppData: CDR(myvar)=zingo
Uniqueid: 1178801218.8
Event: Newexten
Privilege: call,all
Channel: Zap/51-2
Context: extension
Extension: 152
Priority: 2
Application: Dial
AppData: Zap/52|30|TtWw
Uniqueid: 1178801218.8

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

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

Event: Newcallerid
Privilege: call,all
Channel: Zap/52-1
CallerIDNum: 152
CallerIDName: <Unknown>
Uniqueid: 1178801223.9
CID-CallingPres: 0 (Presentation Allowed, Not Screened)
Event: Newstate
Privilege: call,all
Channel: Zap/51-2
State: Ringing
CallerIDNum: 151
CallerIDName: fxs.51
Uniqueid: 1178801218.8

Event: Newstate
Privilege: call,all
Channel: Zap/52-1
State: Up
CallerIDNum: 152
CallerIDName: <unknown>
Uniqueid: 1178801223.9
Event: Newstate
Privilege: call,all
Channel: Zap/51-2
State: Up
CallerIDNum: 151
CallerIDName: fxs.51
Uniqueid: 1178801218.8

Event: Link
Privilege: call,all
Channel1: Zap/51-2
```

```
Channel2: Zap/52-1
Uniqueid1: 1178801218.8
Uniqueid2: 1178801223.9
CallerID1: 151
CallerID2: 152
Event: Unlink
Privilege: call,all
Channel1: Zap/50-1
Channel2: Zap/51-1
Uniqueid1: 1178801162.7
Uniqueid2: 1178801108.6
CallerID1: 150
CallerID2: 151

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

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

```
CallerID2: 151

Event: Hangup
Privilege: call,all
Channel: Zap/51-1
Uniqueid: 1178801108.6
Cause: 16
Cause-txt: Normal
Clearing
Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 1
Application: Goto
AppData: label1
Uniqueid: 1178801162.7
Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 4
Application: Goto
AppData: label2
Uniqueid: 1178801162.7

Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 2
Application: NoOp
AppData: In Hangup! myvar is and accountcode is billsec is 0 and
duration is 0 and end is 2007-05-10 06:48:37.
Uniqueid: 1178801162.7
Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 3
Application: Goto
AppData: label3
```

```
Uniqueid: 1178801162.7

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

Event: Masquerade
Privilege: call,all
Clone: Zap/50-1
CloneState: Up
Original: Zap/51-2
OriginalState: Up
Event: Rename
Privilege: call,all
Oldname: Zap/50-1
Newname: Zap/50-1<MASQ>
Uniqueid: 1178801162.7

Event: Rename
Privilege: call,all
 Oldname: Zap/51-2
Newname: Zap/50-1
Uniqueid: 1178801218.8
Event: Rename
Privilege: call,all
Oldname: Zap/50-1<MASQ>
Newname: Zap/51-2<ZOMBIE>
Uniqueid: 1178801162.7
Event: Hangup
Privilege: call,all
Channel: Zap/51-2<ZOMBIE>
 Uniqueid: 1178801162.7
Cause: 0
Cause-txt: Unknown

Event: Unlink
Privilege: call,all
Channel1: Zap/50-1
Channel2: Zap/52-1
```

```
Uniqueid1: 1178801218.8
Uniqueid2: 1178801223.9
CallerID1: 150
CallerID2: 152
Event: Hangup
Privilege: call,all
Channel: Zap/52-1
Uniqueid: 1178801223.9
Cause: 16
Cause-txt: Normal Clearing

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

Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 4
Application: Goto
AppData: label2
Uniqueid: 1178801218.8
Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 2
Application: NoOp
AppData: In Hangup! myvar is and accountcode is billsec is 90 and
duration is 94 and end is 2007-05-10 06:48:37.
Uniqueid: 1178801218.8
```

```
Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 3
Application: Goto
AppData: label3
Uniqueid: 1178801218.8
Event: Newexten
Privilege: call,all
Channel: Zap/50-1
Context: extension
Extension: h
Priority: 5
Application: NoOp
AppData: More Hangup message after hopping around"
Uniqueid: 1178801218.8
Event: Hangup
Privilege: call,all
Channel: Zap/50-1
Uniqueid: 1178801218.8
```

```
Cause: 16
Cause-txt: Normal Clearing
```

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

```
""fxs.52"
152","152","h","extension","Zap/52-1","Zap/51-1","NoOp","More Hangup
message after hopping around"","2007-05-09 17:35:56","2007-05-09
17:36:20","2007-05-09
17:36:36","40","16","ANSWERED","DOCUMENTATION","","1178753756.0",""
""fxs.50"
150","150","152","extension","Zap/50-1","Zap/51-1","NoOp","More
Hangup message after hopping around"","2007-05-09
17:37:59","2007-05-09 17:38:06","2007-05-09
17:39:11","72","65","ANSWERED","DOCUMENTATION","","1178753871.3",""
```

CEL Events and Fields

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

Table of CEL Events

Event	Description
CHAN_START	The time a channel was created
CHAN_END	The time a channel was terminated
ANSWER	The time a channel was answered (ie, phone taken off-hook, etc)
HANGUP	The time at which a hangup occurred.
CONF_ENTER	The time a channel was connected into a conference room
CONF_EXIT	The time a channel was removed from a conference room
CONF_START	The time the first person enters a conference
CONF_END	The time the last person left a conf (and turned out the lights?)
APP_START	The time a tracked application was started
APP_END	the time a tracked application ended
PARK_START	The time a call was parked
PARK_END	unpark event
BRIDGE_START	The time a bridge is started
BRIDGE_END	The time a bridge is ended
3WAY_START	When a 3-way conf starts (usually via attended xfer)
3WAY_END	When one or all exit a 3-way conf
BLINDTRANSFER	When a blind transfer is initiated

ATTENDEDTRANSFER	When an attended transfer is initiated
TRANSFER	Generic transfer initiated; not used yet...?
HOOKFLASH	So far, when a hookflash event occurs on a Zap interface
USER_EVENT	these are triggered from the dialplan, and have a name given by the user.

Table of CEL Event Fields

Table 11.2: List of CEL Event Fields

Field	Description
eventtype	The name of the event; see the above list; each is prefixed with "EV_".
eventtime	The time the event happened
cidname	CID name field
cidnum	CID number field
cidani	CID ANI field
cidrdnis	CID RDNIS field
ciddnid	CID DNID field
exten	The extension in the dialplan
context	The context in the dialplan
channame	The name assigned to the channel in which the event took place
appname	The name of the current application
appdata	The arguments that will be handed to that application
amaflags	The AMA flags associated with the event; user assignable.
accountcode	A user assigned datum (string)
uniqueid	Each Channel instance gets a unique ID associated with it.
userfield	A user assigned datum (string)
linkedid	the per-call id, spans several events, possibly.
peer	For bridge or other 2-channel events, this would be the other channel name

CEL Applications and Functions

Top-level page for information on CEL Applications and Functions

CEL Function

- • • • • THIS IS NO LONGER TRUE REWRITE *****

The CEL function parallels the CDR function, for fetching values from the channel or event. It has some notable notable differences, though! For instance, CEL data is not stored on the channel. Well, not much of it, anyway! You can use the CEL function to set the amaflags, accountcode, and userfield, which are stored on the channel.

Channel variables are not available for reading from the CEL function, nor can any variable name other than what's in the list, be set. CDRs have a structure attached to the channel, where the CDR function could access the values stored there, or set the values there. CDRs could store

their own variable lists, but CEL has no such storage. There is no reason to store any event information, as they are immediately output to the various backends at the time they are generated.

See the description for the CEL function from the CLI: core show function CEL

Here is a list of all the available channel field names:

- cidname
- userfield
- cidnum
- amaflags
- cidani
- cidrdnis
- ciddnid
- appdata
- exten
- accountcode
- context
- uniqueid
- channame
- appname
- peer
- eventtime
- eventtype

CELGenUserEvent Application

This application allows the dialplan to insert custom events into the event stream.
For more information, in the CLI, type: **core show application CELGenUserEvent**
Its arguments take this format:

```
CELGenUserEvent(eventname)
```

Please note that there is no restrictions on the name supplied. If it happens to match a standard CEL event name, it will look like that event was generated. This could be a blessing or a curse!

CEL Configuration Files

cel.conf

Generating Billing Information from CEL

-
-
-
- This is the Next Big Task ****

CEL Storage Backends

Right now, the CEL package will support CSV, Customized CSV, ODBC, PGSQL, TDS, Sqlite3, and Radius back ends. See the doc/celdriver.tex file for how to use these back ends.

MSSQL CEL Backend

Asterisk can currently store Channel Events into an MSSQL database in two different ways: cel_odbc or cel_tds

Channel Event Records can be stored using unixODBC (which requires the FreeTDS package) [cel_odbc] or directly by using just the FreeTDS package [cel_tds]

The following provide some examples known to get asterisk working with mssql.

> ⚠ Only choose one db connector.

ODBC using cel_odbc

Compile, configure, and install the latest unixODBC package:

```
tar -zxvf unixODBC-2.2.9.tar.gz && cd unixODBC-2.2.9 && ./configure
--sysconfdir=/etc --prefix=/usr --disable-gui && make && make
install
```

Compile, configure, and install the latest FreeTDS package:

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

Compile, or recompile, asterisk so that it will now add support for cel_odbc.

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

Setup odbc configuration files.

These are working examples from my system. You will need to modify for your setup. You are not required to store usernames or passwords here.

/etc/odbcinst.ini

```
[FreeTDS]
Description = FreeTDS ODBC driver for MSSQL
Driver = /usr/lib/libtdsodbc.so
Setup = /usr/lib/libtdsS.so
FileUsage = 1
```

/etc/odbc.ini

```
[MSSQL-asterisk]
description = Asterisk ODBC for MSSQL
driver = FreeTDS
server = 192.168.1.25
port = 1433
database = voipdb
tds_version = 7.0
language = us_english
```

⛔ Only install one database connector. Do not confuse asterisk by using both ODBC (cel_odbc) and FreeTDS (cel_tds). This command will erase the contents of cel_tds.conf

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

⚠ unixODBC requires the freeTDS package, but asterisk does not call freeTDS directly.

Now set up cel_odbc configuration files.

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

/etc/asterisk/cel_odbc.conf

```
[global]
dsn=MSSQL-asterisk
username=voipdbuser
password=voipdbpass
loguniqueid=yes
```

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

```
CREATE TABLE cel (
        [eventtype] [varchar] (30) NOT NULL ,
        [eventtime] [datetime] NOT NULL ,
        [cidname] [varchar] (80) NOT NULL ,
        [cidnum] [varchar] (80) NOT NULL ,
        [cidani] [varchar] (80) NOT NULL ,
        [cidrdnis] [varchar] (80) NOT NULL ,
        [ciddnid] [varchar] (80) NOT NULL ,
        [exten] [varchar] (80) NOT NULL ,
        [context] [varchar] (80) NOT NULL ,
        [channame] [varchar] (80) NOT NULL ,
        [appname] [varchar] (80) NOT NULL ,
        [appdata] [varchar] (80) NOT NULL ,
        [amaflags] [int] NOT NULL ,
        [accountcode] [varchar] (20) NOT NULL ,
        [uniqueid] [varchar] (32) NOT NULL ,
        [peer] [varchar] (80) NOT NULL ,
        [userfield] [varchar] (255) NOT NULL
) ;
```

Start asterisk in verbose mode, you should see that asterisk logs a connection to the database and will now record every desired channel event at the moment it occurs.

FreeTDS, using cel_tds

Compile, configure, and install the latest FreeTDS package:

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

Compile, or recompile, asterisk so that it will now add support for cel_tds.

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

🚫 Only install one database connector. Do not confuse asterisk by using both ODBC (cel_odbc) and FreeTDS (cel_tds). This command will erase the contents of cel_odbc.conf

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

Setup cel_tds configuration files.

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

/etc/asterisk/cel_tds.conf

```
[global]
hostname=192.168.1.25
port=1433
dbname=voipdb
user=voipdbuser
password=voipdpass
charset=BIG5
```

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

```
CREATE TABLE cel (
        [eventtype] [varchar] (30) NULL ,
        [eventtime] [datetime] NULL ,
        [cidname] [varchar] (80) NULL ,
        [cidnum] [varchar] (80) NULL ,
        [cidani] [varchar] (80) NULL ,
        [cidrdnis] [varchar] (80) NULL ,
        [ciddnid] [varchar] (80) NULL ,
        [exten] [varchar] (80) NULL ,
        [context] [varchar] (80) NULL ,
        [channame] [varchar] (80) NULL ,
        [appname] [varchar] (80) NULL ,
        [appdata] [varchar] (80) NULL ,
        [amaflags] [varchar] (16) NULL ,
        [accountcode] [varchar] (20) NULL ,
        [uniqueid] [varchar] (32) NULL ,
        [userfield] [varchar] (255) NULL ,
        [peer] [varchar] (80) NULL
) ;
```

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

MySQL CEL Backend

Using MySQL for Channel Event records is supported by using ODBC and the cel_odbc module.

PostgreSQL CEL Backend

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

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

Here is a sample:

/etc/asterisk/cel_pgsql.conf

```
; Sample Asterisk config file for CEL logging to PostgresSQL
[global]
hostname=localhost
port=5432
dbname=asterisk
password=password
user=postgres
table=cel
```

Now create a table in postgresql for your cels

```
CREATE TABLE cel (
        id serial ,
        eventtype varchar (30) NOT NULL ,
        eventtime timestamp NOT NULL ,
        userdeftype varchar(255) NOT NULL ,
        cid_name varchar (80) NOT NULL ,
        cid_num varchar (80) NOT NULL ,
        cid_ani varchar (80) NOT NULL ,
        cid_rdnis varchar (80) NOT NULL ,
        cid_dnid varchar (80) NOT NULL ,
        exten varchar (80) NOT NULL ,
        context varchar (80) NOT NULL ,
        channame varchar (80) NOT NULL ,
        appname varchar (80) NOT NULL ,
        appdata varchar (80) NOT NULL ,
        amaflags int NOT NULL ,
        accountcode varchar (20) NOT NULL ,
        peeraccount varchar (20) NOT NULL ,
        uniqueid varchar (150) NOT NULL ,
        linkedid varchar (150) NOT NULL ,
        userfield varchar (255) NOT NULL ,
        peer varchar (80) NOT NULL
);
```

SQLite 3 CEL Backend

SQLite version 3 is supported in cel_sqlite3_custom.

RADIUS CEL Backend

What is needed

- FreeRADIUS server

- Radiusclient-ng library
- Asterisk PBX

Installation of the Radiusclient library

Download the sources

From http://developer.berlios.de/projects/radiusclient-ng/

Untar the source tarball:

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

Compile and install the library:

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

Configuration of the Radiusclient library

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

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

```
authserver localhost
```

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

```
acctserver localhost
```

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

File "servers"

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

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

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

file.

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

```
File "dictionary"
```

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

file /usr/local/etc/radiusclient-ng/dictionary the line:

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

Install FreeRADIUS Server (Version 1.1.1)

Download sources tarball from:

http://freeradius.org/

Untar, configure, build, and install the server:

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

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

Configuration of the FreeRADIUS Server

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

File "clients.conf"

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

Example:

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

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

File "dictionary"

> ⚠️ As of version 1.1.2, the dictionary.digium file ships with FreeRADIUS.

The following procedure brings the dictionary.digium file to previous versions of FreeRADIUS.

File /usr/local/etc/raddb/dictionary contains the dictionary of FreeRADIUS server. You have to add the same dictionary file (dictionary.digium), which you added to the dictionary of radiusclient-ng library. You can include it into the main file, adding the following line at the end of file /usr/local/etc/raddb/dictionary:

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

That will include the same new attribute definitions that are used in radiusclient-ng library so the client and server will understand each other.

Asterisk Accounting Configuration

Compilation and installation:

The module will be compiled as long as the radiusclient-ng library has been detected on your system.

By default FreeRADIUS server will log all accounting requests into /usr/local/var/log/radius/radacct directory in form of plain text files. The server will create one file for each hostname in the directory. The following example shows how the log files look like.

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

The configuration file for cel_radius.so module is :
/etc/asterisk/cel.conf This is where you can set CEL related parameters as well as the path to the radiusclient-ng library configuration file.

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

Logged Values

- "Asterisk-Acc-Code", The account name of detail records
- "Asterisk-CidName",
- "Asterisk-CidNum",
- "Asterisk-Cidani",
- "Asterisk-Cidrdnis",
- "Asterisk-Ciddnid",

- "Asterisk-Exten",
- "Asterisk-Context", The destination context
- "Asterisk-Channame", The channel name
- "Asterisk-Appname", Last application run on the channel
- "Asterisk-App-Data", Argument to the last channel
- "Asterisk-Event-Time",
- "Asterisk-Event-Type",
- "Asterisk-AMA-Flags", DOCUMENTATION, BILL, IGNORE etc, specified on a per channel basis like accountcode.
- "Asterisk-Unique-ID", Unique call identifier
- "Asterisk-User-Field" User field set via SetCELUserField
- "Asterisk-Peer" Name of the Peer for 2-channel events (like bridge)

Channel Variables

What's a channel variable? Read on to find out why they're important and how they'll improve your quality of life.

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

1. The first, and most frequently used, is the substitution of variable references with their values.
2. Then there are the evaluations of expressions done in $[..]. This will be discussed below.

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

Parameter Quoting

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

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

Also, characters special to variable substitution, expression evaluation, etc (see below), can be quoted. For example, to literally use a $ on the string "$1231", quote it with a preceding
. Special characters that must be quoted to be used, are [ ] $ " \. (to write \itself, use a backslash.
).

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

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

About Variables

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

To set a variable to a particular value, do:

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

You can substitute the value of a variable everywhere using ${variablename}. For example, to

stringwise append $lala to $blabla and store result in $koko, do:

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

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

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

stores to the variable "koko" the value "lala" and to variable "lala" the value "blabla".
In fact, everything contained ${here} is just replaced with the value of the variable "here".

Variable Inheritance

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

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

Variable Inheritance Examples

```
Set(__FOO=bar)
```

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

However, NoOp(${__FOO}) is identical to NoOp(${FOO})

Selecting Characters from Variables

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

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

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

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

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

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

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

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

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

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

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

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

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

Expressions

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

For example, after the sequence:

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

the value of variable koko is "6".

and, further:

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

will parse as intended. Extra spaces are ignored.

Spaces Inside Variables Values

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

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

As an example:

```
exten => s,6,GotoIf($[ "${CALLERID(name)}" : "Privacy Manager"
]?callerid-liar,s,1:s,7)
```

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

- "DELOREAN MOTORS" : "Privacy Manager"

and will evaluate to 0.

The above without double quotes would have evaluated to:

- DELOREAN MOTORS : Privacy Manager

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

Operators

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

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

- expr1 & expr2

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

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

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

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

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

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

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

If the match succeeds and the pattern contains at least one regular expression subexpression `', the string corresponding to `\1' is returned; otherwise the matching operator returns the number of characters matched. If the match fails and the pattern contains a regular expression subexpression the null string is returned; otherwise 0.

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

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

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

- expr1 ~~ expr2
  Concatenation operator. The two exprs are evaluated and turned into strings, stripped of surrounding double quotes, and are turned into a single string with no invtervening spaces. This operator is new to trunk after 1.6.0; it is not needed in existing extensions.conf code. Because of the way asterisk evaluates [ ] constructs (recursively, bottom- up), no is ever present when the contents of a [] is evaluated. Thus, tokens are usually already merged at evaluation time. But, in AEL, various exprs are evaluated raw, and [] are gathered and treated as tokens. And in AEL, no two tokens can sit side by side without an intervening operator. So, in AEL, concatenation must be explicitly specified in expressions. This new operator will play well into future plans, where expressions (constructs) are merged into a single grammar.

Parentheses are used for grouping in the usual manner.

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

Floating Point Numbers

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

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

Floating point numbers are turned into strings via the '%g'/'%Lg' format of the printf function set. This allows numbers to still 'look' like integers to those counting on integer behavior. If you were counting on 1/4 evaluating to 0, you need to now say TRUNC(1/4). For a list of all the truncation/rounding capabilities, see the next section.

Functions

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

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

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

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

- COS(x) x is in radians. Results vary from -1 to 1.
- SIN(x) x is in radians. Results vary from -1 to 1.
- TAN(x) x is in radians.
- ACOS(x) x should be a value between -1 and 1.
- ASIN(x) x should be a value between -1 and 1.
- ATAN(x) returns the arc tangent in radians; between -PI/2 and PI/2.
- ATAN2(x,y) returns a result resembling y/x, except that the signs of both args are used to determine the quadrant of the result. Its result is in radians, between -PI and PI.
- POW(x,y) returns the value of x raised to the power of y.
- SQRT(x) returns the square root of x.
- FLOOR(x) rounds x down to the nearest integer.
- CEIL(x) rounds x up to the nearest integer.
- ROUND(x) rounds x to the nearest integer, but round halfway cases away from zero.
- RINT(x) rounds x to the nearest integer, rounding halfway cases to the nearest even integer.
- TRUNC(x) rounds x to the nearest integer not larger in absolute value.
- REMAINDER(x,y) computes the remainder of dividing x by y. The return value is x - n*y, where n is the value x/y, rounded to the nearest integer. If this quotient is 1/2, it is rounded to the nearest even number.
- EXP(x) returns e to the x power.
- EXP2(x) returns 2 to the x power.
- LOG(x) returns the natural logarithm of x.
- LOG2(x) returns the base 2 log of x.
- LOG10(x) returns the base 10 log of x.

Expressions Examples

*"One Thousand Five Hundred" =~ "(T[Expressions Examples^])"

returns: Thousand

- "One Thousand Five Hundred" =~ "T[Expressions Examples^]"
  returns: 8

- "One Thousand Five Hundred" : "T[Expressions Examples^]"
  returns: 0

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

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

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

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

- 2 + 8 / 2
  returns: 6
  (because of operator precedence; the division is done first, then the addition)

- 2+8/2
  returns: 6
  Spaces aren't necessary

- (2+8)/2
  returns: 5
  of course

- (3+8)/2
  returns: 5.5

- TRUNC((3+8)/2)
  returns: 5

- FLOOR(2.5)
  returns: 2

- FLOOR(-2.5)
  returns: -3

- CEIL(2.5)
  returns: 3

- CEIL(-2.5)
  returns: -2

- ROUND(2.5)
  returns: 3

- ROUND(3.5)
  returns: 4

- ROUND(-2.5)
  returns: -3

- RINT(2.5)
  returns: 2

- RINT(3.5)

- RINT(-2.5)
  returns: -2

- RINT(-3.5)
  returns: -4

- TRUNC(2.5)
  returns: 2

- TRUNC(3.5)
  returns: 3

- TRUNC(-3.5)
  returns: -3

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

Numbers Vs. Strings

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

Conditionals

There is one conditional application - the conditional goto :

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

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

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

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

Example of use :

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

Expression Parsing Errors

Syntax errors are now output with 3 lines.

If the extensions.conf file contains a line like:

```
exten => s,6,GotoIf($[ "${CALLERID(num)}" = "3071234567" & &
"${CALLERID(name)}" : "Privacy Manager" ]?callerid-liar,s,1:s,7)
```

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

```
Jul 15 21:27:49 WARNING[1251240752]: ast_yyerror(): syntax error:
parse error, unexpected TOK_AND, expecting TOK_M
INUS or TOK_LP or TOKEN; Input:
 "3072312154" = "3071234567" & & "Steves Extension" : "Privacy
Manager"
                    ^
```

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

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

NULL Strings

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

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

or:

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

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

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

Warnings about Expressions

If you need to do complicated things with strings, asterisk expressions is most likely NOT the best way to go about it. AGI scripts are an excellent option to this need, and make available the

full power of whatever language you desire, be it Perl, C, C++, Cobol, RPG, Java, Snobol, PL/I, Scheme, Common Lisp, Shell scripts, Tcl, Forth, Modula, Pascal, APL, assembler, etc.

Expression Parser Incompatibilities

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

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

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

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

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

1. Tokens separated by space(s). Previously, tokens were separated by spaces. Thus, ' 1 + 1 ' would evaluate to the value '2', but '1+1' would evaluate to the string '1+1'. If this behavior was depended on, then the expression evaluation will break. '1+1' will now evaluate to '2', and something is not going to work right. To keep such strings from being evaluated, simply wrap them in double quotes: ' "1+1" '
2. The colon operator. In versions previous to double quoting, the colon operator takes the right hand string, and using it as a regex pattern, looks for it in the left hand string. It is given an implicit ôperator at the beginning, meaning the pattern will match only at the beginning of the left hand string. If the pattern or the matching string had double quotes around them, these could get in the way of the pattern match. Now, the wrapping double quotes are stripped from both the pattern and the left hand string before applying the pattern. This was done because it recognized that the new way of scanning the expression doesn't use spaces to separate tokens, and the average regex expression is full of operators that the scanner will recognize as expression operators. Thus, unless the pattern is wrapped in double quotes, there will be trouble. For instance, ${VAR1} : (WhoWhat)+ may have have worked before, but unless you wrap the pattern in double quotes now, look out for trouble! This is better: "${VAR1}" : "(WhoWhat*)+" and should work as previous.*
3. Variables and Double Quotes Before these changes, if a variable's value contained one or more double quotes, it was no reason for concern. It is now !
4. LE, GE, NE operators removed. The code supported these operators, but they were not documented. The symbolic operators, =, =, and != should be used instead.
5. Added the unary '-' operator. So you can 3+ -4 and get -1.
6. Added the unary '!' operator, which is a logical complement. Basically, if the string or number is null, empty, or '0', a '1' is returned. Otherwise a '0' is returned.
7. Added the '=~' operator, just in case someone is just looking for match anywhere in the string. The only diff with the ':' is that match doesn't have to be anchored to the beginning of the string.
8. Added the conditional operator 'expr1 ? true_expr :: false_expr' First, all 3 exprs are evaluated, and if expr1 is false, the 'false_expr' is returned as the result. See above for details.
9. Unary operators '-' and '!' were made right associative.

Expression Debugging Hints

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

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

```
make testexpr2
```

in the top level asterisk source directory. This will build a small executable, that is able to take the first command line argument, and run it thru the expression parser. No variable substitutions will be performed. It might be safest to wrap the expression in single quotes...

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

is an example.

And, in the utils directory, you can say:

```
make check_expr
```

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

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

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

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

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

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

on stdout, you will see something like:

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

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

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

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

Asterisk standard channel variables

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

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

- ${CDR(accountcode)} * - Account code (if specified)
- ${BLINDTRANSFER} - The name of the channel on the other side of a blind transfer
- ${BRIDGEPEER} - Bridged peer
- ${BRIDGEPVTCALLID} - Bridged peer PVT call ID (SIP Call ID if a SIP call)
- ${CALLERID(ani)} * - Caller ANI (PRI channels)
- ${CALLERID(ani2)} * - ANI2 (Info digits) also called Originating line information or OLI
- ${CALLERID(all)} * - Caller ID
- ${CALLERID(dnid)} * - Dialed Number Identifier
- ${CALLERID(name)} * - Caller ID Name only
- ${CALLERID(num)} * - Caller ID Number only
- ${CALLERID(rdnis)} * - Redirected Dial Number ID Service
- ${CALLINGANI2} * - Caller ANI2 (PRI channels)
- ${CALLINGPRES} * - Caller ID presentation for incoming calls (PRI channels)
- ${CALLINGTNS} * - Transit Network Selector (PRI channels)
- ${CALLINGTON} * - Caller Type of Number (PRI channels)
- ${CHANNEL} * - Current channel name
- ${CONTEXT} * - Current context
- ${DATETIME} * - Current date time in the format: DDMMYYYY-HH:MM:SS (Deprecated; use ${STRFTIME(${EPOCH},,%d%m%Y-%H:%M:%S)})
- ${DB_RESULT} - Result value of DB_EXISTS() dial plan function
- ${EPOCH} * - Current unix style epoch
- ${EXTEN} * - Current extension
- ${ENV(VAR)} - Environmental variable VAR
- ${GOTO_ON_BLINDXFR} - Transfer to the specified context/extension/priority after a blind transfer (use ^ characters in place of | to separate context/extension/priority when setting this variable from the dialplan)
- ${HANGUPCAUSE} * - Asterisk cause of hangup (inbound/outbound)
- ${HINT} * - Channel hints for this extension
- ${HINTNAME} * - Suggested Caller*ID name for this extension
- ${INVALID_EXTEN} - The invalid called extension (used in the "i" extension)
- ${LANGUAGE} * - Current language (Deprecated; use ${LANGUAGE()})
- ${LEN(VAR)} - String length of VAR (integer)
- ${PRIORITY} * - Current priority in the dialplan
- ${PRIREDIRECTREASON} - Reason for redirect on PRI, if a call was directed
- ${TIMESTAMP} * - Current date time in the format: YYYYMMDD-HHMMSS (Deprecated; use ${STRFTIME(${EPOCH},,%Y%m%d-%H%M%S)})
- ${TRANSFER_CONTEXT} - Context for transferred calls
- ${FORWARD_CONTEXT} - Context for forwarded calls
- ${DYNAMIC_PEERNAME} - The name of the channel on the other side when a dynamic feature is used.
- ${DYNAMIC_FEATURENAME} The name of the last triggered dynamic feature.
- ${UNIQUEID} * - Current call unique identifier
- ${SYSTEMNAME} * - value of the systemname option of asterisk.conf
- ${ENTITYID} * - Global Entity ID set automatically, or from asterisk.conf

Application return values

Many applications return the result in a variable that you read to get the result of the application.

These status fields are unique for each application. For the various status values, see each application's help text.

- ${AGISTATUS} * agi()
- ${AQMSTATUS} * addqueuemember()
- ${AVAILSTATUS} * chanisavail()
- ${CHECKGROUPSTATUS} * checkgroup()
- ${CHECKMD5STATUS} * checkmd5()
- ${CPLAYBACKSTATUS} * controlplayback()
- ${DIALSTATUS} * dial()
- ${DBGETSTATUS} * dbget()
- ${ENUMSTATUS} * enumlookup()
- ${HASVMSTATUS} * hasnewvoicemail()
- ${LOOKUPBLSTATUS} * lookupblacklist()
- ${OSPAUTHSTATUS} * ospauth()
- ${OSPLOOKUPSTATUS} * osplookup()
- ${OSPNEXTSTATUS} * ospnext()
- ${OSPFINISHSTATUS} * ospfinish()
- ${PARKEDAT} * parkandannounce()
- ${PLAYBACKSTATUS} * playback()
- ${PQMSTATUS} * pausequeuemember()
- ${PRIVACYMGRSTATUS} * privacymanager()
- ${QUEUESTATUS} * queue()
- ${RQMSTATUS} * removequeuemember()
- ${SENDIMAGESTATUS} * sendimage()
- ${SENDTEXTSTATUS} * sendtext()
- ${SENDURLSTATUS} * sendurl()
- ${SYSTEMSTATUS} * system()
- ${TRANSFERSTATUS} * transfer()
- ${TXTCIDNAMESTATUS} * txtcidname()
- ${UPQMSTATUS} * unpausequeuemember()
- ${VMSTATUS} * voicemail()
- ${VMBOXEXISTSSTATUS} * vmboxexists()
- ${WAITSTATUS} * waitforsilence()

Various application variables

- ${CURL} - Resulting page content for CURL()
- ${ENUM} - Result of application EnumLookup()
- ${EXITCONTEXT} - Context to exit to in IVR menu (Background()) or in the RetryDial() application
- ${MONITOR} - Set to "TRUE" if the channel is/has been monitored (app monitor())
- ${MONITOR_EXEC} - Application to execute after monitoring a call
- ${MONITOR_EXEC_ARGS} - Arguments to application
- ${MONITOR_FILENAME} - File for monitoring (recording) calls in queue
- ${QUEUE_PRIO} - Queue priority
- ${QUEUE_MAX_PENALTY} - Maximum member penalty allowed to answer caller
- ${QUEUE_MIN_PENALTY} - Minimum member penalty allowed to answer caller
- ${QUEUESTATUS} - Status of the call, one of: (TIMEOUT | FULL | JOINEMPTY | LEAVEEMPTY | JOINUNAVAIL | LEAVEUNAVAIL)
- ${QUEUEPOSITION} - When a caller is removed from a queue, his current position is logged in this variable. If the value is 0, then this means that the caller was serviced by a queue member. If non-zero, then this was the position in the queue the caller was in when he left.
- ${RECORDED_FILE} - Recorded file in record()
- ${TALK_DETECTED} - Result from talkdetect()
- ${TOUCH_MONITOR} - The filename base to use with Touch Monitor (auto record)
- ${TOUCH_MONITOR_PREF} - The prefix for automonitor recording filenames.
- ${TOUCH_MONITOR_FORMAT} - The audio format to use with Touch Monitor (auto record)
- ${TOUCH_MONITOR_OUTPUT} - Recorded file from Touch Monitor (auto record)
- ${TOUCH_MONITOR_MESSAGE_START} - Recorded file to play for both channels at start of monitoring session
- ${TOUCH_MONITOR_MESSAGE_STOP} - Recorded file to play for both channels at end of monitoring session
- ${TOUCH_MIXMONITOR} - The filename base to use with Touch MixMonitor (auto record)
- ${TOUCH_MIXMONITOR_FORMAT} - The audio format to use with Touch MixMonitor (auto record)
- ${TOUCH_MIXMONITOR_OUTPUT} - Recorded file from Touch MixMonitor (auto record)
- ${TXTCIDNAME} - Result of application TXTCIDName
- ${VPB_GETDTMF} - chan_vpb

MeetMe Channel Variables

- ${MEETME_RECORDINGFILE} - Name of file for recording a conference with the "r" option
- ${MEETME_RECORDINGFORMAT} - Format of file to be recorded
- ${MEETME_EXIT_CONTEXT} - Context for exit out of meetme meeting

- ${MEETME_AGI_BACKGROUND} - AGI script for Meetme (DAHDI only)
- ${MEETMESECS} * - Number of seconds a user participated in a MeetMe conference
- ${CONF_LIMIT_TIMEOUT_FILE} - File to play when time is up. Used with the L() option.
- ${CONF_LIMIT_WARNING_FILE} - File to play as warning if 'y' is defined. The default is to say the time remaining. Used with the L() option.
- ${MEETMEBOOKID} * - This variable exposes the bookid column for a realtime configured conference bridge.

VoiceMail Channel Variables

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

VMAuthenticate Channel Variables

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

DUNDiLookup Channel Variables

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

chan_dahdi Channel Variables

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

chan_sip Channel Variables

- ${SIPCALLID} * - SIP Call-ID: header verbatim (for logging or CDR matching)
- ${SIPDOMAIN} * - SIP destination domain of an inbound call (if appropriate)
- ${SIPFROMDOMAIN} - Set SIP domain on outbound calls
- ${SIPUSERAGENT} * - SIP user agent (deprecated)
- ${SIPURI} * - SIP uri
- ${SIP_MAX_FORWARDS} - Set the value of the Max-Forwards header for outbound call
- ${SIP_CODEC} - Set the SIP codec for an inbound call
- ${SIP_CODEC_INBOUND} - Set the SIP codec for an inbound call
- ${SIP_CODEC_OUTBOUND} - Set the SIP codec for an outbound call
- ${SIP_URI_OPTIONS} * - additional options to add to the URI for an outgoing call
- ${RTPAUDIOQOS} - RTCP QoS report for the audio of this call
- ${RTPVIDEOQOS} - RTCP QoS report for the video of this call

chan_agent Channel Variables

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

Dial Channel Variables

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

Chanisavail() Channel Variables

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

Dialplan Macros Channel Variables

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

ChanSpy Channel Variables

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

Open Settlement Protocol (OSP) Channel Variables

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

- ${OSPOUTSPID} - The outbound service provider identity.
- ${OSPOUTOCN} - The outbound operator company number.
- ${OSPOUTSPN} - The outbound service provider name.
- ${OSPOUTALTSPN} - The outbound alternate service provider name.
- ${OSPOUTMCC} - The outbound mobile country code.
- ${OSPOUTMNC} - The outbound mobile network code.
- ${OSPDIALSTR} - The outbound Dial command string.
- ${OSPINAUDIOQOS} - The inbound call leg audio QoS string.
- ${OSPOUTAUDIOQOS} - The outbound call leg audio QoS string.

Digit Manipulation Channel Variables

- `${REDIRECTING_CALLEE_SEND_MACRO}`
  Macro to call before sending a redirecting update to the callee

- `${REDIRECTING_CALLEE_SEND_MACRO_ARGS}`
  Arguments to pass to ${REDIRECTING_CALLEE_SEND_MACRO}

—

- `${REDIRECTING_CALLER_SEND_MACRO}`
  Macro to call before sending a redirecting update to the caller

- `${REDIRECTING_CALLER_SEND_MACRO_ARGS}`
  Arguments to pass to ${REDIRECTING_CALLER_SEND_MACRO}

- `${CONNECTED_LINE_CALLEE_SEND_MACRO}`
  Macro to call before sending a connected line update to the callee

- `${CONNECTED_LINE_CALLEE_SEND_MACRO_ARGS}`
  Arguments to pass to ${CONNECTED_LINE_CALLEE_SEND_MACRO}

—

- `${CONNECTED_LINE_CALLER_SEND_MACRO}`
  Macro to call before sending a connected line update to the caller

- `${CONNECTED_LINE_CALLER_SEND_MACRO_ARGS}`
  Arguments to pass to ${CONNECTED_LINE_CALLER_SEND_MACRO}

Distributed Universal Number Discovery (DUNDi)

Top-level page for all things DUNDi

Introduction to DUNDi

http://www.dundi.com

Mark Spencer, Digium, Inc.

DUNDi is essentially a trusted, peer-to-peer system for being able to call any phone number from the Internet. DUNDi works by creating a network of nodes called the "DUNDi E.164 Trust Group" which are bound by a common peering agreement known as the General Peering Agreement or GPA. The GPA legally binds the members of the Trust Group to provide good-faith accurate information to the other nodes on the network, and provides standards by which the community

can insure the integrity of the information on the nodes themselves. Unlike ENUM or similar systems, DUNDi is explicitly designed to preclude any necessity for a single centralized system which could be a source of fees, regulation, etc.

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

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

DUNDIQUERY and DUNDIRESULT

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

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

DUNDi Peering Agreement

```
DIGIUM GENERAL PEERING AGREEMENT (TM)
Version 1.0.0, September 2004
Copyright (C) 2004 Digium, Inc.
150 West Park Loop Suite 100, Huntsville, AL 35806 USA

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


-------------------------------------------------------


DIGIUM GENERAL PEERING AGREEMENT


PREAMBLE


For most of the history of telecommunications, the power of being
able to locate and communicate with another person in a system, be
it across a hall or around the world, has always centered around a
```

centralized authority -- from a local PBX administrator to regional and national RBOCs, generally requiring fees, taxes or regulation. By contrast, DUNDi is a technology developed to provide users the freedom to communicate with each other without the necessity of any centralized authority. This General Peering Agreement ("GPA") is used by individual parties (each, a "Participant") to allow them to build the E164 trust group for the DUNDi protocol.

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

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

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

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

The parties to this GPA agree as follows:

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

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

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

May, 1997.

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

(d) The term "Egress Gateway" refers an Internet facility that
provides a communications path to a Service or Services that may not
be directly addressable via the Internet.

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

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

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

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

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

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

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

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

(m) The term "Weight" refers to a numeric quality assigned to a

Route as per the DUNDi protocol specification. The current Weight definitions are shown in Exhibit A.

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

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

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

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

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

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

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

(f) A Participant may not disclose any specific Route, Service or

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

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

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

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

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

(c) A Participant may provide Routes for those Services whose Subscriber has authorized the Participant to do so, provided that the Participant is able to confirm that the Authorizing Individual is the Subscriber through:

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

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

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

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

(a) receipt of a revocation request;

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

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

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

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

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

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

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

10. CALLER IDENTIFICATION. A Participant will make a good faith effort to ensure the accuracy and appropriate nature of any caller identification that it transmits via any Route obtained from the

Peering System. Caller identification shall at least be provided as a valid E.164 number.

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

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

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

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

15. END-USER AGREEMENTS. The Participants may independently enter into agreements with end-users to provide certain services (e.g., fees to a Subscriber to originate Routes for that Service). To the

extent that provision of these services employs the Peering System, the Parties will include in their agreements with their end-users terms and conditions consistent with the terms of this GPA with respect to the exclusion of warranties, limitation of liability and Acceptable Use Policy. In no event may a Participant extend the warranty described in Section 12 in this GPA to any end-users.

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

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

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

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

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

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

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

(c) If the dispute has not been resolved pursuant to (a) or (b) above, the dispute shall be promptly referred, no later than one (1) year from the date of original notice and subject to applicable statute of limitations, to binding arbitration in accordance with the UNCITRAL Arbitration Rules in effect on the date of this contract. The appointing authority shall be the International Centre for Dispute Resolution. The case shall be administered by the International Centre for Dispute Resolution under its Procedures for Cases under the UNCITRAL Arbitration Rules. Each Participant shall bear its own expenses and shall share equally in fees of the arbitrator. All arbitrators shall have substantial experience in information technology and/or in the telecommunications business and shall be selected by the disputing participants in accordance with UNCITRAL Arbitration Rules. If any arbitrator, once selected is unable or unwilling to continue for any reason, replacement shall be filled via the process described above and a re-hearing shall be conducted. The disputing Participants will provide each other with all requested documents and records reasonably related to the dispute in a manner that will minimize the expense and inconvenience of both parties. Discovery will not include depositions or interrogatories except as the arbitrators expressly allow upon a showing of need. If disputes arise concerning discovery requests,

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

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

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

23. INDEPENDENT CONTRACTORS. Nothing in this GPA shall make the Parties partners, joint venturers, or otherwise associated in or with the business of the other. Parties are, and shall always remain, independent contractors. No Participant shall be liable for any debts, accounts, obligations, or other liabilities of the other Participant, its agents or employees. No party is authorized to

incur debts or other obligations of any kind on the part of or as agent for the other. This GPA is not a franchise agreement and does not create a franchise relationship between the parties, and if any provision of this GPA is deemed to create a franchise between the parties, then this GPA shall automatically terminate.

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

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

Exhibit A

Weight Range Requirements

0-99 May only be used under authorization of Owner

100-199 May only be used by the Owner's service provider, regardless of authorization.

200-299 Reserved -- do not use for e164 context.

300-399 May only be used by the owner of the code under which the Owner's number is a part of.

400-499 May be used by any entity providing access via direct connectivity to the Public Switched Telephone Network.

500-599 May be used by any entity providing access via indirect connectivity to the Public Switched Telephone Network (e.g. Via another VoIP provider)

600- Reserved-- do not use for e164 context.

Participant Participant

Company:

Address:

Email:

_____ _____
Authorized Signature Authorized Signature

Name:

END OF GENERAL PEERING AGREEMENT

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

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

E.164 NUmber Mapping (ENUM)

The ENUMLOOKUP Dialplan Function

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

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

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

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

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

ENUMLOOKUP Arguments

- number - Telephone number or search string. Only numeric values within this string are parsed; all other digits are ignored for search, but are re-written during NAPTR regexp expansion.
- service_type - tel, sip, h323, iax2, mailto, ...[any other string], ALL. Default type is "sip". Special name of "ALL" will create a list of method types across all NAPTR records for the search number, and then put the results in an ordinal list starting with 1. The position number specified will then be returned, starting with 1 as the first record (lowest value) in the list. The service types are not hardcoded in Asterisk except for the default (sip) if no other service type specified; any method type string (IANA-approved or not) may be used except for the string "ALL".
- options
  - c - count. Returns the number of records of this type are returned (regardless of order or priority.) If "ALL" is the specified service_type, then a count of all methods will be returned for the DNS record.
- record# - Which record to present if multiple answers are returned integer = The record in priority/order sequence based on the total count of records passed back by the query. If a service_type is specified, all entries of that type will be sorted into an ordinal list starting with 1 (by order first, then priority). The default of options is "1"
- zone_suffix - Allows customization of the ENUM zone. Default is e164.arpa.

ENUMLOOKUP Examples

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

```
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 10 100 "u" "E2U+tel"
"Unable to render embedded object: File (+12125551212) not found." .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 21 100 "u" "E2U+tel"
"Unable to render embedded object: File (+14155551212) not found." .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 25 100 "u" "E2U+sip"
"Unable to render embedded object: File (2203@sip.fox-den.com) not
found." . 0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 26 100 "u"
"E2U+sip" "Unable to render embedded object: File
(1234@sip-2.fox-den.com) not found." .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 30 100 "u" "E2U+sip"
"Unable to render embedded object: File (\\1@sip-3.fox-den.com) not
found." .
0.2.0.1.1.6.5.1.0.3.1.loligo.com. 3600 IN NAPTR 55 100 "u"
"E2U+mailto" "Unable to render embedded object: File
(jtodd@fox-den.com) not found." .
```

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

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

returns: ${foo}="2203@sip.fox-den.com"

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

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

returns: ${foo}="+12125551212"

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

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

returns: ${foo}=3

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

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

returns: ${foo}="+14155551212"

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

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

returns: ${foo}=6

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

```
exten =>
100,1,Set(foo=${ENUMLOOKUP(+13015611020,ALL,,2,loligo.com)})
```

returns: ${foo}="tel:+14155551212" [note the "tel:" prefix in the string]

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

```
exten => 100,1,Set(foo=${ENUMLOOKUP(+437203001721)})
```

returns: ${foo}="enum-test@sip.nemox.net" [note: this result is subject to change as it is "live" DNS and not under my control]

Example 8: Look up the ISN mapping in freenum.org alpha test zone

```
exten => 100,1,Set(foo=${ENUMLOOKUP(1234*256,,,,freenum.org)})
```

returns: ${foo}="1234@204.91.156.10" [note: this result is subject to change as it is "live" DNS]

Example 9: Give back the first SIP pointer for a number in the enum.yoydynelabs.com zone (invalid lookup)

```
exten =>
100,1,Set(foo=${ENUMLOOKUP(1234567890,sip,,1,enum.yoyodynelabs.com)})
```

returns: ${foo}=""

ENUMLOOKUP Usage Notes and Subtle Features

- The use of "" in lookups is confusing, and warrants further explanation. All E.164 numbers ("global phone numbers") by definition need a leading "" during ENUM lookup. If you neglect to add a leading "", you may discover that numbers that seem to exist in the DNS aren't getting matched by the system or are returned with a null string result. This is due to the NAPTR reply requiring a "" in the regular expression matching sequence. Older versions of Asterisk add a "" from within the code, which may confuse administrators converting to the new function. Please ensure that all ENUM (e164.arpa) lookups contain a leading "" before lookup, so ensure your lookup includes the leading plus sign. Other DNS trees may or may not require a leading "" - check before using those trees, as it is possible the parsed NAPTRs will not provide correct results unless you have the correct dialed string. If you get console messages like "WARNING[24907]: enum.c:222 parse_naptr: NAPTR Regex match failed." then it is very possible that the returned NAPTR expects a leading "" in the search string (or the returned NAPTR is mis-formed.)

- If a query is performed of type "c" ("count") and let's say you get back 5 records and then some seconds later a query is made against record 5 in the list, it may not be the case that the DNS resolver has the same answers as it did a second or two ago - maybe there are only 4 records in the list in the newest query. The resolver should be the canonical storage location for DNS records, since that is the intent of ENUM. However, some obscure future cases may have wildly changing NAPTR records within several seconds. This is a corner case, and probably only worth noting as a very rare circumstance. (note: I do not object to Asterisk's dnsmgr method of locally caching DNS replies, but this method needs to honor the TTL given by the remote zone master. Currently, the ENUMLOOKUP function does not use the dnsmgr method of caching local DNS replies.)

- If you want strict NAPTR value ordering, then it will be necessary to use the "ALL" method to incrementally step through the different returned NAPTR pointers. You will need to use string manipulation to strip off the returned method types, since the results will look like "sip:12125551212" in the returned value. This is a non-trivial task, though it is required in order to have strict RFC compliance and to comply with the desires of the remote party who is presenting NAPTRs in a particular order for a reason.

- Default behavior for the function (even in event of an error) is to move to the next priority, and the result is a null value. Most ENUM lookups are going to be failures, and it is the responsibility of the dialplan administrator to manage error conditions within their dialplan. This is a change from the old app_enumlookup method and it's arbitrary priority jumping based on result type or failure.

- Anything other than digits will be ignored in lookup strings. Example: a search string of "+4372030blah01721" will turn into 1.2.7.1.0.0.3.0.2.7.3.4.e164.arpa. for the lookup. The NAPTR parsing may cause unexpected results if there are strings inside your NAPTR lookups.
  If there exist multiple records with the same weight and order as a result of your query, the function will RANDOMLY select a single NAPTR from those equal results.

- Currently, the function ignores the settings in enum.conf as the search zone name is now specified within the function, and the H323 driver can be chosen by the user via the dialplan. There were no other values in this file, and so it becomes deprecated.

- The function will digest and return NAPTRs which use older (deprecated) style, reversed method strings such as "sip+E2U" instead of the more modern "E2U+sip"

- There is no provision for multi-part methods at this time. If there are multiple NAPTRs with (as an example) a method of "E2U+voice:sip" and then another NAPTR in the same DNS record with a method of ""E2U+sip", the system will treat these both as method "sip" and they will be separate records from the perspective of the function. Of course, if both records point to the same URI and have equal priority/weight (as is often the case) then this will cause no serious difficulty, but it bears mentioning.

- ISN (ITAD Subscriber Number) usage: If the search number is of the form ABC*DEF (where ABC and DEF are at least one numeric digit) then perform an ISN-style lookup where the lookup is manipulated to C.B.A.DEF.domain.tld (all other settings and options apply.) See http://www.freenum.org/ for more details on ISN lookups. In the unlikely event you wish to avoid ISN re-writes, put an "n" as the first digit of the search string - the "n" will be ignored for the search.

More ENUMLOOKUP Examples

All examples below except where noted use "e164.arpa" as the referenced domain, which is the default domain name for ENUMLOOKUP. All numbers are assumed to not have a leading "+" as dialed by the inbound channel, so that character is added where necessary during ENUMLOOKUP function calls.

```
; example 1
;
; Assumes North American international dialing (011) prefix.
; Look up the first SIP result and send the call there, otherwise
; send the call out a PRI. This is the most simple possible
```

```
; ENUM example, but only uses the first SIP reply in the list of
; NAPTR(s).
;
exten => _011.,1,Set(enumresult=${ENUMLOOKUP(${EXTEN:3})})
exten => _011.,n,Dial(SIP/${enumresult})
exten => _011.,n,Dial(DAHDI/g1/${EXTEN})
;
; end example 1
; example 2
;
; Assumes North American international dialing (011) prefix.
; Check to see if there are multiple SIP NAPTRs returned by
; the lookup, and dial each in order. If none work (or none
; exist) then send the call out a PRI, group 1.
;
exten =>
_011.,1,Set(sipcount=${ENUMLOOKUP(${EXTEN:3},sip,c)}|counter=0)
exten => _011.,n,While($["${counter}"<"${sipcount}"])
exten => _011.,n,Set(counter=$[${counter}+1])
exten => _011.,n,Dial(SIP/${ENUMLOOKUP(${EXTEN:3},sip,,${counter})})

exten => _011.,n,EndWhile
exten => _011.,n,Dial(DAHDI/g1/${EXTEN})
;
; end example 2
; example 3
;
; This example expects an ${EXTEN} that is an e.164 number (like
; 14102241145 or 437203001721)
; Search through e164.arpa and then also search through e164.org
; to see if there are any valid SIP or IAX termination capabilities.

; If none, send call out via DAHDI channel 1.
;
; Start first with e164.arpa zone...
;
exten => _X.,1,Set(sipcount=${ENUMLOOKUP(${EXTEN},sip,c)}|counter=0)

exten => _X.,2,GotoIf($["${counter}"<"${sipcount}"]?3:6)
exten => _X.,3,Set(counter=$[${counter}+1])
exten => _X.,4,Dial(SIP/${ENUMLOOKUP(${EXTEN},sip,,${counter})})
exten => _X.,5,GotoIf($["${counter}"<"${sipcount}"]?3:6)
;
exten =>
_X.,6,Set(iaxcount=${ENUMLOOKUP(${EXTEN},iax2,c)}|counter=0)
exten => _X.,7,GotoIf($["${counter}"<"${iaxcount}"]?8:11)
```

```
exten => _X.,8,Set(counter=$[${counter}+1])
exten => _X.,9,Dial(IAX2/${ENUMLOOKUP(${EXTEN},iax2,,${counter})})
exten => _X.,10,GotoIf($["${counter}"<"${iaxcount}"]?8:11)
;
exten => _X.,11,NoOp("No valid entries in e164.arpa for ${EXTEN} -
checking in e164.org")
;
; ...then also try e164.org, and look for SIP and IAX NAPTRs...
;
exten =>
_X.,12,Set(sipcount=${ENUMLOOKUP(${EXTEN},sip,c,,e164.org)}|counter=0)

exten => _X.,13,GotoIf($["${counter}"<"${sipcount}"]?14:17)
exten => _X.,14,Set(counter=$[${counter}+1])
exten =>
_X.,15,Dial(SIP/${ENUMLOOKUP(${EXTEN},sip,,${counter},e164.org)})
exten => _X.,16,GotoIf($["${counter}"<"${sipcount}"]?14:17)
;
exten =>
_X.,17,Set(iaxcount=${ENUMLOOKUP(${EXTEN},iax2,c,,e164.org)}|counter=0
exten => _X.,18,GotoIf($["${counter}"<"${iaxcount}"]?19:22)
exten => _X.,19,Set(counter=$[${counter}+1])
exten =>
_X.,20,Dial(IAX2/${ENUMLOOKUP(${EXTEN},iax2,,${counter},e164.org)})
exten => _X.,21,GotoIf($["${counter}"<"${iaxcount}"]?19:22)
;
; ...then send out PRI.
;
exten => _X.,22,NoOp("No valid entries in e164.org for ${EXTEN} -
sending out via DAHDI")
```

```
exten => _X.,23,Dial(DAHDI/g1/${EXTEN})
;
; end example 3
```

Features

Miscellaneous documents that talk about Asterisk functionality. All of this needs to be integrated into Configuration and Operation.

Asterisk Applications

This page is a container page for Asterisk applications, e.g. those things that appear in the apps source directory.

MacroExclusive()

About the MacroExclusive application

By: Steve Davies <steve@connection-telecom.com

The MacroExclusive application was added to solve the problem of synchronization between calls running at the same time.

This is usually an issue when you have calls manipulating global variables or the Asterisk database, but may be useful elsewhere.

Consider this example macro, intended to return a "next" number - each caller is intended to get a different number:

```
[macro-next]
exten => s,1,Set(RESULT=${COUNT})
exten => s,n,SetGlobalVar(COUNT=$[${COUNT} + 1])
```

The problem is that in a box with high activity, you can be sure that two calls will come along together - both will get the same "RESULT", or the "COUNT" value will get mangled.

Calling this Macro via MacroExclusive will use a mutex to make sure that only one call executes in the Macro at a time. This ensures that the two lines execute as a unit.

Note that even the s,2 line above has its own race problem. Two calls running that line at once will step on each other and the count will end up as +1 rather than +2.

I've also been able to use MacroExclusive where I have two Macros that need to be mutually exclusive.

Here's the example:

```
[macro-push]
; push value ${ARG2} onto stack ${ARG1}
exten => s,1,Set(DB(STACK/${ARG1})=${ARG2}^${DB(STACK/${ARG1})})

[macro-pop]
; pop top value from stack ${ARG1}
exten => s,1,Set(RESULT=${DB(STACK/${ARG1})})
exten => s,n,Set(DB(STACK/${ARG1})=${CUT(RESULT,^,2)})
exten => s,n,Set(RESULT=${CUT(RESULT,^,1)})
```

All that futzing with the STACK/${ARG1} in the astdb needs protecting if this is to work. But neither push nor pop can run together.

So add this "pattern":

```
[macro-stack]
exten => Macro(${ARG1},${ARG2},${ARG3})
```

... and use it like so:

```
exten => s,1,MacroExclusive(stack,push,MYSTACK,bananas)
exten => s,n,MacroExclusive(stack,push,MYSTACK,apples)
exten => s,n,MacroExclusive(stack,push,MYSTACK,guavas)
exten => s,n,MacroExclusive(stack,push,MYSTACK,pawpaws)
exten => s,n,MacroExclusive(stack,pop,MYSTACK) ; RESULT gets pawpaws
(yum)
exten => s,n,MacroExclusive(stack,pop,MYSTACK) ; RESULT gets guavas
exten => s,n,MacroExclusive(stack,pop,MYSTACK) ; RESULT gets apples
exten => s,n,MacroExclusive(stack,pop,MYSTACK) ; RESULT gets bananas
```

We get to the push and pop macros "via" the stack macro. But only one call can execute the stack macro at a time; ergo, only one of push OR pop can run at a time.

Hope people find this useful.

Lastly, its worth pointing out that only Macros that access shared data will require this MacroExclusive protection. And Macro's that you call with macroExclusive should run quickly or you will clog up your Asterisk system.

SMS()

The SMS application

SMS() is an application to handles calls to/from text message capable phones and message centres using ETSI ES 201 912 protocol 1 FSK messaging over analog calls.

Basically it allows sending and receiving of text messages over the PSTN. It is compatible with BT Text service in the UK and works on ISDN and PSTN lines. It is designed to connect to an ISDN or DAHDI interface directly and uses FSK so would
probably not work over any sort of compressed link (like a VoIP call using GSM codec).

Typical applications include:-

1. Connection to a message centre to send text messages - probably initiated via the manager interface or "outgoing" directory
2. Connection to an POTS line with an SMS capable phone to send messages - probably initiated via the manager interface or "outgoing" directory
3. Acceptance of calls from the message centre (based on CLI) and storage of received messages
4. Acceptance of calls from a POTS line with an SMS capable phone and storage of received messages

Arguments to sms():

- First argument is queue name
- Second is options:
  - a: SMS() is to act as the answering side, and so send the initial FSK frame
  - s: SMS() is to act as a service centre side rather than as terminal equipment

- If a third argument is specified, then SMS does not handle the call at all, but takes the third argument as a destination number to send an SMS to

- The forth argument onward is a message to be queued to the number in the third argument. All this does is create the file in the me-sc directory.

- If 's' is set then the number is the source address and the message placed in the sc-me directory.

All text messages are stored in /var/spool/asterisk/sms

A log is recorded in /var/log/asterisk/sms

There are two subdirectories called sc-me.<queuename> holding all messages from service centre to phone, and me-sc.<queuename> holding all messages from phone to service centre.

In each directory are messages in files, one per file, using any filename not starting with a dot.

When connected as a service centre, SMS(s) will send all messages waiting in the sc-me-<queuename> directory, deleting the files as it goes. Any received in this mode are placed in the me-sc-<queuename> directory.

When connected as a client, SMS() will send all messages waiting in the me-sc-<queuename> directory, deleting the files as it goes. Any received in this mode are placed in the sc-me-<queuename> directory.

Message files created by SMS() use a time stamp/reference based filename.

The format of the sms file is lines that have the form of key=value

Keys are :

- oa - Originating Address. Telephone number, national number if just digits. Telephone number starting with + then digits for international. Ignored on sending messages to service centre (CLI used)
- da - Destination Address. Telephone number, national number if just digits. Telephone number starting with + then digits for international.
- scts - Service Centre Time Stamp in the format YYYY-MM-DD HH:MM:SS
- pid - Protocol Identifier (decimal octet value)

- dcs - Data coding scheme (decimal octet value)
- mr - Message reference (decimal octet value)
- ud - The message (see escaping below)
- srr - 0/1 Status Report Request
- rp - 0/1 Return Path
- vp - mins validity period

Omitted fields have default values.

Note that there is special format for ud, ud# instead of ud= which is followed by raw hex (2 characters per octet). This is used in output where characters other than 10,13,32-126,128-255 are included in the data. In this case a comment (line starting ;) is added showing the printable characters

When generating files to send to a service centre, only da and ud need be specified. oa is ignored.

When generating files to send to a phone, only oa and ud need be specified. da is ignored.

When receiving a message as a service centre, only the destination address is sent, so the originating address is set to the callerid.

EXAMPLES

The following are examples of use within the UK using BT Text SMS/landline service.

This is a context to use with a manager script.

```
[smsdial]
; create and send a text message, expects number+message and
; connect to 17094009
exten => _X.,1,SMS(${CALLERIDNUM},,${EXTEN},${CALLERIDNAME})
exten => _X.,n,SMS(${CALLERIDNUM})
exten => _X.,n,Hangup
```

The script sends

```
action: originate
 callerid: message <from>
 exten: to
 channel: Local/17094009
 context: smsdial
 priority: 1
```

You put the message as the name of the caller ID (messy, I know), the originating number and hence queue name as the number of the caller ID and the exten as the number to which the sms is to be sent. The context uses SMS to create the message in the queue and then SMS to communicate with 17094009 to actually send the message.

Note that the 9 on the end of 17094009 is the sub address 9 meaning no sub address (BT specific). If a different digit is used then that is the sub address for the sending message source address (appended to the outgoing CLI by BT).

For incoming calls you can use a context like this :-

```
[incoming]
exten => _XXXXXX/_8005875290,1,SMS(${EXTEN:3},a)
exten => _XXXXXX/_8005875290,n,System(/usr/lib/asterisk/smsin
${EXTEN:3})
exten =>
_XXXXXX/_80058752[0-8]0,1,SMS(${EXTEN:3}${CALLERIDNUM:8:1},a)
exten => _XXXXXX/_80058752[0-8]0,n,System(/usr/lib/asterisk/smsin
${EXTEN>:3}${CALLERIDNUM:8:1})
exten => _XXXXXX/_80058752[0-8]0,n,Hangup
```

In this case the called number we get from BT is 6 digits (XXXXXX) and we are using the last 3 digits as the queue name.

Priority 1 causes the SMS to be received and processed for the incoming call. It is from 080058752X0. The two versions handle the queue name as 3 digits (no sub address) or 4 digits (with sub address). In both cases, after the call a script (smsin) is run - this is optional, but is useful to actually processed the received queued SMS. In our case we email them based on the target number. Priority 3 hangs up.

If using the CAPI drivers they send the right CLI and so the _800... would be _0800...

Asterisk Call FIles

Asterisk call files

Asterisk has the ability to initiate a call from outside of the normal methods such as the dialplan, manager interface, or spooling interface.

Using the call file method, you must give Asterisk the following information:

- How to perform the call, similar to the Dial() application
- What to do when the call is answered

With call files you submit this information simply by creating a file with the required syntax and placing it in the outgoing spooling directory, located by default in /var/spool/asterisk/outgoing/ (configurable in asterisk.conf).

The pbx_spool module aggressively examines the directory contents every second, creating a new call for every call file it finds. Do NOT write or create the call file directly in the outgoing directory, but always create the file in another directory of the same filesystem and then move the file to the /var/spool/asterisk/outgoing directory, or Asterisk may read just a partial file.

The call file syntax

The call file consists of <Key>: <value> pairs; one per line.

Comments are indicated by a '#' character that begins a line, or follows a space or tab character. To be consistent with the configuration files in Asterisk, comments can also be indicated by a semicolon. However, the multiline comments (;----;) used in Asterisk configuration files are not supported. Semicolons can be escaped by a backslash.

The following keys-value pairs are used to specify how setup a call:

- Channel: <channel> - The channel to use for the new call, in the form **technology/resource** as in the Dial application. This value is required.

- Callerid: <callerid> - The caller id to use.

- WaitTime: <number> - How many seconds to wait for an answer before the call fails (ring cycle). Defaults to 45 seconds.

- Maxretries: <number> - Number of retries before failing, not including the initial attempt. Default = 0 e.g. don't retry if fails.

- RetryTime: <number> - How many seconds to wait before retry. The default is 300 (5 minutes).

- Account: <account> - The account code for the call. This value will be assigned to CDR(accountcode)

When the call answers there are two choices:

1. Execute a single application, or
2. Execute the dialplan at the specified context/extension/priority.

To execute an application:

- Application: <appname> - The application to execute

- Data: <args> - The application arguments

To start executing applications in the dialplan:

- Context: <context> - The context in the dialplan

- Extension: <exten> - The extension in the specified context

- Priority: <priority> - The priority of the specified extension; (numeric or label)

- Setvar: <var=value> - You may also assign values to variables that will be available to the channel, as if you had performed a Set(var=value) in the dialplan. More than one Setvar: may be specified.

The processing of the call file ends when the call is answered and terminated; when the call was not answered in the initial attempt and subsequent retries; or if the call file can't be successfully read and parsed.

To specify what to do with the call file at the end of processing:

- Archive: <yes|no> - If "no" the call file is deleted. If set to "yes" the call file is moved to the "outgoing_done" subdirectory of the Asterisk spool directory. The default is to delete the call file.

If the call file is archived, Asterisk will append to the call file:

- Status: <exitstatus> - Can be "Expired", "Completed" or "Failed"

Other lines generated by Asterisk:

Asterisk keep track of how many retries the call has already attempted, appending to the call file the following key-pairs in the form:

```
StartRetry: <pid> <retrycount> (<time>)
EndRetry: <pid> <retrycount> (<time>)
```

With the main process ID (pid) of the Asterisk process, the retry number, and the attempts start and end times in time_t format.

Directory locations

- <astspooldir>/outgoing - The outgoing dir, where call files are put for processing

- <astspooldir>/outgoing_done - The archive dir

- <astspooldir> - Is specified in asterisk.conf, usually /var/spool/asterisk

How to schedule a call

Call files that have the time of the last modification in the future are ignored by Asterisk. This makes it possible to modify the time of a call file to the wanted time, move to the outgoing directory, and Asterisk will attempt to create the call at that time.

Asterisk Command Line Interface

In addition to being the console for Asterisk, the CLI also sports several features that make it very helpful to use for obtaining information and affecting system configuration. The console can also be seen by starting a remote console, which connects to the running daemon and shows much of the same information as if using the daemon in foreground mode.

Connecting a remote console is as easy as using the -r or -R flags. The only difference between these flags is that the uppercase variation (-R) will automatically reconnect to the daemon (or at least retry) if the daemon restarts. To exit a remote console, simply type 'quit' or 'exit'. Please note that you can differentiate between a remote console and the Asterisk console, as 'quit' or 'exit' will not function on the main console, which prevents an accidental shutdown of the daemon. If you would like to shutdown the Asterisk daemon, you can use the 'stop' set of commands, such as 'stop now',
'stop gracefully', or 'stop when convenient'.

Once on the console, the 'help' command may be used to see a list of commands available for use. Note that in addition to the 'help' command, the Asterisk CLI sports tab command line completion on all commands, including many arguments. To use tab command line completion, simply press the <Tab> key at any time while entering the beginning of any command. If the command can be completed unambiguously, it will do so, otherwise it will complete as much of the command as possible. Additionally, Asterisk will print a list of all possible matches, if possible.

The 'help' command may also be used to obtain more detailed information on how to use a particular command. For example, if you type 'help core show', Asterisk will respond with a list of all commands that start with that string. If you type 'help core show version', specifying a complete command, Asterisk will respond with a usage message which describes how to use that command. As with other commands on the Asterisk console, the help command also responds to tab command line completion.

Asterisk Manager Interface (AMI) Changes

Container page for AMI related content.

AMI 1.1 Changes

Changes to manager version 1.1:

SYNTAX CLEANUPS

```
- Response: headers are now either
 "Success" - Action OK, this message contains response
 "Error"  - Action failed, reason in Message: header
 "Follows" - Action OK, response follows in following Events.

- Manager version changed to 1.1
```

CHANGED EVENTS AND ACTIONS

```
- The Hold/Unhold events
 - Both are now "Hold" events
  For hold, there's a "Status: On" header, for unhold, status is off
 - Modules chan_sip/chan_iax2

- The Ping Action
 - Now use Response: success
 - New header "Ping: pong" :-)

- The Events action
 - Now use Response: Success
 - The new status is reported as "Events: On" or "Events: Off"

- The JabberSend action
 - The Response: header is now the first header in the response
 - now sends "Response: Error" instead of "Failure"

- Newstate and Newchannel events
 - these have changed headers
 "State"  -> ChannelStateDesc Text based channel state
      -> ChannelState  Numeric channel state
```

```
  - The events does not send "<unknown>" for unknown caller IDs just
an empty field

- Newchannel event
 - Now includes "AccountCode"

- Newstate event
 - Now has "CalleridNum" for numeric caller id, like Newchannel
 - The event does not send "<unknown>" for unknown caller IDs just
an empty field

- Newexten and VarSet events
 - Now are part of the new Dialplan privilege class, instead of the
Call class

- Dial event
 - Event Dial has new headers, to comply with other events
 - Source -> Channel   Channel name (caller)
 - SrcUniqueID -> UniqueID   Uniqueid
 (new)  -> Dialstring   Dialstring in app data

- Link and Unlink events
 - The "Link" and "Unlink" bridge events in channel.c are now
renamed to "Bridge"
 - The link state is in the bridgestate: header as "Link" or
"Unlink"
 - For channel.c bridges, "Bridgetype: core" is added. This opens up
for
   bridge events in rtp.c
 - The RTP channel also reports Bridge: events with bridgetypes
  - rtp-native RTP native bridge
  - rtp-direct RTP peer-2-peer bridge (NAT support only)
  - rtp-remote Remote (re-invite) bridge. (Not reported yet)

- The "Rename" manager event has a renamed header, to use the same
 terminology for the current channel as other events
 - Oldname -> Channel

- The "NewCallerID" manager event has a renamed header
 - CallerID -> CallerIDnum
 - The event does not send "<unknown>" for unknown caller IDs just
an empty field

- Reload event
 - The "Reload" event sent at manager reload now has a new header
and is now implemented
```

```
      in more modules than manager to alert a reload. For channels,
there's a CHANNELRELOAD
      event to use.
  (new)  -> Module: manager | CDR | DNSmgr | RTP | ENUM
  (new)  -> Status: enabled | disabled
  - To support reload events from other modules too
   - cdr module added


- Status action replies (Event: Status)
  Header changes
  - link  -> BridgedChannel
  - Account -> AccountCode
  - (new)  -> BridgedUniqueid


- StatusComplete Event
  New header
  - (new)  -> Items  Number of channels reported



- The ExtensionStatus manager command now has a "StatusDesc" field
with text description of the state


- The Registry and Peerstatus events in chan_sip and chan_iax now
use "ChannelType" instead of "ChannelDriver"


- The Response to Action: IAXpeers now have a Response: Success
header


- The MeetmeJoin now has caller ID name and Caller ID number fields
(like MeetMeLeave)


- Action DAHDIShowChannels
  Header changes
  - Channel: -> DAHDIChannel
  For active channels, the Channel: and Uniqueid: headers are added
  You can now add a "DAHDIChannel: " argument to DAHDIshowchannels
actions
  to only get information about one channel.


- Event DAHDIShowChannelsComplete
  New header
  - (new)  -> Items:  Reports number of channels reported


- Action VoicemailUsersList
  Added new headers for SayEnvelope, SayCID, AttachMessage, CanReview
        and CallOperator voicemail configuration settings.
```

```
- Action Originate
 Now requires the new Originate privilege.
 If you call out to a subshell in Originate with the Application
parameter,
  you now also need the System privilege.

- Event QueueEntry now also returns the Uniqueid field like other
events from app_queue.

- Action IAXpeerlist
 Now includes if the IAX link is a trunk or not

- Action IAXpeers
 Now includes if the IAX link is a trunk or not

- Action Ping
 Response now includes a timestamp

- Action SIPshowpeer
 Response now includes the configured parkinglot
```

```
- Action SKINNYshowline
  Response now includes the configured parkinglot
```

NEW ACTIONS

```
- Action: DataGet
 Modules: data.c
 Purpose:
   To be able to retrieve the asterisk data tree.
 Variables:
    ActionID: <id>          Action ID for this transaction. Will be
returned.
    Path: <data path>       The path to the callback node to
retrieve.
    Filter: <filter>        Which nodes to retrieve.
    Search: <search>        Search condition.

- Action: IAXregistry
 Modules: chan_iax2
 Purpose:
   To list all IAX2 peers in the IAX registry with their registration
status.
 Variables:
    ActionID: <id>  Action ID for this transaction. Will be returned.

- Action: ModuleLoad
 Modules: loader.c
 Purpose:
   To be able to unload, reload and unload modules from AMI.
 Variables:
    ActionID: <id>          Action ID for this transaction. Will be
returned.
      Module: <name>          Asterisk module name (including .so
extension)
       or subsystem identifier:
     cdr, enum, dnsmgr, extconfig, manager, rtp, http
          LoadType: load | unload | reload
                          The operation to be done on module
 If no module is specified for a reload loadtype, all modules are
reloaded

- Action: ModuleCheck
 Modules: loader.c
 Purpose:
   To check version of a module - if it's loaded
```

```
 Variables:
   ActionID: <id>              Action ID for this transaction. Will be
returned.
      Module: <name>              Asterisk module name (not including
extension)
 Returns:
  If module is loaded, returns version number of the module

  Note: This will have to change. I don't like sending Response:
failure
  on both command not found (trying this command in earlier versions
of
  Asterisk) and module not found.
  Also, check if other manager actions behave that way.

- Action: QueueSummary
 Modules: app_queue
 Purpose:
  To request that the manager send a QueueSummary event (see the NEW
EVENTS
     section for more details).
 Variables:
   ActionID: <id>  Action ID for this transaction. Will be returned.
   Queue: <name>    Queue for which the summary is desired

- Action: QueuePenalty
 Modules: app_queue
 Purpose:
  To change the penalty of a queue member from AMI
 Variables:
   Interface: <tech/name> The interface of the member whose penalty
you wish to change
   Penalty:  <number>  The new penalty for the member. Must be
nonnegative.
   Queue:  <name>  If specified, only set the penalty for the member
for this queue;
     Otherwise, set the penalty for the member in all queues to
which
     he belongs.

- Action: QueueRule
 Modules: app_queue
 Purpose:
  To list queue rules defined in queuerules.conf
 Variables:
   Rule: <name>    The name of the rule whose contents you wish to
```

list. If this variable
       is not present, all rules in queuerules.conf will be listed.

- Action: Atxfer
 Modules: none
 Purpose:
  Initiate an attended transfer
 Variables:
  Channel: The transferer channel's name
  Exten: The extension to transfer to
  Priority: The priority to transfer to
  Context: The context to transfer to

- Action: SipShowRegistry
 Modules: chan_sip
 Purpose:
  To request that the manager send a list of RegistryEntry events.
 Variables:
   ActionId: <id> Action ID for this transaction. Will be returned.

- Action: QueueReload
 Modules: app_queue
 Purpose:
  To reload queue rules, a queue's members, a queue's parameters, or
all of the aforementioned
 Variable:
  Queuename: <name> The name of the queue to take action on. If no
queue name is specified, then all queues are affected
  Rules: <yes or no> Whether to reload queue_rules.conf
  Members: <yes or no> Whether to reload the queue's members
  Parameters: <yes or no> Whether to reload the other queue options

- Action: QueueReset
 Modules: app_queue
 Purpose:
  Reset the statistics for a queue
 Variables:
  Queuename: <name> The name of the queue on which to reset
statistics

- Action: SKINNYdevices
 Modules: chan_skinny
 Purpose:
  To list all SKINNY devices configured.
 Variables:
  ActionId: <id> Action ID for this transaction. Will be returned.

```
- Action: SKINNYlines
 Modules: chan_skinny
 Purpose:
  To list all SKINNY lines configured.
 Variables:
  ActionId: <id> Action ID for this transaction. Will be returned.

- Action SKINNYshowdevice
 Modules: chan_skinny
 Purpose:
  To list the information about a specific SKINNY device.
 Variables:
  Device: <device> Device to show information about.

- Action SKINNYshowline
 Modules: chan_skinny
 Purpose:
  To list the information about a specific SKINNY line.
 Variables:
  Line: <line> Line to show information about.

- Action: CoreSettings
 Modules: manager.c
 Purpose: To report core settings, like AMI and Asterisk version,
  maxcalls and maxload settings.
  * Integrated in SVN trunk as of May 4th, 2007
 Example:
  Response: Success
  ActionID: 1681692777
  AMIversion: 1.1
  AsteriskVersion: SVN-oej-moremanager-r61756M
  SystemName: EDVINA-node-a
  CoreMaxCalls: 120
  CoreMaxLoadAvg: 0.000000
  CoreRunUser: edvina
  CoreRunGroup: edvina

- Action: CoreStatus
 Modules: manager.c
 Purpose: To report current PBX core status flags, like
  number of concurrent calls, startup and reload time.
  * Integrated in SVN trunk as of May 4th, 2007
 Example:
  Response: Success
  ActionID: 1649760492
```

```
  CoreStartupTime: 22:35:17
  CoreReloadTime: 22:35:17
  CoreCurrentCalls: 20


- Action: MixMonitorMute
 Modules: app_mixmonitor.c
 Purpose:
  Mute / unMute a Mixmonitor recording.
 Variables:
  ActionId: <id> Action ID for this transaction. Will be returned.
  Channel: the channel MixMonitor is running on
  Direction: Which part of the recording to mute:  read, write or
both (from
```

```
 channel, to channel or both channels).
State: Turn mute on or off : 1 to turn on, 0 to turn off.
```

NEW EVENTS

```
- Event: FullyBooted
 Modules: loader.c
 Purpose:
  It is handy to have a single event notification for when all
Asterisk
  modules have been loaded--especially for situations like running
  automated tests. This event will fire 1) immediately upon all
modules
  loading or 2) upon connection to the AMI interface if the modules
have
  already finished loading before the connection was made. This
ensures
  that a user will never miss getting a FullyBooted event. In vary
rare
  circumstances, it might be possible to get two copies of the
message
  if the AMI connection is made right as the modules finish loading.
 Example:
  Event: FullyBooted
  Privilege: system,all
  Status: Fully Booted

- Event: Transfer
 Modules: res_features, chan_sip
 Purpose:
  Inform about call transfer, linking transferer with transfer
target
  You should be able to trace the call flow with this missing piece
  of information. If it works out well, the "Transfer" event should
  be followed by a "Bridge" event
  The transfermethod: header informs if this is a pbx core transfer
  or something done on channel driver level. For SIP, check the
example:
 Example:

  Event: Transfer
  Privilege: call,all
  TransferMethod: SIP
  TransferType: Blind
  Channel: SIP/device1-01849800
```

```
  SIP-Callid: 091386f505842c87016c4d93195ec67d@127.0.0.1
  TargetChannel: SIP/device2-01841200
  TransferExten: 100
  TransferContext: default

- Event: ChannelUpdate
 Modules: chan_sip.c, chan_iax2.c
 Purpose:
  Updates channel information with ID of PVT in channel driver, to
  be able to link events on channel driver level.
   * Integrated in SVN trunk as of May 4th, 2007

  Example:

  Event: ChannelUpdate
  Privilege: system,all
  Uniqueid: 1177271625.27
  Channel: SIP/olle-01843c00
  Channeltype: SIP
  SIPcallid: NTQzYWFiOWM4NmE0MWRkZjExMzU2YzQ3OWQwNzg3ZmI.
  SIPfullcontact: sip:olle@127.0.0.1:49054

- Event: NewAccountCode
 Modules: cdr.c
 Purpose: To report a change in account code for a live channel
 Example:
  Event: NewAccountCode
  Privilege: call,all
  Channel: SIP/olle-01844600
  Uniqueid: 1177530895.2
  AccountCode: Stinas account 1234848484
  OldAccountCode: OllesAccount 12345

- Event: ModuleLoadReport
 Modules: loader.c
 Purpose: To report that module loading is complete. Some aggressive
  clients connect very quickly to AMI and needs to know when
  all manager events embedded in modules are loaded
  Also, if this does not happen, something is seriously wrong.
  This could happen to chan_sip and other modules using DNS.
 Example:
  Event: ModuleLoad
  ModuleLoadStatus: Done
  ModuleSelection: All
  ModuleCount: 24
```

```
- Event: QueueSummary
 Modules: app_queue
 Purpose: To report a summary of queue information. This event is
generated by
   issuing a QueueSummary AMI action.
 Example:
  Event: QueueSummary
  Queue: Sales
  LoggedIn: 12
  Available: 5
  Callers: 10
  HoldTime: 47
 If an actionID was specified for the QueueSummary action, it will
be appended as the
 last line of the QueueSummary event.

- Event: AgentRingNoAnswer
 Modules: app_queue
 Purpose: Reports when a queue member was rung but there was no
answer.
 Example:
  Event: AgentRingNoAnswer
  Queue: Support
  Uniqueid: 1177530895.2
  Channel: SIP/1000-53aee458
  Member: SIP/1000
  MemberName: Thaddeus McClintock
  Ringtime: 10

- Event: RegistryEntry
 Modules: chan_sip
 Purpose: Reports the state of the SIP registrations. This event is
generated by
                issuing a QueueSummary AMI action.
  The RegistrationTime header is expressed as epoch.
 Example:
  Event: RegistryEntry
  Host: sip.myvoipprovider.com
  Port: 5060
  Username: guestuser
  Refresh: 105
  State: Registered
  RegistrationTime: 1219161830
 If an actionID was specified for the SipShowRegistry action, it
will be appended as the
 last line of the RegistrationsComplete event.
```

```
- Event: ChanSpyStart
 Modules: app_chanspy
 Purpose: Reports when an active channel starts to be monitored by
someone.
 Example:
  Event: ChanSpyStart
  SpyerChannel: SIP/4321-13bba124
  SpyeeChannel: SIP/1234-56ecc098

- Event: ChanSpyStop
 Modules: app_chanspy
 Purpose: Reports when an active channel stops to be monitored by
someone.
 Example:
```

```
Event: ChanSpyStop
SpyeeChannel: SIP/1234-56ecc098
```

TODO

...

Building Queues

Building Queues

Written by: Leif Madsen
Initial version: 2010-01-14

In this article, we'll look at setting up a pair of queues in Asterisk called 'sales' and 'support'. These queues can be logged into by queue members, and those members will also have the ability to pause and unpause themselves.

All configuration will be done in flat files on the system in order to maintain simplicity in configuration.

Note that this documentation is based on Asterisk 1.6.2, and this is just one approach to creating queues and the dialplan logic. You may create a better way, and in that case, I would encourage you to submit it to the Asterisk issue tracker at http://issues.asterisk.org for inclusion in Asterisk.

Adding SIP Devices to Your Server

The first thing we want to do is register a couple of SIP devices to our server. These devices will be our agents that can login and out of the queues we'll create later. Our naming convention will be to use MAC addresses as we want to abstract the concepts of user (agent), device, and extension from each other.

In sip.conf, we add the following to the bottom of our file:

```
sip.conf
--------

[std-device](!)
type=peer
context=devices
host=dynamic
secret=s3CuR#p@s5
dtmfmode=rfc2833
disallow=all
allow=ulaw

[0004f2040001](std-device)


[0004f2040002](std-device)
```

What we're doing here is creating a [std-device] template and applying it to a pair of peers that we'll register as 0004f2040001 and 0004f2040002; our devices.

Then our devices can register to Asterisk. In my case I have a hard phone and a soft phone registered. I can verify their connectivity by running 'sip show peers'.

```
*CLI> sip show peers
Name/username              Host            Dyn Nat ACL Port
Status
0004f2040001/0004f2040001  192.168.128.145  D          5060
Unmonitored
0004f2040002/0004f2040002  192.168.128.126  D          5060
Unmonitored
2 sip peers [Monitored: 0 online, 0 offline Unmonitored: 2 online, 0
offline]
```

Configuring Device State

Next, we need to configure our system to track the state of the devices. We do this by defining a 'hint' in the dialplan which creates the ability for a device subscription to be retained in memory. By default we can see there are no hints registered in our system by running the 'core show hints' command.

```
*CLI> core show hints
There are no registered dialplan hint
```

We need to add the devices we're going to track to the extensions.conf file under the [default] context which is the default configuration in sip.conf, however we can change this to any context

we want with the 'subscribecontext'
option.

Add the following lines to extensions.conf:

```
[default]
exten => 0004f2040001,hint,SIP/0004f2040001
exten => 0004f2040002,hint,SIP/0004f2040002
```

Then perform a 'dialplan reload' in order to reload the dialplan.

After reloading our dialplan, you can see the status of the devices with 'core show hints' again.

```
*CLI> core show hints

    -= Registered Asterisk Dial Plan Hints =-
            0004f2040002@default              : SIP/0004f2040002
State:Idle              Watchers  0
            0004f2040001@default              : SIP/0004f2040001
State:Idle              Watchers  0
----------------
- 2 hints registered
```

At this point, create an extension that you can dial that will play a prompt that is long enough for you to go back to the Asterisk console to check the state of your device while it is in use.

To do this, add the 555 extension to the [devices] context and make it playback the tt-monkeys file.

```
extensions.conf
---------------

[devices]
exten => 555,1,Playback(tt-monkeys)
```

Dial that extension and then check the state of your device on the console.

```
*CLI>   == Using SIP RTP CoS mark 5
     -- Executing [555@devices:1]
Playback("SIP/0004f2040001-00000001", "tt-monkeys") in new stack
     -- <SIP/0004f2040001-00000001> Playing 'tt-monkeys.slin'
(language 'en')


*CLI> core show hints

     -= Registered Asterisk Dial Plan Hints =-
             0004f2040002@default               : SIP/0004f2040002
State:Idle            Watchers  0
             0004f2040001@default               : SIP/0004f2040001
State:Idle            Watchers  0
----------------
- 2 hints registered
```

Aha, we're not getting the device state correctly. There must be something else we need to configure.

In sip.conf, we need to enable 'callcounter' in order to activate the ability for Asterisk to monitor whether the device is in use or not. In versions prior to 1.6.0 we needed to use 'call-limit' for this functionality, but call-limit is now deprecated and is no longer necessary.

So, in sip.conf, in our [std-device] template, we need to add the callcounter option.

```
sip.conf
--------


[std-device](!)
type=peer
context=devices
host=dynamic
secret=s3CuR#p@s5
dtmfmode=rfc2833
disallow=all
allow=ulaw
callcounter=yes      ; <-- add this
```

Then reload chan_sip with 'sip reload' and perform our 555 test again. Dial 555 and then check the device state with 'core show hints'.

```
*CLI>    == Using SIP RTP CoS mark 5
    -- Executing [555@devices:1]
Playback("SIP/0004f2040001-00000002", "tt-monkeys") in new stack
    -- <SIP/0004f2040001-00000002> Playing 'tt-monkeys.slin'
(language 'en')


*CLI> core show hints

    -= Registered Asterisk Dial Plan Hints =-
            0004f2040002@default               : SIP/0004f2040002
State:Idle            Watchers  0
            0004f2040001@default               : SIP/0004f2040001
State:InUse           Watchers  0
----------------
- 2 hints registered
```

Note that now we have the correct device state when extension 555 is dialed, showing that our device is InUse after dialing extension 555. This is important when creating queues, otherwise our queue members would get multiple calls from the queues.

Adding Queues to Asterisk

The next step is to add a couple of queues to Asterisk that we can assign queue members into. For now we'll work with two queues; sales and support. Lets create those queues now in queues.conf.

We'll leave the default settings that are shipped with queues.conf.sample in the [general] section of queues.conf. See the queues.conf.sample file for more information about each of the available options.

```
queues.conf
--------

[general]
persistantmembers=yes
autofill=yes
monitor-type=MixMonitor
shared_lastcall=no
```

We can then define a [queue_template] that we'll assign to each of the queues we create. These definitions can be overridden by each queue individually if you reassign them under the [sales] or [support] headers. So under the [general]
section of your queues.conf file, add the following.

```
queues.conf
----------


[queue_template](!)
musicclass=default        ; play [default] music
strategy=rrmemory         ; use the Round Robin Memory strategy
joinempty=yes             ; join the queue when no members available
leavewhenempty=no         ; don't leave the queue no members available
ringinuse=no              ; don't ring members when already InUse


[sales](queue_template)
; Sales queue


[support](queue_template)
; Support queue
```

After defining our queues, lets reload our app_queue.so module.

```
*CLI> module reload app_queue.so
    -- Reloading module 'app_queue.so' (True Call Queueing)

  == Parsing '/etc/asterisk/queues.conf':    == Found
```

Then verify our queues loaded with 'queue show'.

```
*CLI> queue show
support       has 0 calls (max unlimited) in 'rrmemory' strategy (0s
holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
   No Members
   No Callers

sales         has 0 calls (max unlimited) in 'rrmemory' strategy (0s
holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
   No Members
   No Callers
```

Adding Queue Members

You'll notice that we have no queue members available to take calls from the queues. We can add queue members from the Asterisk CLI with the 'queue add member' command.

This is the format of the 'queue add member' command:

```
Usage: queue add member <channel> to <queue> [[[penalty <penalty>]
as <membername>] state_interface <interface>]
        Add a channel to a queue with optionally:  a penalty,
membername and a state_interface
```

The penalty, membername, and state_interface are all optional values. Special attention should be brought to the 'state_interface' option for a member though. The reason for state_interface is that if you're using a channel that does not have device state itself (for example, if you were using the Local channel to deliver a call to an end point) then you could assign the device state of a SIP device to the pseudo channel. This allows the state of a SIP device to be applied to the Local channel for correct device state information.

Lets add our device located at SIP/0004f2040001

```
*CLI> queue add member SIP/0004f2040001 to sales
Added interface 'SIP/0004f2040001' to queue 'sales'
```

Then lets verify our member was indeed added.

```
*CLI> queue show sales
sales          has 0 calls (max unlimited) in 'rrmemory' strategy (0s
holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
    Members:
        SIP/0004f2040001 (dynamic) (Not in use) has taken no calls yet
    No Callers
```

Now, if we dial our 555 extension, we should see that our member becomes InUse within the queue.

```
*CLI>   == Using SIP RTP CoS mark 5
    -- Executing [555@devices:1]
Playback("SIP/0004f2040001-00000001", "tt-monkeys") in new stack
    -- <SIP/0004f2040001-00000001> Playing 'tt-monkeys.slin'
(language 'en')


*CLI> queue show sales
sales          has 0 calls (max unlimited) in 'rrmemory' strategy (0s
holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
    Members:
        SIP/0004f2040001 (dynamic) (In use) has taken no calls yet
    No Callers
```

We can also remove our members from the queue using the 'queue remove' CLI command.

```
*CLI> queue remove member SIP/0004f2040001 from sales
Removed interface 'SIP/0004f2040001' from queue 'sales'
```

Because we don't want to have to add queue members manually from the CLI, we should create a method that allows queue members to login and out from their devices. We'll do that in the next section.

But first, lets add an extension to our dialplan in order to permit people to dial into our queues so calls can be delivered to our queue members.

```
extensions.conf
---------------

[devices]
exten => 555,1,Playback(tt-monkeys)

exten => 100,1,Queue(sales)

exten => 101,1,Queue(support)
```

Then reload the dialplan, and try calling extension 100 from SIP/0004f2040002, which is the device we have not logged into the queue.

```
*CLI> dialplan reload
```

And now we call the queue at extension 100 which will ring our device at SIP/0004f2040001.

```
*CLI>   == Using SIP RTP CoS mark 5
    -- Executing [100@devices:1] Queue("SIP/0004f2040002-00000005",
"sales") in new stack
    -- Started music on hold, class 'default', on
SIP/0004f2040002-00000005
  == Using SIP RTP CoS mark 5
    -- SIP/0004f2040001-00000006 is ringing
```

We can see the device state has changed to Ringing while the device is ringing.

```
*CLI> queue show sales
sales          has 1 calls (max unlimited) in 'rrmemory' strategy (2s
holdtime, 3s talktime), W:0, C:1, A:1, SL:0.0% within 0s
    Members:
       SIP/0004f2040001 (dynamic) (Ringing) has taken 1 calls (last
was 14 secs ago)
    Callers:
       1. SIP/0004f2040002-00000005 (wait: 0:03, prio: 0)
```

Our queue member then answers the phone.

```
*CLI>      -- SIP/0004f2040001-00000006 answered
SIP/0004f2040002-00000005
     -- Stopped music on hold on SIP/0004f2040002-00000005
     -- Native bridging SIP/0004f2040002-00000005 and
SIP/0004f2040001-00000006
```

And we can see the queue member is now in use.

```
*CLI> queue show sales
sales          has 0 calls (max unlimited) in 'rrmemory' strategy (3s
holdtime, 3s talktime), W:0, C:1, A:1, SL:0.0% within 0s
    Members:
       SIP/0004f2040001 (dynamic) (In use) has taken 1 calls (last
was 22 secs ago)
    No Callers
```

Then the call is hung up.

```
*CLI>   == Spawn extension (devices, 100, 1) exited non-zero on
'SIP/0004f2040002-00000005'
```

And we see that our queue member is available to take another call.

```
*CLI> queue show sales
sales          has 0 calls (max unlimited) in 'rrmemory' strategy (3s
holdtime, 4s talktime), W:0, C:2, A:1, SL:0.0% within 0s
    Members:
       SIP/0004f2040001 (dynamic) (Not in use) has taken 2 calls
(last was 6 secs ago)
    No Callers
```

Logging In and Out of Queues

In this section we'll show how to use the AddQueueMember() and RemoveQueueMember() dialplan applications to login and out of queues. For more information about the available options to AddQueueMember() and RemoveQueueMember() use the 'core show application <app>' command from the CLI.

The following bit of dialplan is a bit long, but stick with it, and you'll see that it isn't really all that bad. The gist of the dialplan is that it will check to see if the active user (the device that is dialing the extension) is currently logged into the queue extension that has been requested, and if logged in, then will log them out; if not logged in, then they will be logged into the queue.

We've updated the two lines we added in the previous section that allowed us to dial the sales and support queues. We've abstracted this out a bit in order to make it easier to add new queues in the future. This is done by adding the queue
names to a global variable, then utilizing the extension number dialed to look up the queue name.

So we replace extension 100 and 101 with the following dialplan.

```
; Call any of the queues we've defined in the [globals] section.
exten => _1XX,1,Verbose(2,Call queue as configured in the
QUEUE_${EXTEN} global variable)
exten => _1XX,n,Set(thisQueue=${GLOBAL(QUEUE_${EXTEN})})
exten => _1XX,n,GotoIf($["${thisQueue}" = ""]?invalid_queue,1)
exten => _1XX,n,Verbose(2, --> Entering the ${thisQueue} queue)
exten => _1XX,n,Queue(${thisQueue})
exten => _1XX,n,Hangup()

exten => invalid_queue,1,Verbose(2,Attempted to enter invalid queue)
exten => invalid_queue,n,Playback(silence/1&invalid)
exten => invalid_queue,n,Hangup()
```

The [globals] section contains the following two global variables.

```
[globals]
QUEUE_100=sales
QUEUE_101=support
```

So when we dial extension 100, it matches our pattern _1XX. The number we dialed (100) is then retrievable via ${EXTEN} and we can get the name of queue 100 (sales) from the global variable QUEUE_100. We then assign it to the channel variable thisQueue so it is easier to work with in our dialplan.

```
exten => _1XX,n,Set(thisQueue=${GLOBAL(QUEUE_${EXTEN})})
```

We then check to see if we've gotten a value back from the global variable which would indicate whether the queue was valid or not.

```
exten => _1XX,n,GotoIf($["${thisQueue}" = ""]?invalid_queue,1)
```

If ${thisQueue} returns nothing, then we Goto the invalid_queue extension and playback the 'invalid' file.

We could alternatively limit our pattern match to only extension 100 and 101 with the _10[0-1] pattern instead.

Lets move into the nitty-gritty section and show how we can login and logout our devices to the pair of queues we've created.

First, we create a pattern match that takes star ⭐ plus the queue number that we want to login or logout of. So to login/out of the sales queue (100) we would dial *100. We use the same extension for logging in and out.

```
; Extension *100 or *101 will login/logout a queue member from sales
or support queues respectively.
exten => _*10[0-1],1,Set(xtn=${EXTEN:1})                    ;
save ${EXTEN} with * chopped off to ${xtn}
exten => _*10[0-1],n,Goto(queueLoginLogout,member_check,1)      ;
check if already logged into a queue
```

We save the value of ${EXTEN:1} to the 'xtn' channel variable so we don't need to keep typing the complicated pattern match.

Now we move into the meat of our login/out dialplan inside the [queueLoginLogout] context.

The first section is initializing some variables that we need throughout the member_check extension such as the name of the queue, the members currently logged into the queue, and the current device peer name (i.e. SIP/0004f2040001).

```
; ### Login or Logout a Queue Member
[queueLoginLogout]
exten => member_check,1,Verbose(2,Logging queue member in or out of
the request queue)
exten => member_check,n,Set(thisQueue=${GLOBAL(QUEUE_${xtn})})
; assign queue name to a variable
exten =>
member_check,n,Set(queueMembers=${QUEUE_MEMBER_LIST(${thisQueue})})
; assign list of logged in members of thisQueue to

; a variable (comma separated)
exten =>
member_check,n,Set(thisActiveMember=SIP/${CHANNEL(peername)})
; initialize 'thisActiveMember' as current device

exten => member_check,n,GotoIf($["${queueMembers}" = ""]?q_login,1)
; short circuit to logging in if we don't have

; any members logged into this queue
```

At this point if there are no members currently logged into our sales queue, we then short-circuit
our dialplan to go to the 'q_login' extension since there is no point in wasting cycles searching to
see if we're already logged in.

The next step is to finish initializing some values we need within the While() loop that we'll use to
check if we're already logged into the queue. We set our ${field} variable to 1, which will be used
as the field number offset in the CUT() function.

```
; Initialize some values we'll use in the While() loop
exten => member_check,n,Set(field=1)
; start our field counter at one
exten => member_check,n,Set(logged_in=0)
; initialize 'logged_in' to "not logged in"
exten =>
member_check,n,Set(thisQueueMember=${CUT(queueMembers,\,,${field})})
; initialize 'thisQueueMember' with the value in the

;   first field of the comma-separated list
```

Now we get to enter our While() loop to determine if we're already logged in.

```
; Enter our loop to check if our member is already logged into this
queue
exten => member_check,n,While($[${EXISTS(${thisQueueMember})}])
; while we have a queue member...
```

This is where we check to see if the member at this position of the list is the same as the device we're calling from. If it doesn't match, then we go to the 'check_next' priority label (where we increase our ${field} counter variable). If it does match, then we continue on in the dialplan.

```
exten => member_check,n,GotoIf($["${thisQueueMember}" !=
"${thisActiveMember}"]?check_next)      ; if 'thisQueueMember' is not
the

;    same as our active peer, then

;    check the next in the list of

;    logged in queue members
```

If we continued on in the dialplan, then we set the ${logged_in} channel variable to '1' which represents we're already logged into this queue. We then exit the While() loop with the ExitWhile() dialplan application.

```
exten => member_check,n,Set(logged_in=1)
; if we got here, set as logged in
exten => member_check,n,ExitWhile()
;   then exit our loop
```

If we didn't match this peer name in the list, then we increase our ${field} counter variable by one, update the ${thisQueueMember} channel variable and then move back to the top of the loop for another round of checks.

```
exten => member_check,n(check_next),Set(field=$[${field} + 1])
; if we got here, increase counter
exten =>
member_check,n,Set(thisQueueMember=${CUT(queueMembers,\,,${field})})
; get next member in the list
exten => member_check,n,EndWhile()
; ...end of our loop
```

And once we exit our loop, we determine whether we need to log our device in or out of the queue.

```
; if not logged in, then login to this queue, otherwise, logout
exten => member_check,n,GotoIf($[${logged_in} =
0]?q_login,1:q_logout,1)          ; if not logged in, then login,
otherwise, logout
```

The following two extensions are used to either log the device in or out of the queue. We use the AddQueueMember() and RemovQueueMember() applications to login or logout the device from the queue.

The first two arguments for AddQueueMember() and RemoveQueueMember() are 'queue' and 'device'. There are additional arguments we can pass, and you can check those out with 'core show application AddQueueMember' and 'core show application RemoveQueueMember()'.

```
; ### Login queue member ###
exten => q_login,1,Verbose(2,Logging ${thisActiveMember} into the
${thisQueue} queue)
exten => q_login,n,AddQueueMember(${thisQueue},${thisActiveMember})
; login our active device to the queue

; requested
exten => q_login,n,Playback(silence/1)  ; answer the channel by
playing one second of silence

; If the member was added to the queue successfully, then playback
"Agent logged in", otherwise, state an error occurred
exten => q_login,n,ExecIf($["${AQMSTATUS}" =
"ADDED"]?Playback(agent-loginok):Playback(an-error-has-occurred))
exten => q_login,n,Hangup()


; ### Logout queue member ###
exten => q_logout,1,Verbose(2,Logging ${thisActiveMember} out of
${thisQueue} queue)
exten =>
q_logout,n,RemoveQueueMember(${thisQueue},${thisActiveMember})
exten => q_logout,n,Playback(silence/1)
exten => q_logout,n,ExecIf($["${RQMSTATUS}" =
"REMOVED"]?Playback(agent-loggedoff):Playback(an-error-has-occurred))
=> q_logout,n,Hangup()
```

And that's it! Give it a shot and you should see console output similar to the following which will login and logout your queue members to the queues you've configured.

You can see there are already a couple of queue members logged into the sales queue.

```
*CLI> queue show sales
sales          has 0 calls (max unlimited) in 'rrmemory' strategy (3s
holdtime, 4s talktime), W:0, C:2, A:1, SL:0.0% within 0s
    Members:
        SIP/0004f2040001 (dynamic) (Not in use) has taken no calls yet
        SIP/0004f2040002 (dynamic) (Not in use) has taken no calls yet
    No Callers
```

Then we dial *100 to logout the active device from the sales queue.

```
*CLI>   == Using SIP RTP CoS mark 5
    -- Executing [*100@devices:1] Set("SIP/0004f2040001-00000012",
"xtn=100") in new stack
    -- Executing [*100@devices:2] Goto("SIP/0004f2040001-00000012",
"queueLoginLogout,member_check,1") in new stack
    -- Goto (queueLoginLogout,member_check,1)
    -- Executing [member_check@queueLoginLogout:1]
Verbose("SIP/0004f2040001-00000012", "2,Logging queue member in or
out of the request queue") in new stack
  == Logging queue member in or out of the request queue
    -- Executing [member_check@queueLoginLogout:2]
Set("SIP/0004f2040001-00000012", "thisQueue=sales") in new stack
    -- Executing [member_check@queueLoginLogout:3]
Set("SIP/0004f2040001-00000012",
"queueMembers=SIP/0004f2040001,SIP/0004f2040002") in new stack
    -- Executing [member_check@queueLoginLogout:4]
Set("SIP/0004f2040001-00000012",
"thisActiveMember=SIP/0004f2040001") in new stack
    -- Executing [member_check@queueLoginLogout:5]
GotoIf("SIP/0004f2040001-00000012", "0?q_login,1") in new stack
    -- Executing [member_check@queueLoginLogout:6]
Set("SIP/0004f2040001-00000012", "field=1") in new stack
    -- Executing [member_check@queueLoginLogout:7]
Set("SIP/0004f2040001-00000012", "logged_in=0") in new stack
    -- Executing [member_check@queueLoginLogout:8]
Set("SIP/0004f2040001-00000012", "thisQueueMember=SIP/0004f2040001")
in new stack
    -- Executing [member_check@queueLoginLogout:9]
While("SIP/0004f2040001-00000012", "1") in new stack
    -- Executing [member_check@queueLoginLogout:10]
GotoIf("SIP/0004f2040001-00000012", "0?check_next") in new stack
    -- Executing [member_check@queueLoginLogout:11]
Set("SIP/0004f2040001-00000012", "logged_in=1") in new stack
    -- Executing [member_check@queueLoginLogout:12]
```

```
ExitWhile("SIP/0004f2040001-00000012", "") in new stack
    -- Jumping to priority 15
    -- Executing [member_check@queueLoginLogout:16]
GotoIf("SIP/0004f2040001-00000012", "0?q_login,1:q_logout,1") in new
stack
    -- Goto (queueLoginLogout,q_logout,1)
    -- Executing [q_logout@queueLoginLogout:1]
Verbose("SIP/0004f2040001-00000012", "2,Logging SIP/0004f2040001 out
of sales queue") in new stack
  == Logging SIP/0004f2040001 out of sales queue
    -- Executing [q_logout@queueLoginLogout:2]
RemoveQueueMember("SIP/0004f2040001-00000012",
"sales,SIP/0004f2040001") in new stack
[Nov 12 12:08:51] NOTICE[11582]: app_queue.c:4842 rqm_exec: Removed
interface 'SIP/0004f2040001' from queue 'sales'
    -- Executing [q_logout@queueLoginLogout:3]
Playback("SIP/0004f2040001-00000012", "silence/1") in new stack
    -- <SIP/0004f2040001-00000012> Playing 'silence/1.slin'
(language 'en')
    -- Executing [q_logout@queueLoginLogout:4]
ExecIf("SIP/0004f2040001-00000012",
"1?Playback(agent-loggedoff):Playback(an-error-has-occurred)") in
new stack
    -- <SIP/0004f2040001-00000012> Playing 'agent-loggedoff.slin'
(language 'en')
    -- Executing [q_logout@queueLoginLogout:5]
```

```
Hangup("SIP/0004f2040001-00000012", "") in new stack
   == Spawn extension (queueLoginLogout, q_logout, 5) exited non-zero
on 'SIP/0004f2040001-00000012'
```

And we can see that the device we loggd out by running 'queue show sales'.

```
*CLI> queue show sales
sales          has 0 calls (max unlimited) in 'rrmemory' strategy (3s
holdtime, 4s talktime), W:0, C:2, A:1, SL:0.0% within 0s
    Members:
        SIP/0004f2040002 (dynamic) (Not in use) has taken no calls yet
    No Callers
```

Pausing and Unpausing Members of Queues

Once we have our queue members logged in, it is inevitable that they will want to pause themselves during breaks, and other short periods of inactivity. To do this we can utilize the 'queue pause' and 'queue unpause' CLI commands.

We have two devices logged into the sales queue as we can see with the 'queue show sales' CLI command.

```
*CLI> queue show sales
sales          has 0 calls (max unlimited) in 'rrmemory' strategy (0s
holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
    Members:
        SIP/0004f2040002 (dynamic) (Not in use) has taken no calls yet
        SIP/0004f2040001 (dynamic) (Not in use) has taken no calls yet
    No Callers
```

We can then pause our devices with 'queue pause' which has the following format.

```
Usage: queue {pause|unpause} member <member> [queue <queue> [reason
<reason>]]
 Pause or unpause a queue member. Not specifying a particular queue
 will pause or unpause a member across all queues to which the
member
 belongs.
```

Lets pause device 0004f2040001 in the sales queue by executing the following.

```
*CLI> queue pause member SIP/0004f2040001 queue sales
paused interface 'SIP/0004f2040001' in queue 'sales' for reason
'lunch'
```

And we can see they are paused with 'queue show sales'.

```
*CLI> queue show sales
sales         has 0 calls (max unlimited) in 'rrmemory' strategy (0s
holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
    Members:
        SIP/0004f2040002 (dynamic) (Not in use) has taken no calls yet
        SIP/0004f2040001 (dynamic) (paused) (Not in use) has taken no
calls yet
    No Callers
```

At this point the queue member will no longer receive calls from the system. We can unpause them with the CLI command 'queue unpause member'.

```
*CLI> queue unpause member SIP/0004f2040001 queue sales
unpaused interface 'SIP/0004f2040001' in queue 'sales'
```

And if you don't specify a queue, it will pause or unpause from all queues.

```
*CLI> queue pause member SIP/0004f2040001
paused interface 'SIP/0004f2040001'
```

Of course we want to allow the agents to pause and unpause themselves from their devices, so we need to create an extension and some dialplan logic for that to happen.

Below we've created the pattern patch **_0[01]! which will match on *00 and *01, and will *also** match with zero or more digits following it, such as the queue extension number.

So if we want to pause ourselves in all queues, we can dial *00; unpausing can be done with *01. But if our agents just need to pause or unpause themselves from a single queue, then we will also accept *00100 to pause in queue 100 (sales), or we can unpause ourselves from sales with *01100.

```
extensions.conf
---------------


; Allow queue members to pause and unpause themselves from all
queues, or an individual queue.
;
; _*0[01]! pattern match will match on *00 and *01 plus 0 or more
digits.
exten => _*0[01]!,1,Verbose(2,Pausing or unpausing queue member from
one or more queues)
exten => _*0[01]!,n,Set(xtn=${EXTEN:3})
; save the queue extension to 'xtn'
exten => _*0[01]!,n,Set(thisQueue=${GLOBAL(QUEUE_${xtn})})
; get the queue name if available
exten => _*0[01]!,n,GotoIf($[${ISNULL(${thisQueue})} &
${EXISTS(${xtn})}]?invalid_queue,1)  ; if 'thisQueue' is blank and
the

;   the agent dialed a queue exten,

;   we will tell them it's invalid
```

The following line will determine if we're trying to pause or unpause. This is done by taking the value dialed (e.g. *00100) and chopping off the first 2 digits which leaves us with 0100, and then the :1 will return the next digit, which in this case is '0' that we're using to signify that the queue member wants to be paused (in queue 100).

So we're doing the following with our EXTEN variable.

```
${EXTEN:2:1}
offset        ^ ^  length
```

Which causes the following.

```
 *00100
  ^^       offset these characters


 *00100
   ^       then return a digit length of one, which is digit 0
```

```
exten => _*0[01]!,n,GotoIf($[${EXTEN:2:1} = 0]?pause,1:unpause,1)
; determine if they wanted to pause

;   or to unpause.
```

The following two extensions, pause & unpause, are used for pausing and unpausing our extension from the queue(s). We use the PauseQueueMember() and UnpauseQueueMember() dialplan applications which accept the queue name (optional) and the queue member name. If the queue name is not provided, then it is assumed we want to pause or unpause from all logged in queues.

```
; Unpause ourselves from one or more queues
exten => unpause,1,NoOp()
exten =>
unpause,n,UnpauseQueueMember(${thisQueue},SIP/${CHANNEL(peername)})
; if 'thisQueue' is populated we'll pause in

;   that queue, otherwise, we'll unpause in

;   in all queues
```

Once we've unpaused ourselves, we use GoSub() to perform some common dialplan logic that is used for pausing and unpausing. We pass three arguments to the subroutine:

- variable name that contains the result of our operation
- the value we're expecting to get back if successful
- the filename to play

```
exten =>
unpause,n,GoSub(changePauseStatus,start,1(UPQMSTATUS,UNPAUSED,availabl
; use the changePauseStatus subroutine and

;   pass the values for: variable to check,

;   value to check for, and file to play
exten => unpause,n,Hangup()
```

And the same method is done for pausing.

```
; Pause ourselves in one or more queues
exten => pause,1,NoOp()
exten =>
pause,n,PauseQueueMember(${thisQueue},SIP/${CHANNEL(peername)})
exten =>
pause,n,GoSub(changePauseStatus,start,1(PQMSTATUS,PAUSED,unavailable)
=> pause,n,Hangup()
```

Lets explore what happens in the subroutine we're using for pausing and unpausing.

```
; ### Subroutine we use to check pausing and unpausing status ###
[changePauseStatus]
; ARG1:  variable name to check, such as PQMSTATUS and UPQMSTATUS
(PauseQueueMemberStatus / UnpauseQueueMemberStatus)
; ARG2:  value to check for, such as PAUSED or UNPAUSED
; ARG3:  file to play back if our variable value matched the value
to check for
;
exten => start,1,NoOp()
exten => start,n,Playback(silence/1)
; answer line with silence
```

The following line is probably the most complex. We're using the IF() function inside the Playback() application which determines which file to playback to the user.

Those three values we passed in from the pause and unpause extensions could have been something like:

- ARG1 – PQMSTATUS
- ARG2 – PAUSED
- ARG3 – unavailable

So when expanded, we'd end up with the following inside the IF() function.

```
$["${PQMSTATUS}" = "PAUSED"]?unavailable:not-yet-connected
```

${PQMSTATUS} would then be expanded further to contain the status of our PauseQueueMember() dialplan application, which could either be PAUSED or NOTFOUND. So if ${PQMSTATUS} returned PAUSED, then it would match what we're looking to match on, and we'd then return 'unavailable' to Playback() that would tell the user they are now unavailable.

Otherwise, we'd get back a message saying "not yet connected" to indicate they are likely not logged into the queue they are attempting to change status in.

```
; Please note that ${ARG1} is wrapped in ${  } in order to expand
the value of ${ARG1} into
;   the variable we want to retrieve the value from, i.e. ${${ARG1}}
turns into ${PQMSTATUS}
exten => start,n,Playback(${IF($["${${ARG1}}" =
"${ARG2}"]?${ARG3}:not-yet-connected)})          ; check if value of
variable

;   matches the value we're looking

;   for and playback the file we want

;   to play if it does
```

If ${xtn} is null, then we just go to the end of the subroutine, but if it isn't then we will play back "in the queue" followed by the queue extension number indicating which queue they were (un)paused from.

```
exten => start,n,GotoIf($[${ISNULL(${xtn})}]?end)        ; if ${xtn}
is null, then just Return()
exten => start,n,Playback(in-the-queue)                  ;   if not
null, then playback "in the queue"
exten => start,n,SayNumber(${xtn})                       ;   and the
queue number that we (un)paused from
exten => start,n(end),Return()                           ; return
from were we came
```

Conclusion

You should now have a simple system that permits you to login and out of queues you create in queues.conf, and to allow queue members to pause themselves within one or more queues. There are a lot of dialplan concepts utilized in this
article, so you are encouraged to seek out additional documentation if any of these concepts are a bit fuzzy for you.

A good start is the doc/ subdirectory of the Asterisk sources, or the various configuration samples files located in the configs/ subdirectory of your Asterisk source code.

Call Completion Supplementary Services

Placeholder page for CCSS information.

Call Queues

Template holder while we wait for input on a good README for call queues. Please open a bug report and add text to this document

General advice on the agent channel Using dynamic queue members SIP channel configuration

Queues depend on the channel driver reporting the proper state for each member of the queue. To get proper signalling on
queue members that use the SIP channel driver, you need to enable a call limit (could be set to a high value so it
is not put into action) and also make sure that both inbound and outbound calls are accounted for.

Example:

```
[general]
limitonpeer = yes

[peername]
type=friend
call-limit=10
```

Other references

- queuelog.txt
- queues-with-callback-members.txt

(Should we merge those documents into this?)

Channel Drivers

placeholder page to store the various channel driver subpages

chan_unistim

Unified Networks IP Stimulus (UNIStim) Channel Driver for Asterisk

This is a channel driver for Unistim protocol. You can use a least a Nortel i2002, i2004 and i2050.

Following features are supported : Send/Receive CallerID, Redial, SoftKeys, SendText(), Music On Hold, Message Waiting Indication (MWI), Distinctive ring, Transfer, Threeway call, History, Forward, Dynamic SoftKeys.

How to configure the i2004

1. Power on the phone
2. Wait for message "Nortel Networks"
3. Press quickly the four buttons just below the LCD screen, in sequence from left to right
4. If you see "Locating server", power off or reboot the phone and try again
5. DHCP : 0
6. SET IP : a free ip of your network
7. NETMSK / DEF GW : netmask and default gateway
8. S1 IP : ip of the asterisk server
9. S1 PORT : 5000
10. S1 ACTION : 1
11. S1 RETRY COUNT : 10
12. S2 : same as S1

The line=> entry in unistim.conf does not add an extension in asterisk by default. If you want to do that, add extension=line in your phone context.

If you have this entry on unistim.conf :

```
[violet]
device=006038abcdef
line => 102
```

then use:

```
exten => 2100,1,Dial(USTM/102@violet)
```

You can display a text with :

```
exten => 555,1,SendText(Sends text to client. Greetings)
```

Rebooting a Nortel phone

- Press mute,up,down,up,down,up,mute,9,release(red button)

Distinctive ring

1. You need to append /r to the dial string.
2. The first digit must be from 0 to 7 (inclusive). It's the 'melody' selection.
3. The second digit (optional) must be from 0 to 3 (inclusive). It's the ring volume. 0 still produce a sound.

Select the ring style #1 and the default volume :

```
exten => 2100,1,Dial(USTM/102@violet/r1)
```

Select the ring style #4 with a very loud volume :

```
exten => 2100,1,Dial(USTM/102@violet/r43)
```

Country code

- You can use the following codes for country= (used for dial tone) - us fr au nl uk fi es jp no at nz tw cl se be sg il br hu lt pl za pt ee mx in de ch dk cn

- If you want a correct ring, busy and congestion tone, you also need a valid entry in indications.conf and check if res_indications.so is loaded.

- language= is also supported but it's only used by Asterisk (for more information see http://www.voip-info.org/wiki/view/Asterisk+multi-language). The end user interface of the phone will stay in english.

Bookmarks, Softkeys

Layout

```
|--------------------|
|   5            2    |
|   4            1    |
|   3            0    |
```

- When the second letter of bookmark= is @, then the first character is used for positioning this entry
- If this option is omitted, the bookmark will be added to the next available sofkey
- Also work for linelabel (example : linelabel="5@Line 123")
- You can change a softkey programmatically with SendText(@position@icon@label@extension) ex: SendText(@1@55@Stop Forwd@908)

Autoprovisioning

- This feature must only be used on a trusted network. It's very insecure : all unistim phones will be able to use your asterisk pbx.
- You must add an entry called [template]. Each new phones will be based on this profile.
- You must set a least line=>. This value will be incremented when a new phone is registered. device= must not be specified. By default, the phone will asks for a number. It will be added into the dialplan. Add extension=line for using the generated line number instead.

Example :

```
[general]
port=5000
autoprovisioning=yes


[template]
line => 100
bookmark=Support@123  ; Every phone will have a softkey Support
```

- If a first phone have a mac = 006038abcdef, a new device named USTM/100@006038abcdef will be created.
- If a second phone have a mac = 006038000000, it will be named USTM/101@006038000000 and so on.

- When autoprovisioning=tn, new phones will ask for a tn, if this number match a tn= entry in a device, this phone will be mapped into.

Example:

```
[black]
tn=1234
line => 100
```

- If a user enter TN 1234, the phone will be known as USTM/100@black.

History

- Use the two keys located in the middle of the Fixed feature keys row (on the bottom of the phone) to enter call history.

- By default, chan_unistim add any incoming and outgoing calls in files (/var/log/asterisk/unistimHistory). It can be a privacy issue, you can disable this feature by adding callhistory=0. If history files were created, you also need to delete them. callhistory=0 will NOT disable

normal asterisk CDR logs.

Forward

- This feature requires chan_local (loaded by default)

Generic asterisk features

You can use the following entries in unistim.conf

- Billing - accountcode amaflags
- Call Group - callgroup pickupgroup (untested)
- Music On Hold - musiconhold
- Language - language (see section Coutry Code)
- RTP NAT - nat (control ast_rtp_setnat, default = 0. Obscure behaviour)

Trunking

- It's not possible to connect a Nortel Succession/Meridian/BCM to Asterisk via chan_unistim. Use either E1/T1 trunks, or buy UTPS (UNISTIM Terminal Proxy Server) from Nortel.

Wiki, Additional infos, Comments :

- http://www.voip-info.org/wiki-Asterisk+UNISTIM+channels

*BSD :

- Comment #define HAVE_IP_PKTINFO in chan_unistim.c
- Set public_ip with an IP of your computer
- Check if unistim.conf is in the correct directory

Issues

- As always, NAT can be tricky. If a phone is behind a NAT, you should port forward UDP 5000 (or change [general] port= in unistim.conf) and UDP 10000 (or change [yourphone] rtp_port=)

- Only one phone per public IP (multiple phones behind the same NAT don't work). You can either :
  - Setup a VPN
  - Install asterisk inside your NAT. You can use IAX2 trunking if you're master asterisk is outside.
  - If asterisk is behind a NAT, you must set [general] public_ip= with your public IP. If you don't do that or the bindaddr is invalid (or no longer valid, eg dynamic IP), phones should be able to display messages but will be unable to send/receive RTP packets (no sound)
- Don't forget : this work is based entirely on a reverse engineering, so you may encounter compatibility issues. At this time, I know three ways to establish a RTP session. You can modify [yourphone] rtp_method= with 0, 1, 2 or 3. 0 is the default method, should work. 1 can be used on new firmware (black i2004) and 2 on old violet i2004. 3 can be used on black i2004 with chrome.
- If you have difficulties, try unistim debug and set verbose 3 on the asterisk CLI. For extra debug, uncomment #define DUMP_PACKET 1 and recompile chan_unistim.

Database Transactions

As of 1.6.2, Asterisk now supports doing database transactions from the Dialplan. A number of new applications and functions have been introduced for this purpose and this document should hopefully familiarize you with all of them.

First, the ODBC() function has been added which is used to set up all new database transactions. Simply write the name of the transaction to this function, along with the arguments of "transaction" and the database name, e.g. Set(ODBC(transaction,postgres-asterisk)=foo). In

this example, the name of the transaction is "foo". The name doesn't really matter, unless you're manipulating multiple transactions within the same dialplan, at the same time. Then, you use the transaction name to change which transaction is active for the next dialplan function.

The ODBC() function is also used to turn on a mode known as forcecommit. For most cases, you won't need to use this, but it's there. It simply causes a transaction to be committed, when the channel hangs up. The other property which may be set is the isolation property. Please consult with your database vendor as to which values are supported by their ODBC driver. Asterisk supports setting all standard ODBC values, but many databases do not support the entire complement.

Finally, when you have run multiple statements on your transaction and you wish to complete the transaction, use the ODBC_Commit and ODBC_Rollback applications, along with the transaction ID (in the example above, "foo") to commit or rollback the transaction. Please note that if you do not explicitly commit the transaction or if forcecommit is not turned on, the transaction will be automatically rolled back at channel destruction (after hangup) and all related database resources released back to the pool.

Distributed Device State with AIS

1. Introduction

Various changes have been made related to "event handling" in Asterisk. One of the most important things included in these changes is the ability to share certain events between servers. The two types of events that can currently be shared between servers are:

1. **MWI** - *Message Waiting Indication* - This gives you a high performance option for letting servers in a cluster be aware of changes in the state of a mailbox. Instead of having each server have to poll an ODBC database, this lets the server that actually made the change to the mailbox generate an event which will get distributed to the other servers that have subscribed to this information.
2. **Device State** - This lets servers in a local cluster inform each other about changes in the state of a device on that particular server. When the state of a device changes on any server, the overall state of that device across the cluster will get recalculated. So, any subscriptions to the state of a device, such as hints in the dialplan or an application like Queue() which reads device state, will then reflect the state of a device across a cluster.

2. OpenAIS Installation

Description

The current solution for providing distributed events with Asterisk is done by using the AIS (Application Interface Specification), which provides an API for a distributed event service. While this API is standardized, this code has been developed exclusively against the open source implementation of AIS called OpenAIS.

For more information about OpenAIS, visit their web site http://www.openais.org/.

Install Dependencies

- Ubuntu
  - libnss3-dev
- Fedora
  - nss-devel

Download

Download the latest versions of Corosync and OpenAIS from http://www.corosync.org/ and http://www.openais.org/.

Compile and Install

```
$ tar xvzf corosync-1.2.8.tar.gz
$ cd corosync-1.2.8
$ ./configure
$ make
$ sudo make install
```

```
$ tar xvzf openais-1.1.4.tar.gz
$ cd openais-1.1.4
$ ./configure
$ make
$ sudo make install
```

3. OpenAIS Configuration

Basic OpenAIS configuration to get this working is actually pretty easy. Start by copying in a sample configuration file for Corosync and OpenAIS.

```
$ sudo mkdir -p /etc/ais
$ cd openais-1.1.4
$ sudo cp conf/openais.conf.sample /etc/ais/openais.conf
```

```
$ sudo mkdir -p /etc/corosync
$ cd corosync-1.2.8
$ sudo cp conf/corosync.conf.sample /etc/corosync/corosync.conf
```

Now, edit openais.conf using the editor of your choice.

```
$ ${EDITOR:-vim} /etc/ais/openais.conf
```

The only section that you should need to change is the totem - interface section.

<table>
<tr><th colspan="1" style="text-align:center">/etc/ais/openais.conf</th></tr>
</table>

```
totem {
    ...
    interface {
        ringnumber: 0
        bindnetaddr: 10.24.22.144
        mcastaddr: 226.94.1.1
        mcastport: 5405
    }
}
```

The default mcastaddr and mcastport is probably fine. You need to change the bindnetaddr to match the address of the network interface that this node will use to communicate with other nodes in the cluster.

Now, edit /etc/corosync/corosync.conf, as well. The same change will need to be made to the totem-interface section in that file.

4. Running OpenAIS

While testing, I recommend starting the aisexec application in the foreground so that you can see debug messages that verify that the nodes have discovered each other and joined the cluster.

```
$ sudo aisexec -f
```

For example, here is some sample output from the first server after starting aisexec on the second server:

```
Nov 13 06:55:30 corosync [CLM   ] CLM CONFIGURATION CHANGE
Nov 13 06:55:30 corosync [CLM   ] New Configuration:
Nov 13 06:55:30 corosync [CLM   ]       r(0) ip(10.24.22.144)
Nov 13 06:55:30 corosync [CLM   ]       r(0) ip(10.24.22.242)
Nov 13 06:55:30 corosync [CLM   ] Members Left:
Nov 13 06:55:30 corosync [CLM   ] Members Joined:
Nov 13 06:55:30 corosync [CLM   ]       r(0) ip(10.24.22.242)
Nov 13 06:55:30 corosync [TOTEM ] A processor joined or left the
membership and a new membership was formed.
Nov 13 06:55:30 corosync [MAIN  ] Completed service synchronization,
ready to provide service.
```

5. Installing Asterisk

Install Asterisk as usual. Just make sure that you run the configure script after OpenAIS gets installed. That way, it will find the AIS header files and will let you build the res_ais module.

Check menuselect to make sure that res_ais is going to get compiled and installed.

```
$ cd asterisk-source
$ ./configure

$ make menuselect
   ---> Resource Modules
```

If you have existing configuration on the system being used for testing, just be sure to install the addition configuration file needed for res_ais.

```
$ sudo cp configs/ais.conf.sample /etc/asterisk/ais.conf
```

6. Configuring Asterisk

First, ensure that you have a unique "entity ID" set for each server.

```
*CLI> core show settings
   ...
   Entity ID:                    01:23:45:67:89:ab
```

The code will attempt to generate a unique entity ID for you by reading MAC addresses off of a network interface. However, you can also set it manually in the [options] section of asterisk.conf.

```
$ sudo ${EDITOR:-vim} /etc/asterisk/asterisk.conf
```

asterisk.conf
[options]

entity_id=01:23:45:67:89:ab |

Edit the Asterisk ais.conf to enable distributed events. For example, if you would like to enable distributed device state, you should add the following section to the file:

```
$ sudo ${EDITOR:-vim} /etc/asterisk/ais.conf
```

/etc/asterisk/ais.conf

```
[device_state]
type=event_channel
publish_event=device_state
subscribe_event=device_state
```

For more information on the contents and available options in this configuration file, please see the sample configuration file:

```
$ cd asterisk-source
$ less configs/ais.conf.sample
```

7. Basic Testing of Asterisk with OpenAIS

If you have OpenAIS successfully installed and running, as well as Asterisk with OpenAIS support successfully installed, configured, and running, then you are ready to test out some of the AIS functionality in Asterisk.

The first thing to test is to verify that all of the nodes that you think should be in your cluster are actually there. There is an Asterisk CLI command which will list the current cluster members using the AIS Cluster Membership Service
(CLM).

```
*CLI> ais clm show members


================================================================
=== Cluster Members ========================================
================================================================
===
=== --------------------------------------------------------
=== Node Name: 10.24.22.144
=== ==> ID: 0x9016180a
=== ==> Address: 10.24.22.144
=== ==> Member: Yes
=== --------------------------------------------------------
===
=== --------------------------------------------------------
=== Node Name: 10.24.22.242
=== ==> ID: 0xf216180a
=== ==> Address: 10.24.22.242
=== ==> Member: Yes
=== --------------------------------------------------------
===
================================================================
```

✅ If you're having trouble getting the nodes of the cluster to see each other, make sure you do not have firewall rules that are blocking the multicast traffic that is used to communicate amongst the nodes.

The next thing to do is to verify that you have successfully configured some event channels in the Asterisk ais.conf file. This command is related to the event service (EVT), so like the previous command, uses the syntax: `ais <service name> <command>`.

```
*CLI> ais evt show event channels


================================================================
=== Event Channels =========================================
================================================================
===
=== --------------------------------------------------------
=== Event Channel Name: device_state
=== ==> Publishing Event Type: device_state
=== ==> Subscribing to Event Type: device_state
=== --------------------------------------------------------
===
================================================================
```

8. Testing Distributed Device State

The easiest way to test distributed device state is to use the DEVICE_STATE() diaplan function. For example, you could have the following piece of dialplan on every server:

/etc/asterisk/extensions.conf
| ```
[devstate_test]

exten => 1234,hint,Custom:mystate
``` |

Now, you can test that the cluster-wide state of "Custom:mystate" is what you would expect after going to the CLI of each server and adjusting the state.

```
server1*CLI> dialplan set global DEVICE_STATE(Custom:mystate)
NOT_INUSE
    ...

server2*CLI> dialplan set global DEVICE_STATE(Custom:mystate) INUSE
    ...
```

Various combinations of setting and checking the state on different servers can be used to verify that it works as expected. Also, you can see the status of the hint on each server, as well, to see how extension state would reflect the
state change with distributed device state:

```
server2*CLI> core show hints
    -= Registered Asterisk Dial Plan Hints =-
                1234@devstate_test       : Custom:mystate
State:InUse         Watchers   0
```

One other helpful thing here during testing and debugging is to enable debug logging. To do so, enable debug on the console in /etc/asterisk/logger.conf. Also, enable debug at the Asterisk CLI.

```
*CLI> core set debug 1
```

When you have this debug enabled, you will see output during the processing of every device state change. The important thing to look for is where the known state of the device for each server is added together to determine the overall
state.

Distributed Device State with XMPP PubSub

1. Introduction

This document describes installing and utilizing XMPP PubSub events to distribute device state and message waiting indication (MWI) events between servers. The difference between this method and OpenAIS (see Distributed Device State with AIS) is that OpenAIS can only be used in low latency networks; meaning only on the LAN, and not across the internet.

If you plan on distributing device state or MWI across the internet, then you will require the use of XMPP PubSub events.

2. Tigase Installation

Description

Currently the only server supported for XMPP PubSub events is the Tigase open source XMPP/Jabber environment. This is the server that the various Asterisk servers will connect to in order to distribute the events. The Tigase server can even be clustered in order to provide high availability for your device state; however, that is beyond the scope of this document.

For more information about Tigase, visit their web site http://www.tigase.org/.

Download

To download the Tigase environment, get the latest version at http://www.tigase.org/content/tigase-downloads. Some distributions have Tigase packaged, as well.

Install

The Tigase server requires a working Java environment, including both a JRE (Java Runtime Environment) and a JDK (Java Development Kit), currently at least version 1.6.

For more information about how to install Tigase, see the web site http://www.tigase.org/content/quick-start.

2.1. Tigase Configuration

While installing Tigase, be sure you enable the PubSub module. Without it, the PubSub events won't be accepted by the server, and your device state will not be distributed.

There are a couple of things you need to configure in Tigase before you start it in order for Asterisk to connect. The first thing we need to do is generate the self-signed certificate. To do this we use the keytool application. More

information can be found here http://www.tigase.org/content/server-certificate.

Generating the keystore file

Generally, we need to run the following commands to generate a new keystore file.

```
# cd /opt/Tigase-4.3.1-b1858/certs
```

Be sure to change the 'yourdomain' to your domain.

```
# keytool -genkey -alias yourdomain -keystore rsa-keystore -keyalg
RSA -sigalg MD5withRSA
```

The keytool application will then ask you for a password. Use the password 'keystore' as this is the default password that Tigase will use to load the keystore file.

You then need to specify your domain as the first value to be entered in the security certificate.

```
What is your first and last name?
   [Unknown]: asterisk.mydomain.tld
What is the name of your organizational unit?
   [Unknown]:
What is the name of your organization?
   [Unknown]:
What is the name of your City or Locality?
   [Unknown]:
What is the name of your State or Province?
   [Unknown]:
What is the two-letter country code for this unit?
   [Unknown]:
Is CN=asterisk.mydomain.tld, OU=Unknown, O=Unknown, L=Unknown,
ST=Unknown, C=Unknown correct?
   [no]: yes
```

You will then be asked for another password, in which case you must just press enter for the same password as Tigase will not work without them being the same.

```
Enter key password for <mykey>
              (RETURN if same as keystore password):
```

Configuring init.properties

The next step is to configure the init.properties file which is used by Tigase to generate the tigase.xml file. Whenever you change the init.properties file because sure to remove the current

tigase.xml file so that it will be regenerated at start up.

```
# cd /opt/Tigase-4.3.1-b1858/etc
```

Then edit the init.properties file and add the following:

```
config-type=--gen-config-def
--admins=admin@asterisk.mydomain.tld
--virt-hosts=asterisk.mydomain.tld
--debug=server
--user-db=derby
--user-db-uri=jdbc:derby:/opt/Tigase-4.3.1-b1858
--comp-name-1=pubsub
--comp-class-1=tigase.pubsub.PubSubComponent
```

Be sure to change the domain in the --admin and --virt-hosts options. The most important lines are --comp-name-1 and --comp-class-1 which tell Tigase to load the PubSub module.

2.2. Running Tigase

You can then start the Tigase server with the tigase.sh script.

```
# cd /opt/Tigase-4.3.1-b1858
# ./scripts/tigase.sh start etc/tigase.conf
```

2.3. Adding Buddies to Tigase

At this time, Asterisk is not able to automatically register your peers for you, so you'll need to use an external application to do the initial registration.

Pidgin is an excellent multi-protocol instant messenger application which supports XMPP. It runs on Linux, Windows, and OSX, and is open source. You can get Pidgin from http://www.pidgin.im

Then add the two buddies we'll use in Asterisk with Pidgin by connecting to the Tigase server. For more information about how to register new buddies, see the Pidgin documentation.

Once the initial registration is done and loaded into Tigase, you no longer need to worry about using Pidgin. Asterisk will then be able to load the peers into memory at start up.

The example peers we've used in the following documentation for our two nodes are:

```
server1@asterisk.mydomain.tld/astvoip1
server2@asterisk.mydomain.tld/astvoip2
```

3. Installing Asterisk

Install Asterisk as usual. However, you'll need to make sure you have the res_jabber module compiled, which requires the iksemel development library. Additionally, be sure you have the OpenSSL development library installed so you can connect securly to the Tigase server.

Make sure you check menuselect that res_jabber is selected so that it will compile.

```
# cd asterisk-source
# ./configure

# make menuselect
   ---> Resource Modules
```

If you don't have jabber.conf in your existing configuration, because sure to copy the sample configuration file there.

```
# cd configs
# cp jabber.conf.sample /etc/asterisk/jabber.conf
```

3.1. Configuring Asterisk

We then need to configure our servers to communicate with the Tigase server. We need to modify the jabber.conf file on the servers. The configurations below are for a 2 server setup, but could be expanded for additional servers easily.

The key note here is to note that the pubsub_node option needs to start with pubsub, so for example, pubsub.asterisk.mydomain.tld. Without the 'pubsub' your Asterisk system will not be able to distribute events.

Additionally, you will need to specify each of the servers you need to connec to using the 'buddy' option.

*Asterisk Server 1

jabber.conf on server1

```
[general]
debug=no                              ;;Turn on debugging by
default.
;autoprune=yes                        ;;Auto remove users from
buddy list. Depending on your
                                      ;;setup (ie, using your
personal Gtalk account for a test)
                                      ;;you might lose your
contacts list. Default is 'no'.
autoregister=yes                      ;;Auto register users from
buddy list.
;collection_nodes=yes                 ;;Enable support for
XEP-0248 for use with
                                      ;;distributed device state.
Default is 'no'.
;pubsub_autocreate=yes                ;;Whether or not the PubSub
server supports/is using
                                      ;;auto-create for nodes.  If
it is, we have to
                                      ;;explicitly pre-create
nodes before publishing them.
                                      ;;Default is 'no'.


[asterisk]
type=client
serverhost=asterisk.mydomain.tld
pubsub_node=pubsub.asterisk.mydomain.tld
username=server1@asterisk.mydomain.tld/astvoip1
secret=welcome
distribute_events=yes
status=available
usetls=no
usesasl=yes
buddy=server2@asterisk.mydomain.tld/astvoip2
```

Asterisk Server 2

<table>
<tr><th colspan="1">jabber.conf on server2</th></tr>
</table>

```
[general]
debug=yes        ;;Turn on debugging by default.
;autoprune=yes       ;;Auto remove users from buddy list. Depending on
your
        ;;setup (ie, using your personal Gtalk account for a test)
        ;;you might lose your contacts list. Default is 'no'.
autoregister=yes     ;;Auto register users from buddy list.
;collection_nodes=yes     ;;Enable support for XEP-0248 for use with
        ;;distributed device state.  Default is 'no'.
;pubsub_autocreate=yes     ;;Whether or not the PubSub server
supports/is using
        ;;auto-create for nodes.  If it is, we have to
        ;;explicitly pre-create nodes before publishing them.
        ;;Default is 'no'.

[asterisk]
type=client
serverhost=asterisk.mydomain.tld
pubsub_node=pubsub.asterisk.mydomain.tld
username=server2@asterisk.mydomain.tld/astvoip2
secret=welcome
distribute_events=yes
status=available
usetls=no
usesasl=yes
buddy=server1@asterisk.mydomain.tld/astvoip1
```

4. Basic Testing of Asterisk with XMPP PubSub

Once you have Asterisk installed with XMPP PubSub, it is time to test it out.

We need to start up our first server and make sure we get connected to the XMPP server. We can verify this with an Asterisk console command to determine if we're connected.

On Asterisk 1 we can run 'jabber show connected' to verify we're connected to the XMPP server.

```
*CLI> jabber show connected
Jabber Users and their status:
        User: server1@asterisk.mydomain.tld/astvoip1     - Connected
----
    Number of users: 1
```

The command above has given us output which verifies we've connected our first server.

We can then check the state of our buddies with the 'jabber show buddies' CLI command.

```
*CLI> jabber show buddies
Jabber buddy lists
Client: server1@asterisk.mydomain.tld/astvoip1
 Buddy: server2@asterisk.mydomain.tld
  Resource: None
 Buddy: server2@asterisk.mydomain.tld/astvoip2
  Resource: None
```

The output above tells us we're not connected to any buddies, and thus we're not distributing state to anyone (or getting it from anyone). That makes sense since we haven't yet started our other server.

Now, let's start the other server and verify the servers are able to establish a connection between each other.

On Asterisk 2, again we run the 'jabber show connected' command to make sure we've connected successfully to the XMPP server.

```
*CLI> jabber show connected
Jabber Users and their status:
       User: server2@asterisk.mydomain.tld/astvoip2     - Connected
----
   Number of users: 1
```

And now we can check the status of our buddies.

```
*CLI> jabber show buddies
Jabber buddy lists
Client: server2@scooter/astvoip2
 Buddy: server1@asterisk.mydomain.tld
  Resource: astvoip1
   node: http://www.asterisk.org/xmpp/client/caps
   version: asterisk-xmpp
   Jingle capable: yes
  Status: 1
  Priority: 0
 Buddy: server1@asterisk.mydomain.tld/astvoip1
  Resource: None
```

Excellent! So we're connected to the buddy on Asterisk 1, and we could run the same command on Asterisk 1 to verify the buddy on Asterisk 2 is seen.

5. Testing Distributed Device State

The easiest way to test distributed device state is to use the DEVICE_STATE() diaplan function. For example, you could have the following piece of dialplan on every server:

```
[devstate_test]

exten => 1234,hint,Custom:mystate

exten => set_inuse,1,Set(DEVICE_STATE(Custom:mystate)=INUSE)
exten => set_not_inuse,1,Set(DEVICE_STATE(Custom:mystate)=NOT_INUSE)

exten => check,1,NoOp(Custom:mystate is
${DEVICE_STATE(Custom:mystate)})
```

Now, you can test that the cluster-wide state of "Custom:mystate" is what you would expect after going to the CLI of each server and adjusting the state.

```
server1*CLI> console dial set_inuse@devstate_test
    ...

server2*CLI> console dial check@devstate_test
    -- Executing [check@devstate_test:1] NoOp("OSS/dsp",
"Custom:mystate is INUSE") in new stack
```

Various combinations of setting and checking the state on different servers can be used to verify that it works as expected. Also, you can see the status of the hint on each server, as well, to see how extension state would reflect the
state change with distributed device state:

```
server2*CLI> core show hints
    -= Registered Asterisk Dial Plan Hints =-
                1234@devstate_test        : Custom:mystate
State:InUse            Watchers  0
```

One other helpful thing here during testing and debugging is to enable debug logging. To do so, enable debug on the console in /etc/asterisk/logger.conf. Also, enable debug at the Asterisk CLI.

```
*CLI> core set debug 1
```

When you have this debug enabled, you will see output during the processing of every device state change. The important thing to look for is where the known state of the device for each server is added together to determine the overall
state.

6. Notes On Large Installations

On larger installations where you want a fully meshed network of buddies (i.e. all servers have all the buddies of the remote servers), you may want some method of distributing those buddies automatically so you don't need to modify
all servers (N+1) every time you add a new server to the cluster.

The problem there is that you're confined by what's allowed in XEP-0060, and unfortunately that means modifying affiliations by individual JID (as opposed to the various subscription access models, which are more flexible).

See here for details http://xmpp.org/extensions/xep-0060.html#owner-affiliations

One method for making this slightly easier is to utilize the #exec functionality in configuration files, and dynamically generate the buddies via script that pulls the information from a database, or to #include a file which is automatically generated on all the servers when you add a new node to the cluster.

Unfortunately this still requires a reload of res_jabber.so on all the servers, but this could also be solved through the use of the Asterisk Manager Interface (AMI).

So while this is not the ideal situation, it is programmatically solvable with existing technologies and features found in Asterisk today.

DUNDi - Distributed Universal Number Discovery

Digium General Peering Agreement

```
DIGIUM GENERAL PEERING AGREEMENT (TM)
                  Version 1.0.0, September 2004
 Copyright (C) 2004 Digium, Inc.
             445 Jan Davis Drive, Huntsville, AL 35806 USA


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


        --------------------------------------------------------


                 DIGIUM GENERAL PEERING AGREEMENT


                             PREAMBLE


   For most of the history of telecommunications, the power of being
   able
to locate and communicate with another person in a system, be it
```

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

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

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

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

   You do not have to execute this GPA to use a program or technology
employing the DUNDi protocol, however if you do not execute this

GPA,
you will not be able to peer using DUNDi and the E164 context with
anyone who is a member of the trust group by virtue of their having
executed this GPA with another member.

The parties to this GPA agree as follows:

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

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

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

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

     (d) The term "Egress Gateway" refers an Internet facility that
         provides a communications path to a Service or Services that
may
         not be directly addressable via the Internet.

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

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

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

(h) The term "Subscriber" means the owner of, or someone who
     contracts to receive, the services identified by an E.164
     number.

(i) The term "Authorizing Individual" means the Subscriber to a
     number who has authorized a Participant to provide Routes
     regarding their services via this Peering System.

(j) The term "Route Authority" refers to a Participant that
provides
     an original source of said Route within the Peering System.
     Routes are propagated from the Route Authorities through the
     Peering System and may be cached at intermediate points.
There
     may be multiple Route Authorities for any Service.

(k) The term "Participant" (introduced above) refers to any
member
     of the Peering System.

(l) The term "Service Provider" refers to the carrier (e.g.,
     exchange carrier, Internet Telephony Service Provider, or
other
     reseller) that provides communication facilities for a
     particular Service to a Subscriber, Customer or other End
User.

(m) The term "Weight" refers to a numeric quality assigned to a
     Route as per the DUNDi protocol specification. The current
     Weight definitions are shown in Exhibit A.

  1. PEERING. The undersigned Participants agree to Propagate Routes
with each other and any other member of the Peering System and
further
agree not to Propagate DUNDi Routes with a third party unless they
have
first have executed this GPA (in its unmodified form) with such
third
party. The Participants further agree only to Propagate Routes with
Participants whom they reasonably believe to be honoring the terms
of
the GPA. Participants may not insert, remove, amend, or otherwise

modify any of the terms of the GPA.

  2. ACCEPTABLE USE POLICY. The DUNDi protocol contains information
that reflect a Subscriber's or Egress Gateway's decisions to receive
calls. In addition to the terms and conditions set forth in this
GPA,
the Participants agree to honor the intent of restrictions encoded
in
the DUNDi protocol. To that end, Participants agree to the
following:

    (a) A Participant may not utilize or permit the utilization of
        Routes for which the Subscriber or Egress Gateway provider
has
        indicated that they do not wish to receive "Unsolicited
Calls"
        for the purpose of making an unsolicited phone call on
behalf of
        any party or organization.

    (b) A Participant may not utilize or permit the utilization of
        Routes which have indicated that they do not wish to receive
        "Unsolicited Commercial Calls" for the purpose of making an
        unsolicited phone call on behalf of a commercial
organization.

    (c) A Participant may never utilize or permit the utilization of
any
        DUNDi route for the purpose of making harassing phone calls.

    (d) A Party may not utilize or permit the utilization of DUNDi
        provided Routes for any systematic or random calling of
numbers
        (e.g., for the purpose of locating facsimile, modem
services, or
        systematic telemarketing).

    (e) Initial control signaling for all communication sessions
that
        utilize Routes obtained from the Peering System must be sent
        from a member of the Peering System to the Service or Egress
        Gateway identified in the selected Route. For example, 'SIP
        INVITES' and IAX2 "NEW" commands must be sent from the
        requesting DUNDi node to the terminating Service.

    (f) A Participant may not disclose any specific Route, Service

or
        Participant contact information obtained from the Peering
System
        to any party outside of the Peering System except as a
        by-product of facilitating communication in accordance with
        section 2e (e.g., phone books or other databases may not be
        published, but the Internet addresses of the Egress Gateway
or
        Service does not need to be obfuscated.)

    (g) The DUNDi Protocol requires that each Participant include
valid
        contact information about itself (including information
about
        nodes connected to each Participant). Participants may use
or
        disclose the contact information only to ensure enforcement
of
        legal furtherance of this Agreement.

  3. ROUTES. The Participants shall only propagate valid Routes, as
defined herein, through the Peering System, regardless of the
original
source. The Participants may only provide Routes as set forth
below,
and then only if such Participant has no good faith reason to
believe
such Route to be invalid or unauthorized.

    (a) A Participant may provide Routes if each Route has as its
        original source another member of the Peering System who has
        duly executed the GPA and such Routes are provided in
accordance
        with this Agreement; provided that the Routes are not
modified
        (e.g., with regards to existence, destination, technology or
        Weight); or

    (b) A Participant may provide Routes for Services with any
Weight
        for which it is the Subscriber; or

    (c) A Participant may provide Routes for those Services whose
        Subscriber has authorized the Participant to do so, provided
        that the Participant is able to confirm that the Authorizing
        Individual is the Subscriber through:

i. a written statement of ownership from the Authorizing
                Individual, which the Participant believes in good
faith
                to be accurate (e.g., a phone bill with the name of
the
                Authorizing Individual and the number in question);
or

            ii. the Participant's own direct personal knowledge that
the
                Authorizing Individual is the Subscriber.

    (d) A Participant may provide Routes for Services, with Weight
in
        accordance with the Current DUNDi Specification, if it can
in
        good faith provide an Egress Gateway to that Service on the
        traditional telephone network without cost to the calling
party.

  4. REVOCATION. A Participant must provide a free, easily
accessible
mechanism by which a Subscriber may revoke permission to act as a
Route
Authority for his Service. A Participant must stop acting as a
Route
Authority for that Service within 7 days after:

    (a) receipt of a revocation request;

    (b) receiving other notice that the Service is no longer valid;
or

    (c) determination that the Subscriber's information is no longer
        accurate (including that the Subscriber is no longer the
service
        owner or the service owner's authorized delegate).

  5. SERVICE FEES. A Participant may charge a fee to act as a Route
Authority for a Service, with any Weight, provided that no
Participant
may charge a fee to propagate the Route received through the Peering
System.

  6. TOLL SERVICES. No Participant may provide Routes for any

Services
that require payment from the calling party or their customer for
communication with the Service. Nothing in this section shall
prohibit
a Participant from providing routes for Services where the calling
party
may later enter into a financial transaction with the called party
(e.g., a Participant may provide Routes for calling cards services).

   7. QUALITY. A Participant may not intentionally impair
communication
using a Route provided to the Peering System (e.g. by adding delay,
advertisements, reduced quality). If for any reason a Participant
is
unable to deliver a call via a Route provided to the Peering System,
that Participant shall return out-of-band Network Congestion
notification (e.g. "503 Service Unavailable" with SIP protocol or
"CONGESTION" with IAX protocol).

   8. PROTOCOL COMPLIANCE. Participants agree to Propagate Routes in
strict compliance with current DUNDi protocol specifications.

   9. ADMINISTRATIVE FEES. A Participant may charge (but is not
required
to charge) another Participant a reasonable fee to cover
administrative
expenses incurred in the execution of this Agreement. A Participant
may
not charge any fee to continue the relationship or to provide Routes
to
another Participant in the Peering System.

   10. CALLER IDENTIFICATION. A Participant will make a good faith
effort
to ensure the accuracy and appropriate nature of any caller
identification that it transmits via any Route obtained from the
Peering
System. Caller identification shall at least be provided as a valid
E.164 number.

   11. COMPLIANCE WITH LAWS. The Participants are solely responsible
for
determining to what extent, if any, the obligations set forth in
this
GPA conflict with any laws or regulations their region. A
Participant

may not provide any service or otherwise use DUNDi under this GPA if
doing so is prohibited by law or regulation, or if any law or
regulation
imposes requirements on the Participant that are inconsistent with
the
terms of this GPA or the Acceptable Use Policy.

   12. WARRANTY. EACH PARTICIPANT WARRANTS TO THE OTHER PARTICIPANTS
THAT
IT MADE, AND WILL CONTINUE TO MAKE, A GOOD FAITH EFFORT TO
AUTHENTICATE
OTHERS IN THE PEERING SYSTEM AND TO PROVIDE ACCURATE INFORMATION IN
ACCORDANCE WITH THE TERMS OF THIS GPA. THIS WARRANTY IS MADE
BETWEEN
THE PARTICIPANTS, AND THE PARTICIPANTS MAY NOT EXTEND THIS WARRANTY
TO
ANY NON-PARTICIPANT INCLUDING END-USERS.

   13. DISCLAIMER OF WARRANTIES. THE PARTICIPANTS UNDERSTAND AND
AGREE
THAT ANY SERVICE PROVIDED AS A RESULT OF THIS GPA IS "AS IS." EXCEPT
FOR
THOSE WARRANTIES OTHERWISE EXPRESSLY SET FORTH HEREIN, THE
PARTICIPANTS
DISCLAIM ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND OR NATURE,
EXPRESS OR IMPLIED, AS TO THE CONDITION, VALUE OR QUALITIES OF THE
SERVICES PROVIDED HEREUNDER, AND SPECIFICALLY DISCLAIM ANY
REPRESENTATION OR WARRANTY OF MERCHANTABILITY, SUITABILITY OR
FITNESS
FOR A PARTICULAR PURPOSE OR AS TO THE CONDITION OR WORKMANSHIP
THEREOF,
OR THE ABSENCE OF ANY DEFECTS THEREIN, WHETHER LATENT OR PATENT,
INCLUDING ANY WARRANTIES ARISING FROM A COURSE OF DEALING, USAGE OR
TRADE PRACTICE. EXCEPT AS EXPRESSLY PROVIDED HEREIN, THE
PARTICIPANTS
EXPRESSLY DISCLAIM ANY REPRESENTATIONS OR WARRANTIES THAT THE
PEERING
SERVICE WILL BE CONTINUOUS, UNINTERRUPTED OR ERROR-FREE, THAT ANY
DATA
SHARED OR OTHERWISE MADE AVAILABLE WILL BE ACCURATE OR COMPLETE OR
OTHERWISE COMPLETELY SECURE FROM UNAUTHORIZED ACCESS.

   14. LIMITATION OF LIABILITIES. NO PARTICIPANT SHALL BE LIABLE TO
ANY
OTHER PARTICIPANT FOR INCIDENTAL, INDIRECT, CONSEQUENTIAL, SPECIAL,
PUNITIVE OR EXEMPLARY DAMAGES OF ANY KIND (INCLUDING LOST REVENUES

OR
PROFITS, LOSS OF BUSINESS OR LOSS OF DATA) IN ANY WAY RELATED TO
THIS
GPA, WHETHER IN CONTRACT OR IN TORT, REGARDLESS OF WHETHER SUCH
PARTICIPANT WAS ADVISED OF THE POSSIBILITY THEREOF.

   15. END-USER AGREEMENTS. The Participants may independently enter
into agreements with end-users to provide certain services (e.g.,
fees
to a Subscriber to originate Routes for that Service). To the
extent
that provision of these services employs the Peering System, the
Parties
will include in their agreements with their end-users terms and
conditions consistent with the terms of this GPA with respect to the
exclusion of warranties, limitation of liability and Acceptable Use
Policy. In no event may a Participant extend the warranty described
in
Section 12 in this GPA to any end-users.

   16. INDEMNIFICATION. Each Participant agrees to defend, indemnify
and
hold harmless the other Participant or third-party beneficiaries to
this
GPA (including their affiliates, successors, assigns, agents and
representatives and their respective officers, directors and
employees)
from and against any and all actions, suits, proceedings,
investigations, demands, claims, judgments, liabilities,
obligations,
liens, losses, damages, expenses (including, without limitation,
attorneys' fees) and any other fees arising out of or relating to
(i)
personal injury or property damage caused by that Participant, its
employees, agents, servants, or other representatives; (ii) any act
or
omission by the Participant, its employees, agents, servants or
other
representatives, including, but not limited to, unauthorized
representations or warranties made by the Participant; or (iii) any
breach by the Participant of any of the terms or conditions of this
GPA.

   17. THIRD PARTY BENEFICIARIES. This GPA is intended to benefit
those
Participants who have executed the GPA and who are in the Peering

System. It is the intent of the Parties to this GPA to give to those
Participants who are in the Peering System standing to bring any
necessary legal action to enforce the terms of this GPA.

18. TERMINATION. Any Participant may terminate this GPA at any
time,
with or without cause. A Participant that terminates must
immediately
cease to Propagate.

19. CHOICE OF LAW. This GPA and the rights and duties of the
Parties
hereto shall be construed and determined in accordance with the
internal
laws of the State of New York, United States of America, without
regard
to its conflict of laws principles and without application of the
United
Nations Convention on Contracts for the International Sale of Goods.

20. DISPUTE RESOLUTION. Unless otherwise agreed in writing, the
exclusive procedure for handling disputes shall be as set forth
herein.
Notwithstanding such procedures, any Participant may, at any time,
seek
injunctive relief in addition to the process described below.

    (a) Prior to mediation or arbitration the disputing Participants
        shall seek informal resolution of disputes. The process
shall be
        initiated with written notice of one Participant to the
other
        describing the dispute with reasonable particularity
followed
        with a written response within ten (10) days of receipt of
        notice. Each Participant shall promptly designate an
executive
        with requisite authority to resolve the dispute. The
informal
        procedure shall commence within ten (10) days of the date of
        response. All reasonable requests for non-privileged
information
        reasonably related to the dispute shall be honored. If the
        dispute is not resolved within thirty (30) days of
commencement
        of the procedure either Participant may proceed to mediation

or
        arbitration pursuant to the rules set forth in (b) or (c)
below.

    (b) If the dispute has not been resolved pursuant to (a) above
or,
        if the disputing Participants fail to commence informal
dispute
        resolution pursuant to (a) above, either Participant may, in
        writing and within twenty (20) days of the response date
noted
        in (a) above, ask the other Participant to participate in a
one
        (1) day mediation with an impartial mediator, and the other
        Participant shall do so. Each Participant will bear its own
        expenses and an equal share of the fees of the mediator. If
the
        mediation is not successful the Participants may proceed
with
        arbitration pursuant to (c) below.

    (c) If the dispute has not been resolved pursuant to (a) or (b)
        above, the dispute shall be promptly referred, no later than
one
        (1) year from the date of original notice and subject to
        applicable statute of limitations, to binding arbitration in
        accordance with the UNCITRAL Arbitration Rules in effect on
the
        date of this contract. The appointing authority shall be
the
        International Centre for Dispute Resolution. The case shall
be
        administered by the International Centre for Dispute
Resolution
        under its Procedures for Cases under the UNCITRAL
Arbitration
        Rules. Each Participant shall bear its own expenses and
shall
        share equally in fees of the arbitrator. All arbitrators
shall
        have substantial experience in information technology and/or
in
        the telecommunications business and shall be selected by the
        disputing participants in accordance with UNCITRAL
Arbitration
        Rules. If any arbitrator, once selected is unable or

unwilling
to continue for any reason, replacement shall be filled via
the
process described above and a re-hearing shall be conducted.
The
disputing Participants will provide each other with all
requested documents and records reasonably related to the
dispute in a manner that will minimize the expense and
inconvenience of both parties. Discovery will not include
depositions or interrogatories except as the arbitrators
expressly allow upon a showing of need. If disputes arise
concerning discovery requests, the arbitrators shall have
sole
and complete discretion to resolve the disputes. The parties
and
arbitrator shall be guided in resolving discovery disputes
by
the Federal Rules of Civil Procedure. The Participants agree
that time of the essence principles shall guide the hearing
and
that the arbitrator shall have the right and authority to
issue
monetary sanctions in the event of unreasonable delay. The
arbitrator shall deliver a written opinion setting forth
findings of fact and the rationale for the award within
thirty
(30) days following conclusion of the hearing. The award of
the
arbitrator, which may include legal and equitable relief,
but
which may not include punitive damages, will be final and
binding upon the disputing Participants, and judgment may be
entered upon it in accordance with applicable law in any
court
having jurisdiction thereof. In addition to award the
arbitrator shall have the discretion to award the prevailing
Participant all or part of its attorneys' fees and costs,
including fees associated with arbitrator, if the arbitrator
determines that the positions taken by the other Participant
on
material issues of the dispute were without substantial
foundation. Any conflict between the UNCITRAL Arbitration
Rules
and the provisions of this GPA shall be controlled by this
GPA.

21. INTEGRATED AGREEMENT. This GPA, constitutes the complete integrated agreement between the parties concerning the subject matter hereof. All prior and contemporaneous agreements, understandings, negotiations or representations, whether oral or in writing, relating to the subject matter of this GPA are superseded and canceled in their entirety.

22. WAIVER. No waiver of any of the provisions of this GPA shall be deemed or shall constitute a waiver of any other provision of this GPA, whether or not similar, nor shall such waiver constitute a continuing waiver unless otherwise expressly so provided in writing. The failure of either party to enforce at any time any of the provisions of this GPA, or the failure to require at any time performance by either party of any of the provisions of this GPA, shall in no way be construed to be a present or future waiver of such provisions, nor in any way affect the ability of a Participant to enforce each and every such provision thereafter.

23. INDEPENDENT CONTRACTORS. Nothing in this GPA shall make the Parties partners, joint venturers, or otherwise associated in or with the business of the other. Parties are, and shall always remain, independent contractors. No Participant shall be liable for any debts, accounts, obligations, or other liabilities of the other Participant, its agents or employees. No party is authorized to incur debts or other obligations of any kind on the part of or as agent for the other. This GPA is not a franchise agreement and does not create a franchise relationship between the parties, and if any provision of this GPA is deemed to create a franchise between the parties, then this GPA shall automatically terminate.

24. CAPTIONS AND HEADINGS. The captions and headings used in this GPA
are used for convenience only and are not to be given any legal
effect.

25. EXECUTION. This GPA may be executed in counterparts, each of
which
so executed will be deemed to be an original and such counterparts
together will constitute one and the same Agreement. The Parties
shall
transmit to each other a signed copy of the GPA by any means that
faithfully reproduces the GPA along with the Signature. For
purposes of
this GPA, the term "signature" shall include digital signatures as
defined by the jurisdiction of the Participant signing the GPA.

                             Exhibit A

Weight Range   Requirements

0-99   May only be used under authorization of
Owner

100-199   May only be used by the Owner's service
                          provider, regardless of authorization.

200-299   Reserved -- do not use for e164 context.

300-399   May only be used by the owner of the code
under
                          which the Owner's number is a part of.

400-499   May be used by any entity providing access
via
                          direct connectivity to the Public Switched
                          Telephone Network.

500-599   May be used by any entity providing access
via
                          indirect connectivity to the Public Switched
                          Telephone Network (e.g. Via another VoIP
                          provider)

600-   Reserved-- do not use for e164 context.

                Participant   Participant

Company:

Address:

Email:

        _____
_____
        Authorized Signature   Authorized Signature

Name:

END OF GENERAL PEERING AGREEMENT

How to Peer using this GPA If you wish to exchange routing
information
with parties using the e164 DUNDi context, all you must do is
execute
this GPA with any member of the Peering System and you will become a
member of the Peering System and be able to make Routes available in

```
accordance with this GPA.

DUNDi, IAX, Asterisk and GPA are trademarks of Digium, Inc.
```

External IVR Interface

Asterisk External IVR Interface

If you load app_externalivr.so in your Asterisk instance, you will have an ExternalIVR()
application available in your dialplan. This application implements a simple protocol for
bidirectional communication with an external process, while simultaneous playing audio files to
the connected channel (without interruption or blocking).

There are two ways to use ExternalIVR(); you can execute an application on the local system or
you can establish a socket
connection to a TCP/IP socket server.

To execute a local application use the form:

```
ExternalIVR(/full/path/to/applcation[(arguments)],options)
```

The arguments are optional, however if they exist they must be enclosed in parentheses. The
external application will be executed in a child process, with its standard file handles connected
to the Asterisk process as follows:

- stdin (0) - Events will be received on this handle
- stdout (1) - Commands can be sent on this handle
- stderr (2) - Messages can be sent on this handle

> ℹ️ Use of stderr for message communication is discouraged because it is not supported by a socket connection.

To create a socket connection use the form:

```
ExternalIVR(ivr://host[:port][(arguments)],options)
```

The host can be a fqdn or an ip address. The port is optional, if not specified the default of 2949
will be used. The arguments are optional however if they exist they must be enclosed in
parentheses. TheExternalIVR application will connect to the specified socket server and
establish a bi-directional socket connection, where events will be sent to the TCP/IP server and
commands received from it.

The specific ExternalIVR options, events and commands are detailed below.

Upon execution, if not specifically prevented by an option, the ExternalIVR application will
answer the channel (if it's not already answered), create an audio generator, and start playing
silence. When your application wants to send audio to the channel, it can send a command (see

below) to add a file to the generator's playlist. The generator will then work its way through the list, playing each file in turn until it either runs out of files to play, the channel is hung up, or a command is received to clear the list and start with a new file. At any time, more files can be added to the list and the generator will play them in sequence.

While the generator is playing audio (or silence), any DTMF events received on the channel will be sent to the child process (see below). Note that this can happen at any time, since the generator, the child process and the channel thread are all executing independently. It is very important that your external application be ready to receive events from Asterisk at all times (without blocking), or you could cause the channel to become non-responsive.

If the child process dies, or the remote server disconnects, ExternalIVR() will notice this and hang up the channel
immediately (and also send a message to the log).

ExternalIVR() Options

- n - 'n'oanswer, don't answer an otherwise unanswered channel.
- i - 'i'gnore_hangup, instead of sending an 'H' event and exiting ExternalIVR() upon channel hangup, it instead sends an 'I' event and expects the external application to exit the process.
- d - 'd'ead, allows the operation of ExternalIVR() on channels that have already been hung up.

Events

All events are be newline-terminated strings and are sent in the following format:

```
tag,timestamp[,data]
```

The tag can be one of the following characters:

- 0-9 - DTMF event for keys 0 through 9
- A-D - DTMF event for keys A through D
- * - DTMF event for key *
- # - DTMF event for key #
- H - The channel was hung up by the connected party
- E - The script requested an exit
- Z - The previous command was unable to be executed. There may be a data element if appropriate, see specific commands below for details
- T - The play list was interrupted (see S command below)
- D - A file was dropped from the play list due to interruption (the data element will be the dropped file name) NOTE: this tag conflicts with the D DTMF event tag. The existence of the data element is used to differentiate between the two cases
- F - A file has finished playing (the data element will be the file name)
- P - A response to the 'P' command (see below)
- G - A response to the 'G' command (see below)
- I - A Inform message, meant to "inform" the client that something has occurred. (see Inform Messages below)

The timestamp will be 10 digits long, and will be a decimal representation of a standard Unix epoch-based timestamp.

Commands

All commands are newline-terminated strings.

The child process can send one of the following commands:

- S,filename

- A,filename
- H,message
- E,message
- O,option
- V,name=value[,name=value[,name=value]]
- G,name[,name[,name]]
- L,log_message
- P,TIMESTAMP
- T,TIMESTAMP

The 'S' command checks to see if there is a playable audio file with the specified name, and if so, clear's the generator's playlist and places the file onto the list. Note that the playability check does not take into account transcoding requirements, so it is possible for the file to not be played even though it was found. If the file does not exist it sends a Z response with the data element set to the file requested. If the generator is not currently playing silence, then T and D events will be sent to signal the playlist interruption and notify it of the files that will not be played.

The 'A' command checks to see if there is a playable audio file with the specified name, and if so, appends it to the generator's playlist. The same playability and exception rules apply as for the 'S' command.

The 'E' command logs the supplied message to the Asterisk log, stops the generator and terminates the ExternalIVR application, but continues execution in the dialplan.

The 'H' command logs the supplied message to the Asterisk log, stops the generator, hangs up the channel and terminates the ExternalIVR application.

The 'O' command allows the child to set/clear options in the ExternalIVR() application. The supported options are:
(no)autoclear: Automatically interrupt and clear the playlist upon reception of DTMF input.

The 'T' command will answer an unanswered channel. If it fails either answering the channel or starting the generator it sends a Z response of "Z,TIMESTAMP,ANSWER_FAILED" or "Z,TIMESTAMP,GENERATOR_FAILED" respectively.

The 'V' command sets the specified channel variable(s) to the specified value(s).

The 'G' command gets the specified channel variable(s). Multiple variables are separated by commas. Response is in name=value format.

The 'P' command gets the parameters passed into ExternalIVR() minus the options to ExternalIVR() itself:

If ExternalIVR() is executed as:

```
ExternalIVR(/usr/bin/foo(arg1,arg2),n)
```

The response to the 'P' command would be:

```
P,TIMESTAMP,/usr/bin/foo,arg1,arg2
```

> ⚠️ This is the only way for a TCP/IP server to be able to get retrieve the arguments.

The 'L' command puts a message into the Asterisk log.

> ⚠️ This is preferred to using stderr and is the only way for a TCP/IP server to log a message.

Inform Messages

The only inform message that currently exists is a HANGUP message, in the form I,TIMESTAMP,HANGUP and is used to inform of a hangup when the i option is specified.

Errors

Any newline-terminated output generated by the child process on its stderr handle will be copied into the Asterisk log.

Followme - Realtime

Followme is now realtime-enabled.

To use, you must define two backend data structures, with the following fields:

```
followme:
 name                 Name of this followme entry.  Specified when
invoking the FollowMe
                      application in the dialplan.  This field is the
only one which is
                      mandatory.  All of the other fields will
inherit the default from
                      followme.conf, if not specified in this data
resource.
 musicclass OR        The musiconhold class used for the caller while
waiting to be
 musiconhold OR       connected.
    music
 context              Dialplan context from which to dial numbers
 takecall             DTMF used to accept the call and be connected.
For obvious reasons,
                      this needs to be a single digit, '*', or '#'.
 declinecall          DTMF used to refuse the call, sending it onto
the next step, if any.
 call_from_prompt     Prompt to play to the callee, announcing the
call.
 norecording_prompt   The alternate prompt to play to the callee,
when the caller
                      refuses to leave a name (or the option isn't
set to allow them).
 options_prompt       Normally, "press 1 to accept, 2 to decline".
 hold_prompt          Message played to the caller while dialing the
followme steps.
 status_prompt        Normally, "Party is not at their desk".
 sorry_prompt         Normally, "Unable to locate party".
```

```
followme_numbers:
 name                 Name of this followme entry.  Must match the
name above.
 ordinal              An integer, specifying the order in which these
numbers will be
                      followed.
 phonenumber          The telephone number(s) you would like to call,
separated by '&'.
 timeout              Timeout associated with this step.  See the
followme documentation
                      for more information on how this value is
handled.
```

IAX2 Security

IAX2 Security

Copyright (c) 2009 - Digium, Inc.
All Rights Reserved.
Document Version 1.0
09/03/09
Asterisk Development Team <asteriskteam@digium.com>

Table of Contents

1. Introduction

1.1. Overview

A change has been made to the IAX2 protocol to help mitigate denial of service attacks. This change is referred to as call token validation. This change affects how messages are exchanged and is not backwards compatible for an older client connecting to an updated server, so a number of options have been provided to disable call token validation as needed for compatibility purposes.

In addition to call token validation, Asterisk can now also limit the number of connections allowed

per IP address to disallow one host from preventing other hosts from making successful connections. These options are referred to as call number limits.

For additional details about the configuration options referenced in this document, see the sample configuration file, iax.conf.sample. For information regarding the details of the call token validation protocol modification, see section 3 (Protocol Modification) of this document.

2. User Guide

2.1. Configuration

2.1.1. Quick Start

We strongly recommend that administrators leave the IAX2 security enhancements in place where possible. However, to bypass the security enhancements completely and have Asterisk work exactly as it did before, the following options can be specified in the [general] section of iax.conf:

```
[general]

...

calltokenoptional = 0.0.0.0/0.0.0.0

maxcallnumbers = 16382

...
```

2.1.2. Controlled Networks

This section discusses what needs to be done for an Asterisk server on a network where no unsolicited traffic will reach the IAX2 service.

2.1.2.1. Full Upgrade

If all IAX2 endpoints have been upgraded, then no changes to configuration need to be made.

2.1.2.2. Partial Upgrade

If only some of the IAX2 endpoints have been upgraded, then some configuration changes will need to be made for interoperability. Since this is for a controlled network, the easiest thing to do is to disable call token validation completely, as described in section 2.1.1.

2.1.3. Public Networks

This section discusses the use of the IAX2 security functionality on public networks where it is possible to receive unsolicited IAX2 traffic.

2.1.3.1. Full Upgrade

If all IAX2 endpoints have been upgraded to support call token validation, then no changes need to be made. However, for enhanced security, the administrator may adjust call number limits to further reduce the potential impact of malicious call number consumption. The following configuration will allow known peers to consume more call numbers than unknown source IP addresses:

```
[general]

; By default, restrict call number usage to a low number.

maxcallnumbers = 16

...

 [callnumberlimits]

; For peers with known IP addresses, call number limits can

; be set in this section. This limit is per IP address for

; addresses that fall in the specified range.

; <IP>/<mask> = <limit>

192.168.1.0/255.255.255.0 = 1024

...

 [peerA]

; Since we won't know the IP address of a dynamic peer until

; they register, a max call number limit can be set in a

; dynamic peer configuration section.

Type = peer

host = dynamic

maxcallnumbers = 1024

...
```

2.1.3.2. Partial Upgrade

If only some IAX2 endpoints have been upgraded, or the status of an IAX2 endpoint is unknown, then call token validation must be disabled to ensure interoperability. To reduce the potential

impact of disabling call token validation, it should only be disabled for a specific peer or user as needed. By using the auto option, call token validation will be changed to required as soon as we determine that the peer supports it.

```
[friendA]

requirecalltoken = auto

...
```

Note that there are some cases where auto should not be used. For example, if multiple peers use the same authentication details, and they have not all upgraded to support call token validation, then the ones that do not support it will get locked out. Once an upgraded client successfully completes an authenticated call setup using call token validation, Asterisk will require it from then on. In that case, it would be better to set the requirecalltoken option to no.

2.1.3.3. Guest Access

Guest access via IAX2 requires special attention. Given the number of existing IAX2 endpoints that do not support call token validation, most systems that allow guest access should do so without requiring call token validation.

```
[guest]

; Note that the name "guest" is special here. When the code

; tries to determine if call token validation is required, it

; will look for a user by the username specified in the

; request. Guest calls can be sent without a username. In

; that case, we will look for a defined user called "guest" to

; determine if call token validation is required or not.

type = user

requirecalltoken = no

...
```

Since disabling call token validation for the guest account allows a huge hole for malicious call number consumption, an option has been provided to segregate the call numbers consumed by

connections not using call token validation from those that do. That way, there are resources dedicated to the more secure connections to ensure that service is not interrupted for them.

```
[general]

maxcallnumbers_nonvalidated = 2048
```

2.2. CLI Commands

2.2.1. iax2 show callnumber usage

Usage: iax2 show callnumber usage [IP address]

Show current IP addresses which are consuming IAX2 call numbers.

2.2.2. iax2 show peer

This command will now also show the configured call number limit and whether or not call token validation is required for this peer.

3. Protocol Modification

This section discusses the modification that has been made to the IAX2 protocol. This information would be most useful to implementors of IAX2.

3.1. Overview

The IAX2 protocol uses a call number to associate messages with which call they belong to. The available amount of call numbers is finite as defined by the protocol. Because of this, it is important to prevent attackers from maliciously consuming call numbers. To achieve this, an enhancement to the IAX2 protocol has been made which is referred to as call token validation.

Call token validation ensures that an IAX2 connection is not coming from a spoofed IP address. In addition to using call token validation, Asterisk will also limit how many call numbers may be consumed by a given remote IP address. These limits have defaults that will usually not need to be changed, but can be modified for a specific need.

The combination of call token validation and call number limits is used to mitigate a denial of service attack to consume all available IAX2 call numbers. An alternative approach to securing IAX2 would be to use a security layer on top of IAX2, such as DTLS [RFC4347] or IPsec [RFC4301].

3.2. Call Token Validation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

For this section, when the word "request" is used, it is referring to the command that starts an IAX2 dialog.

This modification adds a new IAX2 frame type, and a new information element be defined.

Frame Type: CALLTOKEN — 0x28 (40)

IE: CALLTOKEN — 0x36 (54)

When a request is initially sent, it SHOULD include the CALLTOKEN IE with a zero-length payload to indicate that this client supports the CALLTOKEN exchange. When a server receives this request, it MUST respond with the IAX2 message CALLTOKEN. The CALLTOKEN message MUST be sent with a source call number of 0, as a call number will not yet be allocated for this call.

For the sake of backwards compatibility with clients that do not support token validation, server implementations MAY process requests that do not indicate CALLTOKEN support in their initial request. However, this SHOULD NOT be the default behavior, as it gives up the security benefits gained by CALLTOKEN validation.

After a client sends a request with an empty CALLTOKEN IE, it MUST be prepared to receive a CALLTOKEN response, or to receive a response that would be given in the case of a valid CALLTOKEN. This is how a client must behave to inter operate with IAX2 server implementations that do not yet support CALLTOKEN validation.

When an IAX2 client receives a CALLTOKEN response, it MUST send its initial request again. This request MUST include the CALLTOKEN IE with a copy of the value of the CALLTOKEN IE received in the CALLTOKEN response. The IE value is an opaque value. Clients MUST be able to accept a CALLTOKEN payload of any length, up to the maximum length allowed in an IAX2 IE.

The value of the payload in the CALLTOKEN IE is an implementation detail. It is left to the implementor to decide how sophisticated it should be. However, it MUST be enough such that when the CALLTOKEN IE is sent back, it can be used to verify that the source IP address and port number has not been spoofed.

If a server receives a request with an invalid CALLTOKEN IE value, then it MUST drop it and not respond.

3.3. Example Message Exchanges

3.3.1. Call Setup

```
Client Server

\-\--\--\--\- --\--\--\-\-

\----\--\--\--\--\- NEW --\--\------->

(with empty CALLTOKEN IE)

<-\--\--\--\--\- CALLTOKEN --\--\-------\-

(client must ignore

source call number

from this message)

\----\--\--\--\--\- NEW --\--\------->

(with received token)

<-\--\--\--\--\- AUTHREQ --\--\-----\-

... continue as normal ...
```

3.3.2. Call Setup, client does not support CALLTOKEN

```
Client Server

\-\--\--\--\- --\--\--\-\-

\----\--\--\--\--\- NEW --\--\------->

(with no CALLTOKEN IE)

<--\--\--\--\--\- REJECT --\--\-----\-

(sent without allocating

a call number)

\----\--\--\--\--\- ACK --\--\------->
```

3.3.3. Call Setup, client supports CALLTOKEN, server does not

```
Client Server

\-\--\--\--\- --\--\--\-\-

\----\--\--\--\--\- NEW --\--\------->

(with empty CALLTOKEN IE)

<--\--\--\--\--\- AUTHREQ --\--\----\-

(sent without allocating

a call number)

... continue as normal ...
```

3.3.4. Call Setup from client that sends invalid token

```
Client Server

\-\--\--\--\- --\--\--\-\-

\----\--\--\--\--\- NEW --\--\------->

(with invalid CALLTOKEN IE)

... dropped ...
```

4. Asterisk Implementation

This section includes some additional details on the implementation of these changes in Asterisk.

4.1. CALLTOKEN IE Payload

For Asterisk, we will encode the payload of the CALLTOKEN IE such that the server is able to validate a received token without having to store any information after transmitting the CALLTOKEN response. The CALLTOKEN IE payload will contain:

- A timestamp (epoch based)

- SHA1 hash of the remote IP address and port, the timestamp, as well some random data generated when Asterisk starts.

When a CALLTOKEN IE is received, its validity will be determined by recalculating the SHA1 hash. If it is a valid token, the timestamp is checked to determine if the token is expired. The token timeout will be hard coded at 10 seconds for now. However, it may be made configurable at some point if it seems to be a useful addition. If the server determines that a received token is expired, it will treat it as an invalid token and not respond to the request.

By using this method, we require no additional memory to be allocated for a dialog, other than what is on the stack for processing the initial request, until token validation is complete.

However, one thing to note with this CALLTOKEN IE encoding is that a token would be considered valid by Asterisk every time a client sent it until we considered it an expired token. However, with use of the "maxcallnumbers" option, this is not actually a problem. It just means that an attacker could hit their call number limit a bit quicker since they would only have to acquire a single token per timeout period, instead of a token per request.

Jabber in Asterisk

XMPP (Jabber) is an xml based protocol primarily for presence and messaging. It is an open standard and there are several open server implementations, ejabberd, jabberd(2), openfire, and many others, as well as several open source clients, Psi, gajim, gaim etc. XMPP differs from other IM applications as it is immensly extendable. This allows us to easily integrate Asterisk with XMPP. The Asterisk XMPP Interface is provided by res_jabber.so.

res_jabber allows for Asterisk to connect to any XMPP (Jabber) server and is also used to provide the connection interface for chan_jingle and chan_gtalk.

Functions (JABBER_STATUS, JABBER_RECEIVE) and applications (JabberSend) are exposed to the dialplan.

You'll find examples of how to use these functions/applications hereafter. We assume that 'asterisk-xmpp' is properly configured in jabber.conf.

JabberSend

JabberSend sends an XMPP message to a buddy. Example :

```
context default {
 _XXXX => {
      JabberSend(asterisk-xmpp,buddy@gmail.com,${CALLERID(name)} is
calling ${EXTEN});
      Dial(SIP/${EXTEN}, 30);
      Hangup();
 }
}
```

JABBER_STATUS

JABBER_STATUS stores the status of a buddy in a dialplan variable for further use. Here is an AEL example of how to use it:

```
1234 => {
 Set(STATUS=${JABBER_STATUS(asterisk-xmpp,buddy@gmail.com)});
 if (${STATUS}=1) {
  NoOp(User is online and active, ring his Gtalk client.);
  Dial(Gtalk/asterisk-xmpp/buddy@gmail.com);
 } else {
  NoOp(Prefer the SIP phone);
  Dial(SIP/1234);
 }
}
```

JABBER_RECEIVE

JABBER_RECEIVE waits (up to X seconds) for a XMPP message and returns its content. Used along with JabberSend (or SendText,
provided it's implemented in the corresponding channel type), JABBER_RECEIVE helps Asterisk interact with users while calls flow through the dialplan.

JABBER_RECEIVE/JabberSend are not tied to the XMPP media modules chan_gtalk and chan_jingle, and can be used anywhere in the dialplan. In the following example, calls targeted to extension 1234 (be it accessed from SIP, DAHDI or whatever channel type) are controlled by user bob@domain.com. Asterisk notifies him that a call is coming, and asks him to take an action. This dialog takes place over an XMPP chat.

```
context from-ext {
 1234 => {
   Answer();
   JabberSend(asterisk-xmpp,bob@jabber.org,Call from $CALLERID(num) -
choose an option to process the call);
   JabberSend(asterisk-xmpp,bob@jabber.org,1 : forward to cellphone);
   JabberSend(asterisk-xmpp,bob@jabber.org,2 : forward to work
phone);
   JabberSend(asterisk-xmpp,bob@jabber.org,Default action : forward
to your voicemail);
   Set(OPTION=${JABBER_RECEIVE(asterisk-xmpp,bob@jabber.org,20)});
   switch (${OPTION}) {
    case 1:
     JabberSend(asterisk-xmpp,bob@jabber.org,(Calling cellphone...);
     Dial(SIP/987654321);
     break;
    case 2:
     JabberSend(asterisk-xmpp,bob@jabber.org,(Calling workphone...);
     Dial(SIP/${EXTEN});
     break;
    default:
     Voicemail(${EXTEN}|u)
   }
  }
}
```

When calling from a GoogleTalk or Jingle client, the CALLERID(name) is set to the XMPP id of the caller (i.e. his JID). In the following example, Asterisk chats back with the caller identified by the caller id. We also take advantage of the SendText implementation in chan_gtalk (available in chan_jingle, and chan_sip as well), to allow the caller to establish SIP calls from his GoogleTalk client :

```
context gtalk-in {
 s => {
   NoOp(Caller id : ${CALLERID(all)});
   Answer();
   SendText(Please enter the number you wish to call);
   Set(NEWEXTEN=${JABBER_RECEIVE(asterisk-xmpp,${CALLERID(name)})});
   SendText(Calling ${NEWEXTEN} ...);
   Dial(SIP/${NEWEXTEN);
   Hangup();
  }
}
```

The maintainer of res_jabber is Philippe Sultan <philippe.sultan@gmail.com>.

Jingle in Asterisk

⛔ Jingle support in asterisk is experimental

Jingle is an xmpp based protocol for signalling the transfer of media. Currently asterisk supports the proprietary GoogleTalk protocol that is very similar to jingle, and hopes to soon support true jingle specs (JEP-166,167,176,177,180,181 etc) as more clients support the true standard.

Jingle's configuration is very similar to sip.conf only as we are not the jabber server in this case you must provide a connection for the peer to travel out on.

chan_gtalk is for supporting the non-jingle google/libjingle spec and chan_jingle will continue to move in the direction of the correct spec.

LDAP Realtime Driver

Asterisk Realtime LDAP Driver

With this driver Asterisk can retrieve information from an LDAP drectory, including SIP/IAX2 users, extensions and configuration.

See configs/res_ldap.conf.sample for a configuration file sample

Here is a LDAP diff sample:

```
# Base SIP Phones Entry
dn: uid=phone-base,dc=myDomain,dc=myDomainExt
objectClass: top
objectClass: AstAccount
objectClass: AstAccountSIP
uid: phone-base
AstAccountAccountingCode: baseacccode
AstAccountHost: dynamic
preferredLanguage: FR
AstAccountAMAFlags: billing
AstAccountContext: ldaptest


# A Phone. realmedPassword md5 hash should be the result of
#  echo -n "UID:SIPRealm:Password" | md5sum
dn: uid=phone-test,dc=myDomain,dc=myDomainExt
objectClass: top
objectClass: AstAccount
objectClass: AstAccountSIP
```

```
uid: phone-test
AstAccountAccountingCode: acc-phone-base
AstAccountFullContact: Noone <1234>
AstAccountCallerID: 1234
AstAccountBaseDN: uid=phone-base,dc=myDomain,dc=myDomainExt
realmedPassword: {MD5}f67965da780bf9c70d6e337f938cee6f


# extensions,
dn: ou=extensions,dc=myDomain,dc=myDomainExt
ou: extensions
objectClass: top
objectClass: organizationalUnit

# Extension 100 Priority 1 in context ldaptest
dn: cn=100-1,ou=extensions,dc=myDomain,dc=myDomainExt
AstExtensionApplication: NoOp
AstExtensionApplicationData: TEST LDAP
objectClass: top
objectClass: AstExtension
AstExtensionExten: 100
AstExtensionContext: ldaptest
cn: 100-1
AstExtensionPriority: 1

# Extension 100 Priority 1 in context ldaptest
dn: cn=100-2,ou=extensions,dc=myDomain,dc=myDomainExt
objectClass: top
objectClass: AstExtension
AstExtensionExten: 100
AstExtensionContext: ldaptest
```

```
cn: 100-2
AstExtensionPriority: 2
AstExtensionApplication: hangup
```

Open Settlement Protocol (OSP) User Guide

OSP User Guide for Asterisk V1.6

9 February 2007

Table of Contents

Asterisk is a trademark of Digium, Inc.
TransNexus and OSP Secures are trademarks of TransNexus, Inc.

Revision History
Revision Date of Issue Description

```
1        26 Jul 2005    OSP Module User Guide for Asterisk V1.2
1.4      16 Jun 2006    OSP Module User Guide for Asterisk V1.4
1.6.0    13 Dec 2006    OSP Module User Guide for Asterisk V1.6
1.6.1    4 Jan 2007     Clarifying edits, add revision history, add
general
                        purpose extensions.conf example
1.6.2    9 Feb 2007     Replace OSP Toolkit site from SIPfoundry with

                        SourceForge
```

1 Introduction

This document provides instructions on how to build and configure Asterisk V1.6 with the OSP Toolkit to enable secure, multi-lateral peering. This document is also available in the Asterisk source package as doc/osp.txt. The OSP Toolkit is an open source implementation of the OSP peering protocol and is freely available from https://sourceforge.net/projects/osp-toolkit. The OSP standard defined by the European Telecommunications Standards Institute (ETSI TS 101 321) www.etsi.org. If you have questions or need help, building Asterisk with the OSP Toolkit, please post your question on the OSP mailing list at
https://lists.sourceforge.net/lists/listinfo/osp-toolkit-client.

2 OSP Toolkit

Please reference the OSP Toolkit document "How to Build and Test the OSP Toolkit" available from https://sourceforge.net/projects/osp-toolkit.

2.1 Build OSP Toolkit

The software listed below is required to build and use the OSP Toolkit:

- OpenSSL (required for building) - Open Source SSL protocol and Cryptographic Algorithms (version 0.9.7g recommended) from www.openssl.org. Pre-compiled OpenSSL binary packages are not recommended because of the binary compatibility issue.

- Perl (required for building) - A programming language used by OpenSSL for compilation. Any version of Perl should work. One version of Perl is available from www.activestate.com/Products/ActivePer. If pre-compiled OpenSSL packages are used, Perl package is not required.

- C compiler (required for building) - Any C compiler should work. The GNU Compiler Collection from www.gnu.org is routinely used for building the OSP Toolkit for testing.

- OSP Server (required for testing) - Access to any OSP server should work. An open source reference OSP server developed by Cisco System is available at http://www.vovida.org/applications/downloads/openosp/. RAMS, a java based open source OSP server is available at https://sourceforge.net/projects/rams. A free version of the TransNexus commercial OSP server may be downloaded from http://www.transnexus.com/OSP%20Toolkit/Peering_Server/VoIP_Peering_Server.htm.

2.1.1 Unpacking the Toolkit

After downloading the OSP Toolkit (version 3.3.6 or later release) from www.sourceforge.net, perform the following steps in order:

- Copy the OSP Toolkit distribution into the directory where it will reside. The default directory for the OSP Toolkit is /usr/src.

*Un-package the distribution file by executing the following command:

```
gunzip -c OSPToolkit-###.tar.gz | tar xvf -
```

Where ### is the version number separated by underlines. For example, if the version is 3.3.6, then the above command would be:

```
gunzip -c OSPToolkit-3_3_6.tar.gz | tar xvf -
```

A new directory (TK-3_3_6-20060303) will be created within the same directory as the tar file.

- Go to the TK-3_3_6-20060303 directory by running this command:

```
cd TK-3_3_6-20060303
```

Within this directory, you will find directories and files similar to what is listed below if the command "ls -F" is executed):

```
ls -F
enroll/
RelNotes.txt    lib/
README.txt    license.txt
bin/      src/
crypto/      test/
include/
```

2.1.2 Preparing to build the OSP Toolkit

- Compile OpenSSL according to the instructions provided with the OpenSSL distribution (You would need to do this only if you don't have openssl already).

- Copy the OpenSSL header files (the *.h files) into the crypto/openssl directory within the osptoolkit directory. The OpenSSL header files are located under the openssl/include/openssl directory.

- Copy the OpenSSL library files (libcrypto.a and libssl.a) into the lib directory within the osptoolkit directory. The OpenSSL library files are located under the openssl directory.
  Note: Since the Asterisk requires the OpenSSL package. If the OpenSSL package has been installed, steps 4 through 6 are not necessary.

- Optionally, change the install directory of the OSP Toolkit. Open the Makefile in the /usr/src/TK-3_3_6-20060303/src directory, look for the install path variable – INSTALL_PATH, and edit it to be anywhere you want (defaults /usr/local).
  Note: Please change the install path variable only if you are familiar with both the OSP Toolkit and the Asterisk.

2.1.3 Building the OSP Toolkit

- From within the OSP Toolkit directory (/usr/src/TK-3_3_6-20060303), start the compilation script by executing the following commands:

```
cd src
make clean; make build
```

2.1.4 Installing the OSP Toolkit

The header files and the library of the OSP Toolkit should be installed. Otherwise, you must specify the OSP Toolkit path for the Asterisk.

- Use the make script to install the Toolkit.

```
make install
```

The make script is also used to install the OSP Toolkit header files and the library into the INSTALL_PATH specified in the Makefile.

> ⚠️ Please make sure you have the rights to access the INSTALL_PATH directory. For example, in order to access /usr/local directory, root privileges are required.

2.1.5 Building the Enrollment Utility

Device enrollment is the process of establishing a trusted cryptographic relationship between the VoIP device and the OSP Server. The Enroll program is a utility application for establishing a trusted relationship between an OSP client and an OSP server. Please see the document "Device Enrollment" at http://www.transnexus.com/OSP%20Toolkit/OSP%20Toolkit%20Documents/Device_Enrollment.p for more information about the enroll application.

- From within the OSP Toolkit directory (example: /usr/src/TK-3_3_6-20060303), execute the following commands at the command prompt:

```
cd enroll
make clean; make linux
```

Compilation is successful if there are no errors in the compiler output. The enroll program is now located in the OSP Toolkit/bin directory (example: /usr/src/ TK-3_3_6-20060303/bin).

2.2 Obtain Crypto Files

The OSP module in Asterisk requires three crypto files containing a local certificate (localcert.pem), private key (pkey.pem), and CA certificate (cacert_0.pem). Asterisk will try to load the files from the Asterisk public/private key directory - /var/lib/asterisk/keys. If the files are not present, the OSP module will not start and the Asterisk will not support the OSP protocol. Use the enroll.sh script from the toolkit distribution to enroll Asterisk with an OSP server and

obtain the crypto files. Documentation explaining how to use the enroll.sh script (Device Enrollment) to enroll with an OSP server is available at http://www.transnexus.com/OSP%20Toolkit/OSP%20Toolkit%20Documents/Device_Enrollment.p . Copy the files generated by the enrollment process to the Asterisk /var/lib/asterisk/keys directory.

> ⚠ The osptestserver.transnexus.com is configured only for sending and receiving non-SSL messages, and issuing signed tokens. If you need help, post a message on the OSP mailing list at https://lists.sourceforge.net/lists/listinfo/osp-toolkit-client

The enroll.sh script takes the domain name or IP addresses of the OSP servers that the OSP Toolkit needs to enroll with as arguments, and then generates pem files – cacert_#.pem, certreq.pem, localcert.pem, and pkey.pem. The '#' in the cacert file name is used to differentiate the ca certificate file names for the various SP's (OSP servers). If only one address is provided at the command line, cacert_0.pem will be generated. If 2 addresses are provided at the command line, 2 files will be generated – cacert_0.pem and cacert_1.pem, one for each SP (OSP server). The example below shows the usage when the client is registering with osptestserver.transnexus.com.

```
./enroll.sh osptestserver.transnexus.com
Generating a 512 bit RSA private key
.......................++++++++++
.........++++++++++
writing new private key to 'pkey.pem'
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: _____
State or Province Name (full name) [Some-State]: _____
Locality Name (eg, city) []:_____
Organization Name (eg, company) [Internet Widgits Pty Ltd]: _____
Organizational Unit Name (eg, section) []:_____
Common Name (eg, YOUR name) []:_____
Email Address []:_____


Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:_____
An optional company name []:_____


Error Code returned from openssl command : 0


CA certificate received
[SP: osptestserver.transnexus.com]Error Code returned from getcacert
command : 0


output buffer after operation: operation=request
output buffer after nonce: operation=request&nonce=1655976791184458
X509 CertInfo context is null pointer
Unable to get Local Certificate
depth=0 /CN=osptestserver.transnexus.com/O=OSPServer
verify error:num=18:self signed certificate
verify return:1
depth=0 /CN=osptestserver.transnexus.com/O=OSPServer
verify return:1
The certificate request was successful.
Error Code returned from localcert command : 0
```

The files generated should be copied to the /var/lib/asterisk/keys directory.

> ⚠ The script enroll.sh requires AT&T korn shell (ksh) or any of its compatible variants. The /usr/src/TK-3_3_6-20060303/bin directory should be in the PATH variable. Otherwise, enroll.sh cannot find the enroll file.

3 Asterisk

In Asterisk, all OSP support is implemented as dial plan functions. In Asterisk V1.6, all combinations of routing between OSP and non-OSP enabled networks using any combination of SIP, H.323 and IAX protocols are fully supported. Section
3.1 describes the three easy steps to add OSP support to Asterisk:

1. Build Asterisk with OSP Toolkit
2. Configure osp.conf file
3. Cut and paste to extensions.conf

Sections 3.2 and 3.3 provide a detailed explanation of OSP dial plan functions and configuration examples. The detailed information provided in Sections 3.2 and 3.3 is not required for operating Asterisk with OSP, but may be helpful to developers who want to customize their Asterisk OSP implementation.

3.1 Configure for OSP Support

3.1.1 Build Asterisk with OSP Toolkit

The first step is to build Asterisk with the OSP Toolkit. If the OSP Toolkit is installed in the default install directory, /usr/local, no additional configuration is required. Compile Asterisk according to the instructions provided with the Asterisk distribution.

If the OSP Toolkit is installed in another directory, such as /myosp, Asterisk must be configured with the location of the OSP Toolkit. See the example below.

```
--with-osptk=/myosp
```

> ⚠ Please change the install path only if you familiar with both the OSP Toolkit and the Asterisk. Otherwise, the change may result in Asterisk not supporting the OSP protocol.

3.1.2 osp.conf

The /etc/asterisk/osp.conf file, shown below, contains configuration parameters for using OSP. Two parameters, servicepoint and source must be configured. The default values for all other parameters will work well for standard OSP implementations.

```
;
; Open Settlement Protocol Sample Configuration File
;
; This file contains configuration of OSP server providers that
; are used by the Asterisk OSP module.  The section "general" is
; reserved for global options.  All other sections describe specific

; OSP Providers.  The provider "default" is used when no provider is

; otherwise specified.
:
: The "servicepoint" and "source" parameters must be configured.
For
; most implementations the other parameters in this file can be left

; unchanged.
;
[general]
;
; Enable cryptographic acceleration hardware.
;
accelerate=no
;
; Defines the status of tokens that Asterisk will validate.
; 0 - signed tokens only
; 1 - unsigned tokens only
; 2 - both signed and unsigned
; The default value is 0, i.e. the Asterisk will only validate
signed
; tokens.
;
tokenformat=0
;
[default]
;
; List all service points (OSP servers) for this provider.  Use
; either domain name or IP address.  Most OSP servers use port 5045.
;
;servicepoint=http://osptestserver.transnexus.com:5045/osp
servicepoint=http://OSP server IP:5045/osp
;
; Define the "source" device for requesting OSP authorization.
: This value is usually the domain name or IP address of the
: the Asterisk server.
;
;source=domain name or [IP address in brackets]
```

```
source=[host IP]
;
; Define path and file name of crypto files.
; The default path for crypto file is /var/lib/asterisk/keys.  If no
; path is defined, crypto files should be in
; /var/lib/asterisk/keys directory.
;
; Specify the private key file name.
; If this parameter is unspecified or not present, the default name
; will be the osp.conf section name followed by "-privatekey.pem"
; (for example: default-privatekey.pem)
;
privatekey=pkey.pem
;
; Specify the local certificate file.
; If this parameter is unspecified or not present, the default name
; will be the osp.conf section name followed by "- localcert.pem "
; (for example: default-localcert.pem)
;
localcert=localcert.pem
;
; Specify one or more Certificate Authority key file names.  If none

; are listed, a single Certificate Authority key file name is added
; with the default name of the osp.conf section name followed by
; "-cacert_0.pem " (for example: default-cacert_0.pem)
;
cacert=cacert_0.pem
;
; Configure parameters for OSP communication between Asterisk OSP
; client and OSP servers.
;
; maxconnections: Max number of simultaneous connections to the
;                 provider OSP server (default=20)
; retrydelay:     Extra delay between retries (default=0)
; retrylimit:     Max number of retries before giving up (default=2)
; timeout:        Timeout for response in milliseconds (default=500)
;
maxconnections=20
retrydelay=0
retrylimit=2
timeout=500
;
; Set the authentication policy.
; 0 - NO        - Accept all calls.
; 1 - YES       - Accept calls with valid token or no token.
```

```
;                        Block calls with invalid token.
; 2 - EXCLUSIVE - Accept calls with valid token.
;                        Block calls with invalid token or no token.
; Default is 1,
;
authpolicy=1
;
; Set the default destination protocol. The OSP module supports
```

```
; SIP, H323, and IAX protocols.  The default protocol is set to SIP.
;
defaultprotocol=SIP
```

3.1.3 extensions.conf

OSP functions are implemented as dial plan functions in the extensions.conf file. To add OSP support to your Asterisk server, simply copy and paste the text box below to your extensions.conf file. These functions will enable your Asterisk server to support all OSP call scenarios. Configuration of your Asterisk server for OSP is now complete.

```
[globals]
DIALOUT=DAHDI/1

[SrcGW] ; OSP Source Gateway
exten => _XXXX.,1,NoOp(OSPSrcGW)
; Set calling number if necessary
exten => _XXXX.,n,Set(CALLERID(numner)=1234567890)
; OSP lookup using default provider, if fail/error jump to
lookup+101
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||j)
; Deal with outbound call according to protocol
exten => _XXXX.,n,Macro(outbound)
; Dial to destination, 60 timeout, with call duration limit
exten =>
_XXXX.,n,Dial(${OSPDIALSTR},60,oL($[${OSPOUTTIMELIMIT}*1000]))
; Wait 1 second
exten => _XXXX.,n,Wait,1
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})

[DstGW] ; OSP Destination Gateway
exten => _XXXX.,1,NoOp(OSPDstGW)
; Deal with inbound call according to protocol
exten => _XXXX.,n,Macro(inbound)
; Validate token using default provider, if fail/error jump to
auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
```

```
; Ringing
exten => _XXXX.,n,Ringing
; Wait 1 second
exten => _XXXX.,n,Wait,1
; Check inbound call duration limit
exten => _XXXX.,n,GoToIf($[${OSPINTIMELIMIT}=0]?100:200)
; Without duration limit
exten => _XXXX.,100,Dial(${DIALOUT},15,o)
exten => _XXXX.,n,Goto(1000)
; With duration limit
exten =>
_XXXX.,200,Dial(${DIALOUT},15,oL($[${OSPINTIMELIMIT}*1000]))
exten => _XXXX.,n,Goto(1000)
; Wait 1 second
exten => _XXXX.,1000,Wait,1
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})

[GeneralProxy] ; Proxy
exten => _XXXX.,1,NoOp(OSP-GeneralProxy)
; Deal with inbound call according to protocol
exten => _XXXX.,n,Macro(inbound)
; Validate token using default provider, if fail/error jump to
auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; OSP lookup using default provider, if fail/error jump to
lookup+101
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||j)
; Deal with outbound call according to protocol
exten => _XXXX.,n,Macro(outbound)
; Dial to destination, 14 timeout, with call duration limit
exten =>
_XXXX.,n,Dial(${OSPDIALSTR},14,oL($[${OSPOUTTIMELIMIT}*1000]))
; OSP lookup next destination using default provider, if fail/error
jump to next1+101
exten => _XXXX.,n(next1),OSPNext(${HANGUPCAUSE}||j)
; Deal with outbound call according to protocol
exten => _XXXX.,n,Macro(outbound)
; Dial to destination, 15 timeout, with call duration limit
exten =>
_XXXX.,n,Dial(${OSPDIALSTR},15,oL($[${OSPOUTTIMELIMIT}*1000]))
```

```
; OSP lookup next destination using default provider, if fail/error
jump to next2+101
exten => _XXXX.,n(next2),OSPNext(${HANGUPCAUSE}||j)
; Deal with outbound call according to protocol
exten => _XXXX.,n,Macro(outbound)
; Dial to destination, 16 timeout, with call duration limit
exten =>
_XXXX.,n,Dial(${OSPDIALSTR},16,oL($[${OSPOUTTIMELIMIT}*1000]))
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
; Deal with OSPNext fail/error
exten => _XXXX.,next1+101,Hangup
; Deal with OSPNext fail/error
exten => _XXXX.,next2+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})


[macro-inbound]
exten => s,1,NoOp(inbound)
; Get inbound protocol
exten => s,n,Set(CHANTECH=${CUT(CHANNEL,/,1)})
exten => s,n,GoToIf($["${CHANTECH}"="H323"]?100)
exten => s,n,GoToIf($["${CHANTECH}"="IAX2"]?200)
exten => s,n,GoToIf($["${CHANTECH}"="SIP"]?300)
exten => s,n,GoTo(1000)
; H323 ----------------------------------------------------------
; Get peer IP
exten => s,100,Set(OSPPEERIP=${H323CHANINFO(peerip)})
; Get OSP token
exten => s,n,Set(OSPINTOKEN=${H323CHANINFO(osptoken)})
exten => s,n,GoTo(1000)
; IAX ----------------------------------------------------------
; Get peer IP
exten => s,200,Set(OSPPEERIP=${IAXPEER(CURRENTCHANNEL)})
; Get OSP token
exten => s,n,Set(OSPINTOKEN=${IAXCHANINFO(osptoken)})
exten => s,n,GoTo(1000)
; SIP ----------------------------------------------------------
; Get peer IP
exten => s,300,Set(OSPPEERIP=${SIPCHANINFO(peerip)})
; Get OSP token
```

```
exten => s,n,Set(OSPINTOKEN=${SIP_HEADER(P-OSP-Auth-Token)})
exten => s,n,GoTo(1000)
; --------------------------------------------------------------
exten => s,1000,MacroExit

[macro-outbound]
exten => s,1,NoOp(outbound)
; Set calling number which may be translated
exten => s,n,Set(CALLERID(num)=${OSPCALLING})
; Check destinatio protocol
exten => s,n,GoToIf($["${OSPTECH}"="H323"]?100)
exten => s,n,GoToIf($["${OSPTECH}"="IAX2"]?200)
exten => s,n,GoToIf($["${OSPTECH}"="SIP"]?300)
; Something wrong
exten => s,n,Hangup
exten => s,n,GoTo(1000)
; H323 ---------------------------------------------------------
; Set call id
exten => s,100,Set(H323CHANINFO(callid)=${OSPOUTCALLID})
; Set OSP token
exten => s,n,Set(H323CHANINFO(osptoken)=${OSPOUTTOKEN})
exten => s,n,GoTo(1000)
; IAX ----------------------------------------------------------
; Set OSP token
exten => s,200,Set(IAXCHANINFO(osptoken)=${OSPOUTTOKEN})
exten => s,n,GoTo(1000)
; SIP ----------------------------------------------------------
```

```
exten => s,300,GoTo(1000)
; -----------------------------------------------------------------
exten => s,1000,MacroExit
```

3.1.4 dahdi/sip/iax/h323/ooh323.conf

There is no configuration required for OSP.

3.2 OSP Dial Plan Functions

This section provides a description of each OSP dial plan function.

3.2.1 OSPAuth

OSP token validation function.
Input:

- OSPPEERIP: last hop IP address
- OSPINTOKEN: inbound OSP token
- provider: OSP service provider configured in osp.conf. If it is empty, default provider is used.
- priority jump

Output:

- OSPINHANDLE: inbound OSP transaction handle
- OSPINTIMELIMIT: inbound call duration limit
- OSPAUTHSTATUS: OSPAuth return value. SUCCESS/FAILED/ERROR

3.2.2 OSPLookup

OSP lookup function.

Input:

- OSPPEERIP: last hop IP address
- OSPINHANDLE: inbound OSP transaction handle
- OSPINTIMELIMIT: inbound call duration limit
- exten: called number
- provider: OSP service provider configured in osp.conf. If it is empty, default provider is used.
- priority jump
- callidtypes: Generate call ID for the outbound call. h: H.323; s: SIP; i: IAX. Only h, H.323, has been implemented.

Output:

- OSPOUTHANDLE: outbound transaction handle
- OSPTECH: outbound protocol
- OSPDEST: outbound destination IP address
- OSPCALLED: outbound called nummber
- OSPCALLING: outbound calling number
- OSPOUTTOKEN: outbound OSP token
- OSPRESULTS: number of remaining destinations
- OSPOUTTIMELIMIT: outbound call duration limit

- OSPOUTCALLIDTYPES: same as input callidtypes
- OSPOUTCALLID: outbound call ID. Only for H.323
- OSPDIALSTR: outbound dial string
- OSPLOOKUPSTATUS: OSPLookup return value. SUCCESS/FAILED/ERROR

3.2.3 OSPNext

OSP lookup next function.

Input:

- OSPINHANDLE: inbound transaction handle
- OSPOUTHANDLE: outbound transaction handle
- OSPINTIMELIMIT: inbound call duration limit
- OSPOUTCALLIDTYPES: types of call ID generated by Asterisk.
- OSPRESULTS: number of remain destinations
- cause: last destination disconnect cause
- priority jump

Output:

- OSPTECH: outbound protocol
- OSPDEST: outbound destination IP address
- OSPCALLED: outbound called number
- OSPCALLING: outbound calling number
- OSPOUTTOKEN: outbound OSP token
- OSPRESULTS: number of remain destinations
- OSPOUTTIMELIMIT: outbound call duration limit
- OSPOUTCALLID: outbound call ID. Only for H.323
- OSPDIALSTR: outbound dial string
- OSPNEXTSTATUS: OSPLookup return value. SUCCESS/FAILED/ERROR

3.2.4 OSPFinish

OSP report usage function.

Input:

- OSPINHANDLE: inbound transaction handle
- OSPOUTHANDLE: outbound transaction handle
- OSPAUTHSTATUS: OSPAuth return value
- OSPLOOKUPTSTATUS: OSPLookup return value
- OSPNEXTSTATUS: OSPNext return value
- cause: last destination disconnect cause
- priority jump

Output:

- OSPFINISHSTATUS: OSPLookup return value. SUCCESS/FAILED/ERROR

3.3 extensions.conf Examples

The extensions.conf file example provided in Section 3.1 is designed to handle all OSP call scenarios when Asterisk is used as a source or destination gateway to the PSTN or as a proxy between VoIP networks. The extenstion.conf examples in this section are designed for specific use cases only.

3.3.1 Source Gateway

The examples in this section apply when the Asterisk server is being used as a TDM to VoIP gateway. Calls originate on the TDM network and are converted to VoIP by Asterisk. In these cases, the Asterisk server queries an OSP server to find a route to a VoIP destination. When the call ends, Asterisk sends a CDR to the OSP server.
For SIP protocol.

```
[SIPSrcGW]
exten => _XXXX.,1,NoOp(SIPSrcGW)
; Set calling number if necessary
exten => _XXXX.,n,Set(CALLERID(numner)=CallingNumber)
; OSP lookup using default provider, if fail/error jump to
lookup+101
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||j)
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 60 timeout, with call duration limit
exten =>
_XXXX.,n,Dial(${OSPDIALSTR},60,oL($[${OSPOUTTIMELIMIT}*1000]))
; Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

For IAX protocol.

```
[IAXSrcGW]
exten => _XXXX.,1,NoOp(IAXSrcGW)
; Set calling number if necessary
exten => _XXXX.,n,Set(CALLERID(numner)=CallingNumber)
; OSP lookup using default provider, if fail/error jump to
lookup+101
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||j)
; Set outbound OSP token
exten => _XXXX.,n,Set(IAXCHANINFO(osptoken)=${OSPOUTTOKEN})
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 60 timeout, with call duration limit
exten =>
_XXXX.,n,Dial(${OSPDIALSTR},60,oL($[${OSPOUTTIMELIMIT}*1000]))
; Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

For H.323 protocol.

```
[H323SrcGW]
exten => _XXXX.,1,NoOp(H323SrcGW)
; Set calling number if necessary
exten => _XXXX.,n,Set(CALLERID(numner)=CallingNumber)
; OSP lookup using default provider, if fail/error jump to
lookup+101
; "h" parameter is used to generate a call id
; Cisco OSP gateways use this call id to validate OSP token
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||jh)
; Set outbound call id
exten => _XXXX.,n,Set(OH323CHANINFO(callid)=${OSPOUTCALLID})
; Set outbound OSP token
exten => _XXXX.,n,Set(OH323CHANINFO(osptoken)=${OSPOUTTOKEN})
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 60 timeout, with call duration limit
exten =>
_XXXX.,n,Dial(${OSPDIALSTR},60,oL($[${OSPOUTTIMELIMIT}*1000]))
; Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

3.3.2 Destination Gateway

The examples in this section apply when Asterisk is being used as a VoIP to TDM gateway. VoIP calls are received by Asterisk which validates the OSP peering token and completes to the TDM network. After the call ends, Asterisk sends a CDR to the OSP server.

For SIP protocol

```
[SIPDstGW]
exten => _XXXX.,1,NoOp(SIPDstGW)
; Get peer IP
exten => _XXXX.,n,Set(OSPPEERIP=${SIPCHANINFO(peerip)})
; Get OSP token
exten => _XXXX.,n,Set(OSPINTOKEN=${SIP_HEADER(P-OSP-Auth-Token)})
; Validate token using default provider, if fail/error jump to
auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; Ringing
exten => _XXXX.,n,Ringing
; Wait 1 second
exten => _XXXX.,n,Wait,1
; Dial phone, timeout 15 seconds, with call duration limit
exten =>
_XXXX.,n,Dial(${DIALOUTANALOG}/${EXTEN:1},15,oL($[${OSPINTIMELIMIT}*10
Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

For IAX protocol

```
[IAXDstGW]
exten => _XXXX.,1,NoOp(IAXDstGW)
; Get peer IP
exten => _XXXX.,n,Set(OSPPEERIP=${IAXPEER(CURRENTCHANNEL)})
; Get OSP token
exten => _XXXX.,n,Set(OSPINTOKEN=${IAXCHANINFO(osptoken)})
; Validate token using default provider, if fail/error jump to
auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; Ringing
exten => _XXXX.,n,Ringing
; Wait 1 second
exten => _XXXX.,n,Wait,1
; Dial phone, timeout 15 seconds, with call duration limit
exten =>
_XXXX.,n,Dial(${DIALOUTANALOG}/${EXTEN:1},15,oL($[${OSPINTIMELIMIT}*10
Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

For H.323 protocol

```
[H323DstGW]
exten => _XXXX.,1,NoOp(H323DstGW)
; Get peer IP
exten => _XXXX.,n,Set(OSPPEERIP=${H323CHANINFO(peerip)})
; Get OSP token
exten => _XXXX.,n,Set(OSPINTOKEN=${H323CHANINFO(osptoken)})
; Validate token using default provider, if fail/error jump to
auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; Ringing
exten => _XXXX.,n,Ringing
; Wait 1 second
exten => _XXXX.,n,Wait,1
; Dial phone, timeout 15 seconds, with call duration limit
exten =>
_XXXX.,n,Dial(${DIALOUTANALOG}/${EXTEN:1},15,oL($[${OSPINTIMELIMIT}*10
Wait 3 seconds
exten => _XXXX.,n,Wait,3
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

3.3.3 Proxy

The example in this section applies when Asterisk is a proxy between two VoIP networks.

```
[GeneralProxy]
exten => _XXXX.,1,NoOp(GeneralProxy)
; Get peer IP and inbound OSP token
; SIP, un-comment the following two lines.
;exten => _XXXX.,n,Set(OSPPEERIP=${SIPCHANINFO(peerip)})
;exten => _XXXX.,n,Set(OSPINTOKEN=${SIP_HEADER(P-OSP-Auth-Token)})
; IAX, un-comment the following 2 lines
;exten => _XXXX.,n,Set(OSPPEERIP=${IAXPEER(CURRENTCHANNEL)})
;exten => _XXXX.,n,Set(OSPINTOKEN=${IAXCHANINFO(osptoken)})
; H323, un-comment the following two lines.
;exten => _XXXX.,n,Set(OSPPEERIP=${OH323CHANINFO(peerip)})
;exten => _XXXX.,n,Set(OSPINTOKEN=${OH323CHANINFO(osptoken)})
;-------------------------------------------------------------
```

```
; Validate token using default provider, if fail/error jump to
auth+101
exten => _XXXX.,n(auth),OSPAuth(|j)
; OSP lookup using default provider, if fail/error jump to
lookup+101
; "h" parameter is used to generate a call id for H.323 destinations
; Cisco OSP gateways use this call id to validate OSP token
exten => _XXXX.,n(lookup),OSPLookup(${EXTEN}||jh)
; Set outbound call id and OSP token
; IAX, un-comment the following line.
;exten => _XXXX.,n,Set(IAXCHANINFO(osptoken)=${OSPOUTTOKEN})
; H323, un-comment the following two lines.
;exten => _XXXX.,n,Set(OH323CHANINFO(callid)=${OSPOUTCALLID})
;exten => _XXXX.,n,Set(OH323CHANINFO(osptoken)=${OSPOUTTOKEN})
;--------------------------------------------------------------
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 14 timeout, with call duration limit
exten =>
_XXXX.,n,Dial(${OSPDIALSTR},14,oL($[${OSPOUTTIMELIMIT}*1000]))
; OSP lookup next destination using default provider, if fail/error
jump to next1+101
exten => _XXXX.,n(next1),OSPNext(${HANGUPCAUSE}||j)
; Set outbound call id and OSP token
; IAX, un-comment the following line.
;exten => _XXXX.,n,Set(IAXCHANINFO(osptoken)=${OSPOUTTOKEN})
; H323, un-comment the following two lines.
;exten => _XXXX.,n,Set(OH323CHANINFO(callid)=${OSPOUTCALLID})
;exten => _XXXX.,n,Set(OH323CHANINFO(osptoken)=${OSPOUTTOKEN})
;--------------------------------------------------------------
; Set calling number which may be translated
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 15 timeout, with call duration limit
exten =>
_XXXX.,n,Dial(${OSPDIALSTR},15,oL($[${OSPOUTTIMELIMIT}*1000]))
; OSP lookup next destination using default provider, if fail/error
jump to next2+101
exten => _XXXX.,n(next2),OSPNext(${HANGUPCAUSE}||j)
; Set outbound call id and OSP token
; IAX, un-comment the following line.
;exten => _XXXX.,n,Set(IAXCHANINFO(osptoken)=${OSPOUTTOKEN})
; H323, un-comment the following two lines.
;exten => _XXXX.,n,Set(OH323CHANINFO(callid)=${OSPOUTCALLID})
;exten => _XXXX.,n,Set(OH323CHANINFO(osptoken)=${OSPOUTTOKEN})
;--------------------------------------------------------------
; Set calling number which may be translated
```

```
exten => _XXXX.,n,Set(CALLERID(num)=${OSPCALLING})
; Dial to destination, 16 timeout, with call duration limit
exten =>
_XXXX.,n,Dial(${OSPDIALSTR},16,oL($[${OSPOUTTIMELIMIT}*1000]))
; Hangup
exten => _XXXX.,n,Hangup
; Deal with OSPAuth fail/error
exten => _XXXX.,auth+101,Hangup
; Deal with OSPLookup fail/error
exten => _XXXX.,lookup+101,Hangup
; Deal with 1st OSPNext fail/error
exten => _XXXX.,next1+101,Hangup
; Deal with 2nd OSPNext fail/error
exten => _XXXX.,next2+101,Hangup
```

```
exten => h,1,NoOp()
; OSP report usage
exten => h,n,OSPFinish(${HANGUPCAUSE})
```

PSTN Connectivity

Advice of Charge

Written by: David Vossel
Initial version: 04-19-2010
Email: dvossel@digium.com

This document is designed to give an overview of how to configure and generate Advice of Charge along with a detailed explanation of how each option works.

Read This First

PLEASE REPORT ANY ISSUES ENCOUNTERED WHILE USING AOC. This feature has had very little community feedback so far. If you are using this feature please share with us any problems you are having and any improvements that could make this feature more useful. Thank you!

Terminology

AOC: Advice of Charge

AOC-S: Advice of Charge message sent at the beginning of a call during call setup. This message contains a list of rates associated with the call.

AOC-D: Advice of Charge message sent during the call. This message is typically used to update the endpoint with the current call charge.

AOC-E: Advice of Charge message sent at the end of a call. This message is used to indicate to the endpoint the final call charge.

AMI: Asterisk Manager Interface. This interface is used to generate AOC messages and listen for AOC events.

AOC in chan_dahdi

LibPRI Support:
ETSI, or euroisdn, is the only switchtype that LibPRI currently supports for AOC.

Enable AOC Pass-through in chan_dahdi
To enable AOC pass-through between the ISDN and Asterisk use the 'aoc_enable' config option. This option allows for any combination of AOC-S, AOC-D, and AOC-E to be enabled or disabled.

For example:

```
aoc_enable=s,d,e ; enables pass-through of AOC-S, AOC-D, and AOC-E

aoc_enable=s,d   ; enables pass-through of AOC-S and AOC-D. Rejects
                 ; AOC-E and AOC-E request messages
```

Since AOC messages are often transported on facility messages, the 'facilityenable' option must be enabled as well to fully support AOC pass-through.

Handling AOC-E in chan_dahdi
Whenever a dahdi channel receives an AOC-E message from Asterisk, it stores that message to deliver it at the appropriate time during call termination. This means that if two AOC-E messages are received on the same call, the last one will override the first one and only one AOC-E message will be sent during call termination.

There are some tricky situations involving the final AOC-E message. During a bridged call, if the endpoint receiving the AOC messages terminates the call before the endpoint delivering the AOC does, the final AOC-E message sent by the sending side during termination will never make it to the receiving end because Asterisk will have already torn down that channel.
This is where the chan_dahdi.conf 'aoce_delayhangup' option comes into play.

By enabling 'aoce_delayhangup', anytime a hangup is initiated by the ISDN side of an Asterisk channel, instead of hanging up the channel, the channel sends a unique internal AOC-E termination request to its bridge channel. This indicates it is about to hangup and wishes to receive the final AOC-E message from the bridged channel before completely tearing down. If the bridged channel knows what to do with this AOC-E termination request, it will do whatever is necessary to indicate to its endpoint that the call is being terminated without actually hanging up the Asterisk channel. This allows the final AOC-E message to come in and be sent across the bridge while both channels are still up. If the channel delaying its hangup for the final AOC-E message times out, the call will be torn down just as it normally would. In chan_dahdi the timeout period is 1/2 the T305 timer which by default is 15 seconds.

'aoce_delayhangup' currently only works when both bridged channels are dahdi_channels. If a SIP channel receives an AOC-E termination request, it just responds by immediately hanging up the channel. Using this option when bridged to any channel technology besides SIP or DAHDI will result in the 15 second timeout period before tearing down the call completely.

Requesting AOC services

AOC can be requested on a call by call basis using the DAHDI dialstring option, A(). The A() option takes in 's', 'd', and 'e' parameters which represent the three types of AOC messages, AOC-S, AOC-D, and AOC-E. By using this option Asterisk will indicate to the endpoint during call setup that it wishes to receive the specified forms of AOC during the call.

Example Usage in extensions.conf

```
exten => 1111,1,Dial(DAHDI/g1/1112/A(s,d,e) ; requests AOC-S, AOC-D,
and AOC-E on
; call setup
exten => 1111,1,Dial(DAHDI/g1/1112/A(d,e)  ; requests only AOC-D,
and AOC-E on
; call setup
```

AOC in chan_sip

Asterisk supports a very basic way of sending AOC on a SIP channel to Snom phones using an AOC specification designed by Snom. This support is limited to the sending of AOC-D and AOC-E pass-through messages. No support for AOC-E on call termination is present, so if the Snom endpoint receiving the AOC messages from Asterisk terminates the call, the channel will be torn down before the phone can receive the final AOC-E message.

To enable passthrough of AOC messages via the snom specification, use the 'snom_aoc_enabled' option in sip.conf.

Generate AOC Messages via AMI

Asterisk supports a way to generate AOC messages on a channel via the AMI action AOCMessage. At the moment the AOCMessage action is limited to AOC-D and AOC-E message generation. There are some limitations involved with delivering the final AOC-E message as well. The AOCMessage action has its own detailed parameter documentation so this discussion will focus on higher level use. When generating AOC messages on a Dahdi channel first make sure the appropriate chan_dahdi.conf options are enabled. Without enabling 'aoc_enable' correctly for pass-through the AOC messages will never make it out the pri. The same goes with SIP, the 'snom_aoc_enabled' option must be configured before messages can successfully be set to the endpoint.

AOC-D Message Generation
AOC-D message generation can happen anytime throughout the call. This message type is very straight forward.

Example: AOCMessage action generating AOC-D currency message with Success response.

```
Action: AOCMessage
Channel: DAHDI/i1/1111-1
MsgType: d
ChargeType: Currency
CurrencyAmount: 16
CurrencyName: USD
CurrencyMultiplier: OneThousandth
AOCBillingId: Normal
ActionID: 1234


Response: Success
ActionID: 1234
Message: AOC Message successfully queued on channel
```

AOC-E Message Generation

AOC-E messages are sent during call termination and represent the final charge total for the call. Since Asterisk call termination results in the channel being destroyed, it is currently not possible for the AOCMessage AMI action to be used to send the final AOC-E message on call hangup. There is however a work around for this issue that can be used for Dahdi channels. By default chan_dahdi saves any AOC-E message it receives from Asterisk during a call and waits to deliver that message during call termination. If multiple AOC-E messages are received from Asterisk on the same Dahdi channel, only the last message received is stored for delivery. This means that each new AOC-E message received on the channel overrides the previous one. Knowing this the final AOC-E message can be continually updated on a Dahdi channel until call termination occurs allowing the last update to be sent on hangup. This method is only as accurate as the intervals in which it is updated, but allows some form of AOC-E to be generated.

Example: AOCMessage action generating AOC-E unit message with Success response.

```
Action: AOCMessage
Channel: DAHDI/i1/1111-1
MsgType: e
ChargeType: Unit
UnitAmount(0): 111
UnitType(0): 6
UnitAmount(1): 222
UnitType(1): 5
UnitAmount(2): 333
UnitType(3): 4
UnitAmount(4): 444
AOCBillingId: Normal
ActionID: 1234

Response: Success
ActionID: 1234
Message: AOC Message successfully queued on channel
```

Caller ID in India

India finds itself in a unique situation (hopefully). It has several telephone line providers, and they are not all using the same CID signaling; and the CID signalling is not like other countries.

In order to help those in India quickly find to the CID signaling system that their carrier uses (or range of them), and get the configs right with a minimal amount of experimentation, this file is provided. Not all carriers are covered, and not all mentioned below are complete. Those with updates to this table should post the new information on bug 6683 of the asterisk bug tracker.

Provider: Bharti (is this BSNL?)
Config:

```
cidstart=polarity_in
cidsignalling=dtmf
```

Results: ? (this should work), but needs to be tested?

Tested by: ?

Provider: VSNL
Config:

```
null
```

Results: ?

Tested by: ?

Provider: BSNL
Config:

```
cid_start=ring
cid_signalling=dtmf
```

Results: ?

Tested by: (abhi)

Provider: MTNL, old BSNL
Config:

```
cidsignalling = v23
cidstart=ring
```

Results: works

Tested by: (enterux)

Provider: MTNL (Delhi)
Config:

```
cidsignalling = v23
cidstart = ring
```

or:

```
cidsignalling = dtmf
cidstart = polarity_IN
```

or:

```
cidsignalling = dtmf
cidstart = polarity
```

Results: fails

Tested by: brealer

Provider: TATA

Config:

```
cidsignalling = dtmf
cidstart=polarity_IN
```

Results: works

Tested by: brealer

Asterisk still doesn't work with some of the CID scenarios in India. If you are in India, and not able to make CID work with any of the permutations of cidsignalling and cidstart, it could be that this particular situation is not covered by Asterisk. A good course of action would be to get in touch with the provider, and find out from them exactly how their CID signalling works. Describe this to us, and perhaps someone will be able to extend the code to cover their signaling.

Signaling System Number 7

Tested Switches:

- Siemens EWSD - (ITU style) MTP2 and MTP3 comes up, ISUP inbound and outbound calls work as well.
- DTI DXC 4K - (ANSI style) 56kbps link, MTP2 and MTP3 come up, ISUP inbound and outbound calls work as well.
- Huawei M800 - (ITU style) MTP2 and MTP3 comes up, ISUP National, International inbound and outbound calls work as well, CallerID presentation&screening work.

and MORE~!

Thanks:

Mark Spencer, for writing Asterisk and libpri and being such a great friend and boss.

Luciano Ramos, for donating a link in getting the first "real" ITU switch working.

Collin Rose and John Lodden, John for introducing me to Collin, and Collin for the first "real" ANSI link and for holding my hand through the remaining changes that had to be done for ANSI switches.

To Use:

In order to use libss7, you must get at least the following versions of DAHDI and Asterisk:
DAHDI: 2.0.x
libss7: trunk (currently, there **only** is a trunk release).
Asterisk: 1.6.x

You must then do a `make; make install` in each of the directories that you installed in the given order (DAHDI first, libss7 second, and Asterisk last).

⚠
In order to check out the code, you must have the subversion client installed. This is how to check them out from the public subversion server.

These are the commands you would type to install them:

```
`svn co http://svn.digium.com/svn/dahdi/linux/trunk dahdi-trunk`
`cd dahdi-trunk`
`make; make install`

`svn co http://svn.digium.com/svn/dahdi/tools/trunk dahdi-tools`
`cd dahdi-tools`
`./configure; make; make install`

`svn co http://svn.digium.com/svn/libss7/trunk libss7-trunk`
`cd libss7-trunk`
`make; make install`

`svn co http://svn.digium.com/svn/asterisk/trunk asterisk-trunk`
`cd asterisk-trunk`
`./configure; make; make install;`
```

This should build DAHDI, libss7, and Asterisk with SS7 support.

In the past, there was a special asterisk-ss7 branch to use which contained the SS7 code. That code has been merged back into the trunk version of Asterisk, and the old asterisk-ss7 branch has been deprecated and removed. If you are still using the asterisk-ss7 branch, it will not work against the current version of libss7, and you should switch to asterisk-trunk instead.

Configuration:

In /etc/dahdi/system.conf, your signalling channel(s) should be a "dchan" and your bearers should be set as "bchan".

The sample chan_dahdi.conf contains sample configuration for setting up an E1 link.

In brief, here is a simple ss7 linkset setup:

```
signalling = ss7
ss7type = itu    ; or ansi if you are using an ANSI link

linkset = 1  ; Pick a number for your linkset identifier in
chan_dahdi.conf

pointcode = 28  ; The decimal form of your point code.  If you are
using an
    ; ANSI linkset, you can use the xxx-xxx-xxx notation for
```

```
    ; specifying your link
adjpointcode = 2 ; The point code of the switch adjacent to your
linkset

defaultdpc = 3  ; The point code of the switch you want to send your
ISUP
    ; traffic to.  A lot of the time, this is the same as your
    ; adjpointcode.

; Now we configure our Bearer channels (CICs)

cicbeginswith = 1 ; Number to start counting the CICs from.  So if
DAHDI/1 to
    ; DAHDI/15 are CICs 1-15, you would set this to 1 before you
    ; declare channel=1-15

channel=1-15  ; Use DAHDI/1-15 and assign them to CICs 1-15

cicbeginswith = 17 ; Now for DAHDI/17 to DAHDI/31, they are CICs
17-31 so we initialize
    ; cicbeginswith to 17 before we declare those channels

channel = 17-31  ; This assigns CICs 17-31 to channels 17-31

sigchan = 16  ; This is where you declare which DAHDI channel is
your signalling
    ; channel.  In our case it is DAHDI/16.  You can add redundant
    ; signalling channels by adding additional sigchan= lines.

; If we want an alternate redundant signalling channel add this

sigchan = 48  ; This would put two signalling channels in our
linkset, one at
    ; DAHDI/16 and one at DAHDI/48 which both would be used to
send/receive
```

```
    ; ISUP traffic.

; End of chan_dahdi.conf
```

This is how a basic linkset is setup. For more detailed chan_dahdi.conf SS7 config information as well as other options available for that file, see the default chan_dahdi.conf that comes with the samples in asterisk. If you would like, you can do a `make samples` in your asterisk-trunk directory and it will install a sample chan_dahdi.conf for you that contains more information about SS7 setup.

For more information, please use the asterisk-ss7 or asterisk-dev mailing lists (I monitor them regularly) or email me directly.

Matthew Fredrickson
creslin@digium.com

Real-time Text (T.140)

Real-time text in Asterisk

The SIP channel has support for real-time text conversation calls in Asterisk (T.140). This is a way to perform text based conversations in combination with other media, most often video. The text is sent character by character as a media stream.

During a call sometimes there are losses of T.140 packets and a solution to that is to use redundancy in the media stream (RTP). See "http://en.wikipedia.org/wiki/Text_over_IP"http://en.wikipedia.org/wiki/Text_over_IP and RFC 5194 for more information.

The supported real-time text codec is t.140.

Real-time text redundancy support is now available in Asterisk.

ITU-T T.140

You can find more information about T.140 at www.itu.int. RTP is used for the transport T.140, as specified in RFC 4103.

How to enable T.140

In order to enable real-time text with redundancy in Asterisk, modify sip.conf to add:

```
[general]
disallow=all
allow=ulaw
allow = alaw
allow=t140
allow=t140red
textsupport=yes
videosupport=yes ; needed for proper SDP handling even if only text
and voice calls are handled
allow=h263 ; at least one video codec as H.261, H.263 or H.263+ is
needed.
```

The codec settings may change, depending on your phones. The important settings here are to allow t140 and 140red and enable text support.

General information about real-time text support in Asterisk

With the configuration above, calls will be supported with any combination of real-time text, audio and video.

Text for both t140 and t140red is handled on channel and application level in Asterisk conveyed in Text frames, with the subtype "t140". Text is conveyed in such frames usually only containing one or a few characters from the real-time text flow. The packetization interval is 300 ms, handled on lower RTP level, and transmission redundancy level is 2, causing one original and two redundant transmissions of all text so that it is reliable even in high packet loss situations. Transmitting applications do not need to bother about the transmission interval. The t140red support handles any buffering needed during the packetization intervals.

Clients known to support text, audio/text or audio/video/text calls with Asterisk:

- Omnitor Allan eC - SIP audio/video/text softphone
- AuPix APS-50 - audio/video/text softphone.
- France Telecom eConf –audio/video/text softphone.
- SIPcon1 - open source SIP audio/text softphone available in Sourceforge.

Limitations

A known general problem with Asterisk is that when a client which uses audio/video/T.140 calls to an Asterisk with T.140 media offered but video support not specified. In this case Asterisk handles the sdp media description (m=) incorrectly, and the sdp response is not created correctly. To solve this problem, turn on video support in Asterisk.

Modify sip.conf to add

```
[general]
videosupport=yes
allow=h263 ; at least one video codec as H.261, H.263 or H.263+ is
needed.
```

The problem with sdp is a bug and is reported to Asterisk bugtracker, it has id 12434

Credits

- Asterisk real-time text support is developed by AuPix
- Asterisk real-time text redundancy support is developed by Omnitor

The work with Asterisk real-time text redundancy was supported with funding from the National Institute on Disability and Rehabilitation Research (NIDRR), U.S. Department of Education, under grant number H133E040013 as part of a co-operation between the Telecommunication Access Rehabilitation Engineering Research Center of the University of Wisconsin â€" Trace Center joint with Gallaudet University, and Omnitor.

Olle E. Johansson, Edvina AB, has been a liason between the Asterisk project and this project.

RTP Packetization

Overview

Asterisk currently supports configurable RTP packetization per codec for select RTP-based channels.

Channels

These channel drivers allow RTP packetization on a user/peer/friend or global level:

- chan_sip
- chan_skinny
- chan_h323
- chan_ooh323 (Asterisk-Addons)
- chan_gtalk
- chan_jingle

Configuration

To set a desired packetization interval on a specific codec, append that inteval to the allow= statement.

Example:

```
allow=ulaw:30,alaw,g729:60
```

No packetization is specified in the case of alaw in this example, so the default of 20ms is used.

Autoframing

In addition, chan_sip has the ability to negotiate the desired framing at call establishment.

In sip.conf if autoframing=yes is set in the global section, then all calls will try to set the packetization based on the remote endpoint's preferences. This behaviour depends on the endpoints ability to present the desired packetization (ptime 🙂 in the SDP. If the endpoint does not include a ptime attribute, the call will be established with 20ms packetization.

Autoframing can be set at the global level or on a user/peer/friend basis. If it is enabled at the global level, it applies to all users/peers/friends regardless of their prefered codec packetization.

Codec framing options

The following table lists the minimum and maximum values that are valid per codec, as well as the increment value used for each. Please note that the maximum values here are only recommended maximums, and should not exceed the RTP MTU.

```
Name   Min   Max   Default   Increment
g723   30   300   30   30
gsm   20   300   20   20
ulaw   10   150   20   10
alaw   10   150   20   10
g726   10   300   20   10
ADPCM   10   300   20   10
SLIN   10   70   20   10
lpc10   20   20   20   20
g729   10   230   20   10
speex   10   60   20   10
ilbc   30   30   30   30
g726_aal2 10   300   20   10
```

Invalid framing options are handled based on the following rules:

1. If the specified framing is less than the codec's minimum, then the minimum value is used.
2. If the specific framing is greater than the codec's maximum, then the maximum value is used
3. If the specificed framing does not meet the increment requirement, the specified framing is rounded down to the closest valid framing options.

Simple Message Desk Interface (SMDI) Integration

Accessing SMDI information in the dialplan.

There are two dialplan functions that can be used to access the details of incoming SMDI messages.

```
*CLI> core show function SMDI_MSG_RETRIEVE

   -= Info about function 'SMDI_MSG_RETRIEVE' =-
```

Syntax

```
SMDI_MSG_RETRIEVE(<smdi port>,<search key>[,timeout[,options]])
```

Synopsis

Retrieve an SMDI message.

Description

This function is used to retrieve an incoming SMDI message. It returns an ID which can be used
with the SMDI_MSG() function to access details of the message. Note that this is a destructive
function in the sense that once an SMDI message is retrieved using this function, it is no longer
in the global SMDI message queue, and can not be accessed by any other Asterisk channels.
The timeout for this function is optional, and the default is 3 seconds. When providing a timeout,
it should be in milliseconds. The default search is done on the forwarding station ID. However, if
you set one of the search key options in the options field, you can change this behavior.

Options

- t - Instead of searching on the forwarding station, search on the message desk terminal.
- n - Instead of searching on the forwarding station, search on the message desk number.

```
*CLI> core show function SMDI_MSG

   -= Info about function 'SMDI_MSG' =-
```

Syntax

```
SMDI_MSG(<message_id>,<component>)
```

Synopsis

Retrieve details about an SMDI message.

Description

This function is used to access details of an SMDI message that was pulled from the incoming
SMDI message queue using the SMDI_MSG_RETRIEVE() function. Valid message components
are:

- station - The forwarding station
- callerid - The callerID of the calling party that was forwarded
- type - The call type. The value here is the exact character that came in on the SMDI link. Typically, example values are: D - Direct Calls,
  A - Forward All Calls, B - Forward Busy Calls, N - Forward No Answer Calls

Here is an example of how to use these functions:

```
; Retrieve the SMDI message that is associated with the number that
; was called in Asterisk.
exten =>
_0XXX,1,Set(SMDI_MSG_ID=${SMDI_MSG_RETRIEVE(/dev/tty0,${EXTEN})})

; Ensure that the message was retrieved.
exten => _0XXX,n,GotoIf($["x${SMDI_MSG_ID}" !=
"x"]?processcall:hangup)
exten => _0XXX,n(hangup),NoOp(No SMDI message retrieved for
${EXTEN})

; Grab the details out of the SMDI message.
exten => _0XXX,n(processcall),NoOp(Message found for ${EXTEN})
exten => _0XXX,n,Set(SMDI_EXTEN=${SMDI_MSG(${SMDI_MSG_ID},station)})
exten => _0XXX,n,Set(SMDI_CID=${SMDI_MSG(${SMDI_MSG_ID},callerid)})

; Map SMDI message types to the right voicemail option.  If it is
"B", use the
; busy option.  Otherwise, use the unavailable option.
exten => _0XXX,n,GotoIf($["${SMDI_MSG(${SMDI_MSG_ID},type)}" ==
"B"]?usebusy:useunavail)

exten => _0XXX,n(usebusy),Set(SMDI_VM_TYPE=b)
exten => _0XXX,n,Goto(continue)

exten => _0XXX,n,(useunavil),Set(SMDI_VM_TYPE=u)

exten => _0XXX,n(continue),NoOp( Process the rest of the call ... )
```

Ensuring complete MWI information over SMDI

Another change has been made to ensure that MWI state is properly propagated over the SMDI link. This replaces the use of externnotify=smdi for voicemail.conf. The issue is that we have to poll mailboxes occasionally for changes that were made using an IMAP client. So, this ability was added to res_smdi. To configure this, there is a new section in smdi.conf. It looks like this:

```
[mailboxes]
; This section configures parameters related to MWI handling for the
SMDI link.
 ;
; This option configures the polling interval used to check to see
if the
; mailboxes have any new messages.  This option is specified in
seconds.
; The default value is 10 seconds.
;
;pollinginterval=10
;
; Before specifying mailboxes, you must specify an SMDI interface.
All mailbox
; definitions that follow will correspond to that SMDI interface.
If you
; specify another interface, then all definitions following that
will correspond
; to the new interface.
;
; Every other entry in this section of the configuration file is
interpreted as
; a mapping between the mailbox ID on the SMDI link, and the local
Asterisk
; mailbox name.  In many cases, they are the same thing, but they
still must be
; listed here so that this module knows which mailboxes it needs to
pay
; attention to.
;
; Syntax:
;    <SMDI mailbox ID>=<Asterisk Mailbox Name>[@Asterisk Voicemail
Context]
;
; If no Asterisk voicemail context is specified, "default" will be
assumed.
;
;
;smdiport=/dev/ttyS0
;2565551234=1234@vmcontext1
;2565555678=5678@vmcontext2
;smdiport=/dev/ttyS1
;2565559999=9999
```

Simple Network Management Protocol (SNMP) Support

Rudimentary support for SNMP access to Asterisk is available. To build this, one needs to have Net-SNMP development headers and libraries on the build system, including any libraries Net-SNMP depends on.

Note that on some (many?) Linux-distributions the dependency list in the net-snmp-devel list is not complete, and additional packages will need to be installed. This is usually seen as configure failing to detect net-snmp-devel as the configure script does a sanity check of the net-snmp build environment, based on the output of 'net-snmp-config --agent-libs'.

To see what your distribution requires, run:

```
'net-snmp-config --agent-libs'.
```

You will receive a response similar to the following:

```
-L/usr/lib -lnetsnmpmibs -lnetsnmpagent -lnetsnmphelpers -lnetsnmp
-ldl
-lrpm -lrpmio -lpopt -lz -lcrypto -lm -lsensors -L/usr/lib/lib
-lwrap
-Wl,-E -Wl,-rpath,/usr/lib/perl5/5.8.8/i386-linux-thread-multi/CORE
-L/usr/local/lib
/usr/lib/perl5/5.8.8/i386-linux-thread-multi/auto/DynaLoader/DynaLoade

-L/usr/lib/perl5/5.8.8/i386-linux-thread-multi/CORE -lperl -lresolv
-lnsl
-ldl -lm -lcrypt -lutil -lpthread -lc
```

The packages required may include the following:

- bzip2-devel
- lm_sensors-devel
- newt-devel

SNMP support comes in two varieties – as a sub-agent to a running SNMP daemon using the AgentX protocol, or as a full standalone agent. If you wish to run a full standalone agent, Asterisk must run as root in
order to bind to port 161.

Configuring access when running as a full agent is something that is left as an exercise to the reader.

To enable access to the Asterisk SNMP subagent from a master SNMP daemon, one will need to enable AgentX support, and also make sure that Asterisk will be able to access the Unix domain socket. One way of doing this is to add the following to /etc/snmp/snmpd.conf:

```
# Enable AgentX support
master agentx

# Set permissions on AgentX socket and containing
# directory such that process in group 'asterisk'
# will be able to connect
agentXPerms  0660 0550 nobody asterisk
```

This assumes that you run Asterisk under group 'asterisk' (and does not care what user you run as).

Asterisk MIB Definitions

```
ASTERISK-MIB DEFINITIONS ::= BEGIN

IMPORTS
 OBJECT-TYPE, MODULE-IDENTITY, Integer32, Counter32, TimeTicks,
 Unsigned32, Gauge32
  FROM SNMPv2-SMI

 TEXTUAL-CONVENTION, DisplayString, TruthValue
  FROM SNMPv2-TC

 digium
  FROM DIGIUM-MIB;

asterisk MODULE-IDENTITY
 LAST-UPDATED "200806202025Z"
 ORGANIZATION "Digium, Inc."
 CONTACT-INFO
  "Mark A. Spencer
   Postal: Digium, Inc.
           445 Jan Davis Drive
           Huntsville, AL 35806
           USA
      Tel: +1 256 428 6000
    Email: markster@digium.com

   Thorsten Lockert
   Postal: Voop AS
           Boehmergaten 42
    NO-5057 Bergen
    Norway
      Tel: +47 5598 7200
    Email: tholo@voop.no"
```

```
   DESCRIPTION
    "Asterisk is an Open Source PBX.  This MIB defined
    objects for managing Asterisk instances."
   REVISION "200806202025Z"
   DESCRIPTION
    "smilint police --
     Add missing imports; fix initial capitalization
     of enumeration elements; add missing range
     restrictions for Integer32 indices, correct
     spelling of astChanCidANI in its definition.
     Addresses bug 12905. - jeffg@opennms.org"
   REVISION "200708211450Z"
   DESCRIPTION
    "Add total and current call counter statistics."
   REVISION "200603061840Z"
   DESCRIPTION
    "Change audio codec identification from 3kAudio to
    Audio3k to conform better with specification.

    Expand on contact information."
   REVISION "200602041900Z"
   DESCRIPTION
    "Initial published revision."
   ::= { digium 1 }

asteriskVersion  OBJECT IDENTIFIER ::= { asterisk 1 }
asteriskConfiguration OBJECT IDENTIFIER ::= { asterisk 2 }
asteriskModules  OBJECT IDENTIFIER ::= { asterisk 3 }
asteriskIndications OBJECT IDENTIFIER ::= { asterisk 4 }
asteriskChannels OBJECT IDENTIFIER ::= { asterisk 5 }

-- asteriskVersion

astVersionString OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Text version string of the version of Asterisk that
   the SNMP Agent was compiled to run against."
 ::= { asteriskVersion 1 }

astVersionTag OBJECT-TYPE
 SYNTAX  Unsigned32
 MAX-ACCESS read-only
 STATUS  current
```

```
  DESCRIPTION
   "SubVersion revision of the version of Asterisk that
   the SNMP Agent was compiled to run against -- this is
   typically 0 for release-versions of Asterisk."
  ::= { asteriskVersion 2 }


-- asteriskConfiguration

astConfigUpTime OBJECT-TYPE
 SYNTAX   TimeTicks
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Time ticks since Asterisk was started."
  ::= { asteriskConfiguration 1 }


astConfigReloadTime OBJECT-TYPE
 SYNTAX   TimeTicks
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Time ticks since Asterisk was last reloaded."
  ::= { asteriskConfiguration 2 }


astConfigPid OBJECT-TYPE
 SYNTAX   Integer32
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "The process id of the running Asterisk process."
  ::= { asteriskConfiguration 3 }


astConfigSocket OBJECT-TYPE
 SYNTAX   DisplayString
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "The control socket for giving Asterisk commands."
  ::= { asteriskConfiguration 4 }


astConfigCallsActive OBJECT-TYPE
 SYNTAX   Gauge32
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "The number of calls currently active on the Asterisk PBX."
```

```
  ::= { asteriskConfiguration 5 }

astConfigCallsProcessed OBJECT-TYPE
 SYNTAX   Counter32
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "The total number of calls processed through the Asterisk PBX
since last
   restart."
 ::= { asteriskConfiguration 6 }

-- asteriskModules

astNumModules OBJECT-TYPE
 SYNTAX   Integer32
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Number of modules currently loaded into Asterisk."
 ::= { asteriskModules 1 }

-- asteriskIndications

astNumIndications OBJECT-TYPE
 SYNTAX   Integer32
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Number of indications currently defined in Asterisk."
 ::= { asteriskIndications 1 }

astCurrentIndication OBJECT-TYPE
 SYNTAX   DisplayString
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Default indication zone to use."
 ::= { asteriskIndications 2 }

astIndicationsTable OBJECT-TYPE
 SYNTAX   SEQUENCE OF AstIndicationsEntry
 MAX-ACCESS not-accessible
 STATUS   current
 DESCRIPTION
   "Table with all the indication zones currently know to
```

```
  the running Asterisk instance."
 ::= { asteriskIndications 3 }

astIndicationsEntry OBJECT-TYPE
 SYNTAX  AstIndicationsEntry
 MAX-ACCESS not-accessible
 STATUS  current
 DESCRIPTION
   "Information about a single indication zone."
 INDEX  { astIndIndex }
 ::= { astIndicationsTable 1 }

AstIndicationsEntry ::= SEQUENCE {
 astIndIndex  Integer32,
 astIndCountry  DisplayString,
 astIndAlias  DisplayString,
 astIndDescription DisplayString
}

astIndIndex OBJECT-TYPE
 SYNTAX  Integer32 (1 .. 2147483647)
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Numerical index into the table of indication zones."
 ::= { astIndicationsEntry 1 }

astIndCountry OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Country for which the indication zone is valid,
   typically this is the ISO 2-letter code of the country."
 ::= { astIndicationsEntry 2 }

astIndAlias OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   ""
 ::= { astIndicationsEntry 3 }

astIndDescription OBJECT-TYPE
 SYNTAX  DisplayString
```

```
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Description of the indication zone, usually the full
  name of the country it is valid for."
 ::= { astIndicationsEntry 4 }

-- asteriskChannels

astNumChannels OBJECT-TYPE
 SYNTAX  Gauge32
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Current number of active channels."
 ::= { asteriskChannels 1 }

astChanTable OBJECT-TYPE
 SYNTAX  SEQUENCE OF AstChanEntry
 MAX-ACCESS not-accessible
 STATUS  current
 DESCRIPTION
  "Table with details of the currently active channels
  in the Asterisk instance."
 ::= { asteriskChannels 2 }

astChanEntry OBJECT-TYPE
 SYNTAX  AstChanEntry
 MAX-ACCESS not-accessible
 STATUS  current
 DESCRIPTION
  "Details of a single channel."
 INDEX  { astChanIndex }
 ::= { astChanTable 1 }

AstChanEntry ::= SEQUENCE {
 astChanIndex  Integer32,
 astChanName  DisplayString,
 astChanLanguage  DisplayString,
 astChanType  DisplayString,
 astChanMusicClass DisplayString,
 astChanBridge  DisplayString,
 astChanMasq  DisplayString,
 astChanMasqr  DisplayString,
 astChanWhenHangup TimeTicks,
 astChanApp  DisplayString,
```

```
astChanData  DisplayString,
astChanContext  DisplayString,
astChanMacroContext DisplayString,
astChanMacroExten DisplayString,
astChanMacroPri  Integer32,
astChanExten  DisplayString,
astChanPri  Integer32,
astChanAccountCode DisplayString,
astChanForwardTo DisplayString,
astChanUniqueId  DisplayString,
astChanCallGroup Unsigned32,
astChanPickupGroup Unsigned32,
astChanState  INTEGER,
astChanMuted  TruthValue,
astChanRings  Integer32,
astChanCidDNID  DisplayString,
astChanCidNum  DisplayString,
astChanCidName  DisplayString,
astChanCidANI  DisplayString,
astChanCidRDNIS  DisplayString,
astChanCidPresentation DisplayString,
astChanCidANI2  Integer32,
astChanCidTON  Integer32,
astChanCidTNS  Integer32,
astChanAMAFlags  INTEGER,
astChanADSI  INTEGER,
astChanToneZone  DisplayString,
astChanHangupCause INTEGER,
astChanVariables DisplayString,
astChanFlags  BITS,
astChanTransferCap INTEGER
}

astChanIndex OBJECT-TYPE
 SYNTAX  Integer32 (1 .. 2147483647)
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Index into the channel table."
 ::= { astChanEntry 1 }

astChanName OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
```

```
  "Name of the current channel."
 ::= { astChanEntry 2 }


astChanLanguage OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Which language the current channel is configured to
  use -- used mainly for prompts."
 ::= { astChanEntry 3 }


astChanType OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Underlying technology for the current channel."
 ::= { astChanEntry 4 }


astChanMusicClass OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Music class to be used for Music on Hold for this
  channel."
 ::= { astChanEntry 5 }


astChanBridge OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Which channel this channel is currently bridged (in a
  conversation) with."
 ::= { astChanEntry 6 }


astChanMasq OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Channel masquerading for us."
 ::= { astChanEntry 7 }
```

```
astChanMasqr OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Channel we are masquerading for."
 ::= { astChanEntry 8 }

astChanWhenHangup OBJECT-TYPE
 SYNTAX  TimeTicks
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "How long until this channel will be hung up."
 ::= { astChanEntry 9 }

astChanApp OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Current application for the channel."
 ::= { astChanEntry 10 }

astChanData OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Arguments passed to the current application."
 ::= { astChanEntry 11 }

astChanContext OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Current extension context."
 ::= { astChanEntry 12 }

astChanMacroContext OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Current macro context."
```

```
 ::= { astChanEntry 13 }

astChanMacroExten OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Current macro extension."
 ::= { astChanEntry 14 }

astChanMacroPri OBJECT-TYPE
 SYNTAX  Integer32
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Current macro priority."
 ::= { astChanEntry 15 }

astChanExten OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Current extension."
 ::= { astChanEntry 16 }

astChanPri OBJECT-TYPE
 SYNTAX  Integer32
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Current priority."
 ::= { astChanEntry 17 }

astChanAccountCode OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Account Code for billing."
 ::= { astChanEntry 18 }

astChanForwardTo OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
```

```
 DESCRIPTION
  "Where to forward to if asked to dial on this
  interface."
 ::= { astChanEntry 19 }

astChanUniqueId OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Unique Channel Identifier."
 ::= { astChanEntry 20 }

astChanCallGroup OBJECT-TYPE
 SYNTAX  Unsigned32
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Call Group."
 ::= { astChanEntry 21 }

astChanPickupGroup OBJECT-TYPE
 SYNTAX  Unsigned32
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Pickup Group."
 ::= { astChanEntry 22 }

astChanState OBJECT-TYPE
 SYNTAX  INTEGER {
  stateDown(0),
  stateReserved(1),
  stateOffHook(2),
  stateDialing(3),
  stateRing(4),
  stateRinging(5),
  stateUp(6),
  stateBusy(7),
  stateDialingOffHook(8),
  statePreRing(9)
 }
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Channel state."
```

```
 ::= { astChanEntry 23 }

astChanMuted OBJECT-TYPE
 SYNTAX   TruthValue
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Transmission of voice data has been muted."
 ::= { astChanEntry 24 }

astChanRings OBJECT-TYPE
 SYNTAX   Integer32
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Number of rings so far."
 ::= { astChanEntry 25 }

astChanCidDNID OBJECT-TYPE
 SYNTAX   DisplayString
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Dialled Number ID."
 ::= { astChanEntry 26 }

astChanCidNum OBJECT-TYPE
 SYNTAX   DisplayString
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Caller Number."
 ::= { astChanEntry 27 }

astChanCidName OBJECT-TYPE
 SYNTAX   DisplayString
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Caller Name."
 ::= { astChanEntry 28 }

astChanCidANI OBJECT-TYPE
 SYNTAX   DisplayString
 MAX-ACCESS read-only
 STATUS   current
```

```
  DESCRIPTION
   "ANI"
 ::= { astChanEntry 29 }


astChanCidRDNIS OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Redirected Dialled Number Service."
 ::= { astChanEntry 30 }


astChanCidPresentation OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Number Presentation/Screening."
 ::= { astChanEntry 31 }


astChanCidANI2 OBJECT-TYPE
 SYNTAX  Integer32
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "ANI 2 (info digit)."
 ::= { astChanEntry 32 }


astChanCidTON OBJECT-TYPE
 SYNTAX  Integer32
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Type of Number."
 ::= { astChanEntry 33 }


astChanCidTNS OBJECT-TYPE
 SYNTAX  Integer32
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Transit Network Select."
 ::= { astChanEntry 34 }


astChanAMAFlags OBJECT-TYPE
 SYNTAX  INTEGER {
```

```
   default(0),
   omit(1),
   billing(2),
   documentation(3)
  }
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "AMA Flags."
 ::= { astChanEntry 35 }

astChanADSI OBJECT-TYPE
 SYNTAX   INTEGER {
   unknown(0),
   available(1),
   unavailable(2),
   offHookOnly(3)
  }
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Whether or not ADSI is detected on CPE."
 ::= { astChanEntry 36 }

astChanToneZone OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
   "Indication zone to use for channel."
 ::= { astChanEntry 37 }

astChanHangupCause OBJECT-TYPE
 SYNTAX   INTEGER {
   notDefined(0),
   unregistered(3),
   normal(16),
   busy(17),
   noAnswer(19),
   congestion(34),
   failure(38),
   noSuchDriver(66)
  }
 MAX-ACCESS read-only
 STATUS   current
 DESCRIPTION
```

```
   "Why is the channel hung up."
 ::= { astChanEntry 38 }

astChanVariables OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Channel Variables defined for this channel."
 ::= { astChanEntry 39 }

astChanFlags OBJECT-TYPE
 SYNTAX  BITS {
  wantsJitter(0),
  deferDTMF(1),
  writeInterrupt(2),
  blocking(3),
  zombie(4),
  exception(5),
  musicOnHold(6),
  spying(7),
  nativeBridge(8),
  autoIncrementingLoop(9)
 }
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Flags set on this channel."
 ::= { astChanEntry 40 }

astChanTransferCap OBJECT-TYPE
 SYNTAX  INTEGER {
  speech(0),
  digital(8),
  restrictedDigital(9),
  audio3k(16),
  digitalWithTones(17),
  video(24)
 }
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
   "Transfer Capabilities for this channel."
 ::= { astChanEntry 41 }

astNumChanTypes OBJECT-TYPE
```

```
 SYNTAX  Integer32
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Number of channel types (technologies) supported."
 ::= { asteriskChannels 3 }

astChanTypeTable OBJECT-TYPE
 SYNTAX  SEQUENCE OF AstChanTypeEntry
 MAX-ACCESS not-accessible
 STATUS  current
 DESCRIPTION
  "Table with details of the supported channel types."
 ::= { asteriskChannels 4 }

astChanTypeEntry OBJECT-TYPE
 SYNTAX  AstChanTypeEntry
 MAX-ACCESS not-accessible
 STATUS  current
 DESCRIPTION
  "Information about a technology we support, including
  how many channels are currently using this technology."
 INDEX  { astChanTypeIndex }
 ::= { astChanTypeTable 1 }

AstChanTypeEntry ::= SEQUENCE {
 astChanTypeIndex Integer32,
 astChanTypeName  DisplayString,
 astChanTypeDesc  DisplayString,
 astChanTypeDeviceState Integer32,
 astChanTypeIndications Integer32,
 astChanTypeTransfer Integer32,
 astChanTypeChannels Gauge32
}

astChanTypeIndex OBJECT-TYPE
 SYNTAX  Integer32 (1 .. 2147483647)
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Index into the table of channel types."
 ::= { astChanTypeEntry 1 }

astChanTypeName OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
```

```
 STATUS  current
 DESCRIPTION
  "Unique name of the technology we are describing."
 ::= { astChanTypeEntry 2 }

astChanTypeDesc OBJECT-TYPE
 SYNTAX  DisplayString
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Description of the channel type (technology)."
 ::= { astChanTypeEntry 3 }

astChanTypeDeviceState OBJECT-TYPE
 SYNTAX  TruthValue
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Whether the current technology can hold device states."
 ::= { astChanTypeEntry 4 }

astChanTypeIndications OBJECT-TYPE
 SYNTAX  TruthValue
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Whether the current technology supports progress indication."
 ::= { astChanTypeEntry 5 }

astChanTypeTransfer OBJECT-TYPE
 SYNTAX  TruthValue
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Whether the current technology supports transfers, where
   Asterisk can get out from inbetween two bridged channels."
 ::= { astChanTypeEntry 6 }

astChanTypeChannels OBJECT-TYPE
 SYNTAX  Gauge32
 MAX-ACCESS read-only
 STATUS  current
 DESCRIPTION
  "Number of active channels using the current technology."
 ::= { astChanTypeEntry 7 }
```

```
astChanScalars OBJECT IDENTIFIER ::= { asteriskChannels 5 }

astNumChanBridge OBJECT-TYPE
        SYNTAX              Gauge32
        MAX-ACCESS          read-only
        STATUS              current
        DESCRIPTION
                "Number of channels currently in a bridged state."
```

```
              ::= { astChanScalars 1 }

  END
```

Digium MIB Definitions

```
DIGIUM-MIB DEFINITIONS ::= BEGIN

IMPORTS
 enterprises, MODULE-IDENTITY
  FROM SNMPv2-SMI;

digium MODULE-IDENTITY
 LAST-UPDATED "200806202000Z"
 ORGANIZATION "Digium, Inc."
 CONTACT-INFO
   "Mark Spencer
   Email: markster@digium.com"
 DESCRIPTION
   "The Digium private-enterprise MIB"
 REVISION "200806202000Z"
 DESCRIPTION
   "Corrected imports and missing revision for last update.
    Addresses bug 12905. - jeffg@opennms.org"
 REVISION "200602041900Z"
 DESCRIPTION
   "Initial revision."
 ::= { enterprises 22736 }

END
```

SIP Retransmissions

What is the problem with SIP retransmits?

Sometimes you get messages in the console like these:

```
retrans_pkt: Hanging up call XX77yy  - no reply to our critical
packet.
retrans_pkt: Cancelling retransmit of OPTIONs
```

The SIP protocol is based on requests and replies. Both sides send requests and wait for replies. Some of these requests are important. In a TCP/IP network many things can happen with IP packets. Firewalls, NAT devices, Session Border Controllers and SIP Proxys are in the signalling path and they will affect the call.

SIP Call setup - INVITE-200 OK - ACK

To set up a SIP call, there's an INVITE transaction. The SIP software that initiates the call sends an INVITE, then wait to get a reply. When a reply arrives, the caller sends an ACK. This is a three-way handshake that is in place since a phone can ring for a very long time and the protocol needs to make sure that all devices are still on line when call setup is done and media starts to flow.

- The first reply we're waiting for is often a "100 trying". This message means that some type of SIP server has received our request and makes sure that we will get a reply. It could be the other endpoint, but it could also be a SIP proxy or SBC that handles the request on our behalf.

- After that, you often see a response in the 18x class, like "180 ringing" or "183 Session Progress". This typically means that our request has reached at least one endpoint and something is alerting the other end that there's a call coming in.

- Finally, the other side answers and we get a positive reply, "200 OK". This is a positive answer. In that message, we get an address that goes directly to the device that answers. Remember, there could be multiple phones ringing. The address is specified by the Contact: header.

- To confirm that we can reach the phone that answered our call, we now send an ACK to the Contact: address. If this ACK doesn't reach the phone, the call fails. If we can't send an ACK, we can't send anything else, not even a proper hangup. Call signaling will simply fail for the rest of the call and there's no point in keeping it alive.

- If we get an error response to our INVITE, like "Busy" or "Rejected", we send the ACK to the same address as we sent the INVITE, to confirm that we got the response.

In order to make sure that the whole call setup sequence works and that we have a call, a SIP client retransmits messages if there's too much delay between request and expected response. We retransmit a number of times while waiting for the first response. We retransmit the answer to an incoming INVITE while waiting for an ACK. If we get multiple answers, we send an ACK to each of them.

If we don't get the ACK or don't get an answer to our INVITE, even after retransmissions, we will hangup the call with the first error message you see above.

Other SIP requests

Other SIP requests are only based on request - reply. There's no ACK, no three-way handshake. In Asterisk we mark some of these as CRITICAL - they need to go through for the call to work as expected. Some are non-critical, we don't really care what happens with them, the call will go on happily regardless.

The qualification process - OPTIONS

If you turn on qualify= in sip.conf for a device, Asterisk will send an OPTIONS request every minute to the device and check if it replies. Each OPTIONS request is retransmitted a number of times (to handle packet loss) and if we get no reply, the device is considered unreachable. From that moment, we will send a new OPTIONS request (with retransmits) every tenth second.

Why does this happen?

For some reason signalling doesn't work as expected between your Asterisk server and the other device. There could be many reasons why this happens.

- A NAT device in the signalling path. A misconfigured NAT device is in the signalling path and stops SIP messages.
- A firewall that blocks messages or reroutes them wrongly in an attempt to assist in a too clever way.
- A SIP middlebox (SBC) that rewrites contact: headers so that we can't reach the other side with our reply or the ACK.
- A badly configured SIP proxy that forgets to add record-route headers to make sure that signalling works.
- Packet loss. IP and UDP are unreliable transports. If you loose too many packets the retransmits doesn't help and communication is impossible. If this happens with signaling, media would be unusable anyway.

What can I do?

Turn on SIP debug, try to understand the signalling that happens and see if you're missing the reply to the INVITE or if the ACK gets lost. When you know what happens, you've taken the first step to track down the problem. See the list above and investigate your network.

For NAT and Firewall problems, there are many documents to help you. Start with reading sip.conf.sample that is part of your Asterisk distribution.

The SIP signalling standard, including retransmissions and timers for these, is well documented in the IETF RFC 3261.

Good luck sorting out your SIP issues!

/Olle E. Johansson

– oej (at) edvina.net, Sweden, 2008-07-22
– http://www.voip-forum.com

SIP TLS Transport

Asterisk SIP/TLS Transport

When using TLS the client will typically check the validity of the certificate chain. So that means you either need a certificate that is signed by one of the larger CAs, or if you use a self signed certificate you must install a copy of your CA certificate on the client.

So far this code has been tested with:

- Asterisk as client and server (TLS and TCP)
- Polycom Soundpoint IP Phones (TLS and TCP) - Polycom phones require that the host (ip or hostname) that is configured match the 'common name' in the certificate
- Minisip Softphone (TLS and TCP)
- Cisco IOS Gateways (TCP only)
- SNOM 360 (TLS only)
- Zoiper Biz Softphone (TLS and TCP)

sip.conf options

- tlsenable=[yes] - Enable TLS server, default is no
- tlsbindaddr=<ip address> - Specify IP address to bind TLS server to, default is 0.0.0.0
- tlscertfile=</path/to/certificate> - The server's certificate file. Should include the key and certificate. This is mandatory if you're going to run a TLS server.
- tlscafile=</path/to/certificate> - If the server your connecting to uses a self signed certificate you should have their certificate installed here so the code can verify the authenticity of their certificate.
- tlscadir=</path/to/ca/dir> - A directory full of CA certificates. The files must be named with the CA subject name hash value. (see man SSL_CTX_load_verify_locations for more info)
- tlsdontverifyserver=[yes] - If set to yes, don't verify the servers certificate when acting as a client. If you don't have the server's CA certificate you can set this and it will connect without requiring tlscafile to be set. Default is no.
- tlscipher=<SSL cipher string> - A string specifying which SSL ciphers to use or not use. A list of valid SSL cipher strings can be found at http://www.openssl.org/docs/apps/ciphers.html#CIPHER_STRINGS

Sample config

Here are the relevant bits of config for setting up TLS between 2 Asterisk servers. With server_a registering to server_b

On server_a:

```
[general]
tlsenable=yes
tlscertfile=/etc/asterisk/asterisk.pem
tlscafile=/etc/ssl/ca.pem  ; This is the CA file used to generate
both certificates
register => tls://100:test@192.168.0.100:5061

[101]
type=friend
context=internal
host=192.168.0.100 ; The host should be either IP or hostname and
should
                   ; match the 'common name' field in the servers
certificate
secret=test
dtmfmode=rfc2833
disallow=all
allow=ulaw
transport=tls
port=5061
```

On server_b:

```
[general]
tlsenable=yes
tlscertfile=/etc/asterisk/asterisk.pem

[100]
type=friend
context=internal
host=dynamic
secret=test
dtmfmode=rfc2833
disallow=all
allow=ulaw
;You can specify transport= and port=5061 for TLS, but its not
necessary in
;the server configuration, any type of SIP transport will work
;transport=tls
;port=5061
```

Speech Recognition API

The Asterisk Speech Recognition API

The generic speech recognition engine is implemented in the res_speech.so module. This module connects through the API to speech recognition software, that is not included in the module.

To use the API, you must load the res_speech.so module before any connectors. For your convenience, there is a preload line commented out in the modules.conf sample file.

Dialplan Applications:

The dialplan API is based around a single speech utilities application file, which exports many applications to be used for speech recognition. These include an application to prepare for speech recognition, activate a grammar, and play back a sound file while waiting for the person to speak. Using a combination of these applications you can easily make a dialplan use speech recognition without worrying about what speech recognition engine is being used.

SpeechCreate(Engine Name)

This application creates information to be used by all the other applications. It must be called before doing any speech recognition activities such as activating a grammar. It takes the engine name to use as the argument, if not specified the default engine will be used.

If an error occurs are you are not able to create an object, the variable ERROR will be set to 1. You can then exit your speech recognition specific context and play back an error message, or resort to a DTMF based IVR.

SpeechLoadGrammar(Grammar Name|Path)

Loads grammar locally on a channel. Note that the grammar is only available as long as the channel exists, and you must call SpeechUnloadGrammar before all is done or you may cause a memory leak. First argument is the grammar name that it will be loaded as and second argument is the path to the grammar.

SpeechUnloadGrammar(Grammar Name)

Unloads a locally loaded grammar and frees any memory used by it. The only argument is the name of the grammar to unload.

SpeechActivateGrammar(Grammar Name)

This activates the specified grammar to be recognized by the engine. A grammar tells the speech recognition engine what to recognize, and how to portray it back to you in the dialplan. The grammar name is the only argument to this application.

SpeechStart()

Tell the speech recognition engine that it should start trying to get results from audio being fed to it. This has no arguments.

SpeechBackground(Sound File|Timeout)

This application plays a sound file and waits for the person to speak. Once they start speaking playback of the file stops, and silence is heard. Once they stop talking the processing sound is played to indicate the speech recognition engine is working. Note it is possible to have more then one result. The first argument is the sound file and the second is the timeout. Note the timeout will only start once the sound file has stopped playing.

SpeechDeactivateGrammar(Grammar Name)

This deactivates the specified grammar so that it is no longer recognized. The only argument is the grammar name to deactivate.

- SpeechProcessingSound(Sound File)*

This changes the processing sound that SpeechBackground plays back when the speech recognition engine is processing and working to get results. It takes the sound file as the only argument.

- SpeechDestroy()*

This destroys the information used by all the other speech recognition applications. If you call this application but end up wanting to recognize more speech, you must call SpeechCreate again before calling any other application. It takes no arguments.

Getting Result Information:

The speech recognition utilities module exports several dialplan functions that you can use to

examine results.

*${SPEECH(status)} - Returns 1 if SpeechCreate has been called. This uses the same check that applications do to see if a
speech object is setup. If it returns 0 then you know you can not use other speech applications.

*${SPEECH(spoke)} - Returns 1 if the speaker spoke something, or 0 if they were silent.

*${SPEECH(results)} - Returns the number of results that are available.

*${\SPEECH_SCORE(result number)} - Returns the score of a result.

*${SPEECH_TEXT(result number)} - Returns the recognized text of a result.

*${SPEECH_GRAMMAR(result number)} - Returns the matched grammar of the result.

*SPEECH_ENGINE(name)=value - Sets a speech engine specific attribute.

Dialplan Flow:

1. Create a speech recognition object using SpeechCreate()
2. Activate your grammars using SpeechActivateGrammar(Grammar Name)
3. Call SpeechStart() to indicate you are going to do speech recognition immediately
4. Play back your audio and wait for recognition using SpeechBackground(Sound File|Timeout)
5. Check the results and do things based on them
6. Deactivate your grammars using SpeechDeactivateGrammar(Grammar Name)
7. Destroy your speech recognition object using SpeechDestroy()

Dialplan Examples:

This is pretty cheeky in that it does not confirmation of results. As well the way the grammar is written it returns the person's extension instead of their name so we can just do a Goto based on the result text.

```
- Grammar: company-directory.gram

#ABNF 1.0;
language en-US;
mode voice;
tag-format <lumenvox/1.0>;
root $company_directory;

$josh = ((Joshua | Josh) [Colp]):"6066";
$mark = (Mark [Spencer] | Markster):"4569";
$kevin = (Kevin [Fleming]):"2567";

$company_directory = ($josh | $mark | $kevin) { $ = $$ };
```

Dialplan logic

```
[dial-by-name]
exten => s,1,SpeechCreate()
exten => s,2,SpeechActivateGrammar(company-directory)
exten => s,3,SpeechStart()
exten => s,4,SpeechBackground(who-would-you-like-to-dial)
exten => s,5,SpeechDeactivateGrammar(company-directory)
exten => s,6,Goto(internal-extensions-${SPEECH_TEXT(0)})
```

Useful Dialplan Tidbits

A simple macro that can be used for confirm of a result. Requires some sound files. ARG1 is equal to the file to play back after "I heard..." is played.

```
[macro-speech-confirm]
exten => s,1,SpeechActivateGrammar(yes_no)
exten => s,2,Set(OLDTEXT0=${SPEECH_TEXT(0)})
exten => s,3,Playback(heard)
exten => s,4,Playback(${ARG1})
exten => s,5,SpeechStart()
exten => s,6,SpeechBackground(correct)
exten => s,7,Set(CONFIRM=${SPEECH_TEXT(0)})
exten => s,8,GotoIf($["${SPEECH_TEXT(0)}" = "1"]?9:10)
exten => s,9,Set(CONFIRM=yes)
exten => s,10,Set(CONFIRMED=${OLDTEXT0})
exten => s,11,SpeechDeactivateGrammar(yes_no)
```

The Asterisk Speech Recognition C API

The module res_speech.so exports a C based API that any developer can use to speech recognize enable their application. The API gives greater control, but requires the developer to do more on their end in comparison to the dialplan speech utilities.

For all API calls that return an integer value, a non-zero value indicates an error has occurred.

Creating a speech structure

```
struct ast_speech *ast_speech_new(char *engine_name, int format)

struct ast_speech *speech = ast_speech_new(NULL,
AST_FORMAT_SLINEAR);
```

This will create a new speech structure that will be returned to you. The speech recognition engine name is optional and if NULL the default one will be used. As well for now format should always be AST_FORMAT_SLINEAR.

Activating a grammar

```
int ast_speech_grammar_activate(struct ast_speech *speech, char
*grammar_name)

res = ast_speech_grammar_activate(speech, "yes_no");
```

This activates the specified grammar on the speech structure passed to it.

Start recognizing audio

```
void ast_speech_start(struct ast_speech *speech)

ast_speech_start(speech);
```

This essentially tells the speech recognition engine that you will be feeding audio to it from then on. It MUST be called every time before you start feeding audio to the speech structure.

Send audio to be recognized

```
int ast_speech_write(struct ast_speech *speech, void *data, int len)

res = ast_speech_write(speech, fr->data, fr->datalen);
```

This writes audio to the speech structure that will then be recognized. It must be written signed linear only at this time. In the future other formats may be supported.

Checking for results

The way the generic speech recognition API is written is that the speech structure will undergo state changes to indicate progress of recognition. The states are outlined below:

- AST_SPEECH_STATE_NOT_READY - The speech structure is not ready to accept audio
- AST_SPEECH_STATE_READY - You may write audio to the speech structure
- AST_SPEECH_STATE_WAIT - No more audio should be written, and results will be available soon.
- AST_SPEECH_STATE_DONE - Results are available and the speech structure can only be used again by calling ast_speech_start

It is up to you to monitor these states. Current state is available via a variable on the speech structure. (state)

Knowing when to stop playback

If you are playing back a sound file to the user and you want to know when to stop play back because the
individual started talking use the following.

```
ast_test_flag(speech, AST_SPEECH_QUIET) - This will return a
positive value when the person has started talking.
```

Getting results

```
struct ast_speech_result *ast_speech_results_get(struct ast_speech
*speech)

 struct ast_speech_result *results = ast_speech_results_get(speech);
```

This will return a linked list of result structures. A result structure looks like the following:

```
struct ast_speech_result {
        char *text;    /*!< Recognized text */
        int score;    /*!< Result score */
        char *grammar;    /*!< Matched grammar */
        struct ast_speech_result *next; /*!< List information */
 };
```

Freeing a set of results

```
int ast_speech_results_free(struct ast_speech_result *result)

 res = ast_speech_results_free(results);
```

This will free all results on a linked list. Results MAY NOT be used as the memory will have been freed.

Deactivating a grammar

```
int ast_speech_grammar_deactivate(struct ast_speech *speech, char
*grammar_name)

 res = ast_speech_grammar_deactivate(speech, "yes_no");
```

This deactivates the specified grammar on the speech structure.

Destroying a speech structure

```
int ast_speech_destroy(struct ast_speech *speech)

 res = ast_speech_destroy(speech);
```

This will free all associated memory with the speech structure and destroy it with the speech recognition engine.

Loading a grammar on a speech structure

```
int ast_speech_grammar_load(struct ast_speech *speech, char
*grammar_name, char *grammar)

 res = ast_speech_grammar_load(speech, "builtin:yes_no", "yes_no");
```

Unloading a grammar on a speech structure

If you load a grammar on a speech structure it is preferred that you unload it as well, or you may cause a memory leak. Don't say I didn't warn you.

```
int ast_speech_grammar_unload(struct ast_speech *speech, char
*grammar_name)

 res = ast_speech_grammar_unload(speech, "yes_no");
```

This unloads the specified grammar from the speech structure.

SQLite Tables

```
/*
 * res_config_sqlite - SQLite 2 support for Asterisk
 *
 * This module can be used as a static/RealTime configuration
module, and a CDR
 * handler.  See the Doxygen documentation for a detailed
description of the
 * module, and the configs/ directory for the sample configuration
file.
 */

/*
 * Tables for res_config_sqlite.so.
 */
```

```
/*
 * RealTime static table.
 */
CREATE TABLE ast_config (
 id   INTEGER,
 cat_metric INT(11)  NOT NULL DEFAULT 0,
 var_metric INT(11)  NOT NULL DEFAULT 0,
 commented TINYINT(1) NOT NULL DEFAULT 0,
 filename VARCHAR(128) NOT NULL DEFAULT '',
 category VARCHAR(128) NOT NULL DEFAULT 'default',
 var_name VARCHAR(128) NOT NULL DEFAULT '',
 var_val  TEXT  NOT NULL DEFAULT '',
 PRIMARY KEY (id)
);

CREATE INDEX ast_config__idx__cat_metric  ON ast_config(cat_metric);
CREATE INDEX ast_config__idx__var_metric  ON ast_config(var_metric);
CREATE INDEX ast_config__idx__filename_commented ON
ast_config(filename, commented);

/*
 * CDR table (this table is automatically created if non existent).
 */
CREATE TABLE ast_cdr (
 id   INTEGER,
 clid  VARCHAR(80) NOT NULL DEFAULT '',
 src  VARCHAR(80) NOT NULL DEFAULT '',
 dst  VARCHAR(80) NOT NULL DEFAULT '',
 dcontext VARCHAR(80) NOT NULL DEFAULT '',
 channel  VARCHAR(80) NOT NULL DEFAULT '',
 dstchannel VARCHAR(80) NOT NULL DEFAULT '',
 lastapp  VARCHAR(80) NOT NULL DEFAULT '',
 lastdata VARCHAR(80) NOT NULL DEFAULT '',
 start  DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
 answer  DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
 end  DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
 duration INT(11)  NOT NULL DEFAULT 0,
 billsec  INT(11)  NOT NULL DEFAULT 0,
 disposition VARCHAR(45) NOT NULL DEFAULT '',
 amaflags INT(11)  NOT NULL DEFAULT 0,
 accountcode VARCHAR(20) NOT NULL DEFAULT '',
 uniqueid VARCHAR(32) NOT NULL DEFAULT '',
 userfield VARCHAR(255) NOT NULL DEFAULT '',
 PRIMARY KEY (id)
);
```

```
/*
 * SIP RealTime table.
 */
CREATE TABLE ast_sip (
 id   INTEGER,
 commented TINYINT(1) NOT NULL DEFAULT 0,
 name  VARCHAR(80) NOT NULL DEFAULT '',
 host  VARCHAR(31) NOT NULL DEFAULT '',
 nat  VARCHAR(5) NOT NULL DEFAULT 'no',
 type  VARCHAR(6) NOT NULL DEFAULT 'friend',
 accountcode VARCHAR(20)   DEFAULT NULL,
 amaflags VARCHAR(13)   DEFAULT NULL,
 callgroup VARCHAR(10)   DEFAULT NULL,
 callerid VARCHAR(80)   DEFAULT NULL,
 cancallforward CHAR(3)    DEFAULT 'yes',
 directmedia CHAR(3)    DEFAULT 'yes',
 context  VARCHAR(80)   DEFAULT NULL,
 defaultip VARCHAR(15)   DEFAULT NULL,
 dtmfmode VARCHAR(7)   DEFAULT NULL,
 fromuser VARCHAR(80)   DEFAULT NULL,
 fromdomain VARCHAR(80)   DEFAULT NULL,
 insecure VARCHAR(4)   DEFAULT NULL,
 language CHAR(2)   DEFAULT NULL,
 mailbox  VARCHAR(50)   DEFAULT NULL,
 md5secret VARCHAR(80)   DEFAULT NULL,
 deny  VARCHAR(95)   DEFAULT NULL,
 permit  VARCHAR(95)   DEFAULT NULL,
 mask  VARCHAR(95)   DEFAULT NULL,
 musiconhold VARCHAR(100)   DEFAULT NULL,
 pickupgroup VARCHAR(10)   DEFAULT NULL,
 qualify  CHAR(3)   DEFAULT NULL,
 regexten VARCHAR(80)   DEFAULT NULL,
 restrictcid CHAR(3)    DEFAULT NULL,
 rtptimeout CHAR(3)    DEFAULT NULL,
 rtpholdtimeout CHAR(3)    DEFAULT NULL,
 secret  VARCHAR(80)   DEFAULT NULL,
 setvar  VARCHAR(100)   DEFAULT NULL,
 disallow VARCHAR(100)   DEFAULT 'all',
 allow  VARCHAR(100)   DEFAULT 'g729,ilbc,gsm,ulaw,alaw',
 fullcontact VARCHAR(80) NOT NULL DEFAULT '',
 ipaddr  VARCHAR(15) NOT NULL DEFAULT '',
 port  INT(11)  NOT NULL DEFAULT 0,
 regserver VARCHAR(100)   DEFAULT NULL,
 regseconds INT(11)  NOT NULL DEFAULT 0,
 username VARCHAR(80) NOT NULL DEFAULT '',
 PRIMARY KEY (id)
```

```
 UNIQUE  (name)
);

CREATE INDEX ast_sip__idx__commented ON ast_sip(commented);

/*
 * Dialplan RealTime table.
 */
CREATE TABLE ast_exten (
 id  INTEGER,
 commented TINYINT(1) NOT NULL DEFAULT 0,
 context  VARCHAR(80) NOT NULL DEFAULT '',
 exten  VARCHAR(40) NOT NULL DEFAULT '',
 priority INT(11)  NOT NULL DEFAULT 0,
 app  VARCHAR(128) NOT NULL DEFAULT '',
 appdata  VARCHAR(128) NOT NULL DEFAULT '',
 PRIMARY KEY (id)
);

CREATE INDEX ast_exten__idx__commented   ON ast_exten(commented);
```

```
CREATE INDEX ast_exten__idx__context_exten_priority ON
ast_exten(context, exten, priority);
```

Storing Voicemail in PostgreSQL via ODBC

How to get ODBC storage with PostgreSQL working with Voicemail

*Install PostgreSQL, PostgreSQL-devel, unixODBC, and unixODBC-devel, and
PostgreSQL-ODBC. Make sure PostgreSQL is running and listening on a TCP socket.

*Log into your server as root, and then type:

```
[root@localhost ~]# su - postgres
```

This will log you into the system as the "postgres" user, so that you can create a new role and
database within the PostgreSQL database system. At the new prompt, type:

```
$ createuser -s -D -R -l -P -e asterisk
Enter password for new role:
Enter it again:
```

Obviously you should enter a password when prompted. This creates the database role (or user).

Next we need to create the asterisk database. Type:

```
$ createdb -O asterisk -e asterisk
```

This creates the database and sets the owner of the database to the asterisk role.

Next, make sure that you are using md5 authentication for the database user. The line in my
/var/lib/pgsql/data/pg_hba.conf looks like:

```
# "local" is for Unix domain socket connections only
local    asterisk     asterisk                            md5
local    all          all                                 ident sameuser
# IPv4 local connections:
host     all          all         127.0.0.1/32            md5
```

As soon as you're done editing that file, log out as the postgres user.

- Make sure you have the PostgreSQL odbc driver setup in /etc/odbcinst.ini. Mine looks like:

```
[PostgreSQL]
Description      = ODBC for PostgreSQL
Driver           = /usr/lib/libodbcpsql.so
Setup            = /usr/lib/libodbcpsqlS.so
FileUsage        = 1
```

You can confirm that unixODBC is seeing the driver by typing:

```
[jsmith2@localhost tmp]$ odbcinst -q -d
[PostgreSQL]
```

- Setup a DSN in /etc/odbc.ini, pointing at the PostgreSQL database and driver. Mine looks like:

```
[testing]
Description          = ODBC Testing
Driver               = PostgreSQL
Trace                = No
TraceFile            = sql.log
Database             = asterisk
Servername           = 127.0.0.1
UserName             = asterisk
Password             = supersecret
Port                 = 5432
ReadOnly             = No
RowVersioning        = No
ShowSystemTables     = No
ShowOidColumn        = No
FakeOidIndex         = No
ConnSettings         =
```

You can confirm that unixODBC sees your DSN by typing:

```
[jsmith2@localhost tmp]$ odbcinst -q -s
[testing]
```

- Test your database connectivity through ODBC. If this doesn't work, something is wrong with your ODBC setup.

```
[jsmith2@localhost tmp]$ echo "select 1" | isql -v testing
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL> +-----------+
| ?column?  |
+-----------+
| 1         |
+-----------+
SQLRowCount returns 1
1 rows fetched
```

If your ODBC connectivity to PostgreSQL isn't working, you'll see an error message instead, like this:

```
[jsmith2@localhost tmp]$ echo "select 1" | isql -v testing
[S1000][unixODBC]Could not connect to the server;
Could not connect to remote socket.
[ISQL]ERROR: Could not SQLConnect
bash: echo: write error: Broken pipe
```

- Compile Asterisk with support for ODBC voicemail. Go to your Asterisk source directory and run `make menuselect`. Under "Voicemail Build Options", enable "ODBC_STORAGE". See doc/README.odbcstorage for more information

Recompile Asterisk and install the new version.

- Once you've recompiled and re-installed Asterisk, check to make sure res_odbc.so has been compiled.

```
localhost*CLI> show modules like res_odbc.so
Module                          Description
Use Count
res_odbc.so                     ODBC Resource
0
1 modules loaded
```

- Now it's time to get Asterisk configured. First, we need to tell Asterisk about our ODBC setup. Open /etc/asterisk/res_odbc.conf and add the following:

```
[postgres]
enabled => yes
dsn => testing
pre-connect => yes
```

- At the Asterisk CLI, unload and then load the res_odbc.so module. (You could restart Asterisk as well, but this way makes it easier to tell what's happening.) Notice how it says it's connected to "postgres", which is our ODBC connection as defined in res_odbc.conf, which points to the "testing" DSN in ODBC.

```
localhost*CLI> unload res_odbc.so
Jan  2 21:19:36 WARNING[8130]: res_odbc.c:498 odbc_obj_disconnect:
res_odbc: disconnected 0 from postgres [testing]
Jan  2 21:19:36 NOTICE[8130]: res_odbc.c:589 unload_module: res_odbc
unloaded.
localhost*CLI> load res_odbc.so
 Loaded /usr/lib/asterisk/modules/res_odbc.so => (ODBC Resource)
   == Parsing '/etc/asterisk/res_odbc.conf': Found
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:266 load_odbc_config:
Adding ENV var: INFORMIXSERVER=my_special_database
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:266 load_odbc_config:
Adding ENV var: INFORMIXDIR=/opt/informix
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:295 load_odbc_config:
registered database handle 'postgres' dsn->[testing]
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:555 odbc_obj_connect:
Connecting postgres
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:570 odbc_obj_connect:
res_odbc: Connected to postgres [testing]
Jan  2 21:19:40 NOTICE[8130]: res_odbc.c:600 load_module: res_odbc
loaded.
```

You can also check the status of your ODBC connection at any time from the Asterisk CLI:

```
localhost*CLI> odbc show
Name: postgres
DSN: testing
Connected: yes
```

- Now we can setup our voicemail table in PostgreSQL. Log into PostgreSQL and type (or copy and paste) the following:

```
--
-- First, let's create our large object type, called "lo"
--
CREATE FUNCTION loin (cstring) RETURNS lo AS 'oidin' LANGUAGE
internal IMMUTABLE STRICT;
```

```sql
CREATE FUNCTION loout (lo) RETURNS cstring AS 'oidout' LANGUAGE
internal IMMUTABLE STRICT;
CREATE FUNCTION lorecv (internal) RETURNS lo AS 'oidrecv' LANGUAGE
internal IMMUTABLE STRICT;
CREATE FUNCTION losend (lo) RETURNS bytea AS 'oidrecv' LANGUAGE
internal IMMUTABLE STRICT;

CREATE TYPE lo ( INPUT = loin, OUTPUT = loout, RECEIVE = lorecv,
SEND = losend, INTERNALLENGTH = 4, PASSEDBYVALUE );
CREATE CAST (lo AS oid) WITHOUT FUNCTION AS IMPLICIT;
CREATE CAST (oid AS lo) WITHOUT FUNCTION AS IMPLICIT;


--
-- If we're not already using plpgsql, then let's use it!
--
CREATE TRUSTED LANGUAGE plpgsql;


--
-- Next, let's create a trigger to cleanup the large object table
-- whenever we update or delete a row from the voicemessages table
--

CREATE FUNCTION vm_lo_cleanup() RETURNS "trigger"
    AS $$
    declare
      msgcount INTEGER;
    begin
      --    raise notice 'Starting lo_cleanup function for large
object with oid %',old.recording;
      -- If it is an update action but the BLOB (lo) field was not
changed, dont do anything
      if (TG_OP = 'UPDATE') then
        if ((old.recording = new.recording) or (old.recording is
NULL)) then
          raise notice 'Not cleaning up the large object table, as
recording has not changed';
          return new;
        end if;
      end if;
      if (old.recording IS NOT NULL) then
        SELECT INTO msgcount COUNT(*) AS COUNT FROM voicemessages
WHERE recording = old.recording;
        if (msgcount > 0) then
          raise notice 'Not deleting record from the large object
table, as object is still referenced';
          return new;
```

```
          else
            perform lo_unlink(old.recording);
            if found then
              raise notice 'Cleaning up the large object table';
              return new;
            else
              raise exception 'Failed to cleanup the large object
table';
              return old;
            end if;
          end if;
        else
          raise notice 'No need to cleanup the large object table, no
recording on old row';
          return new;
        end if;
    end$$
    LANGUAGE plpgsql;

--
-- Now, let's create our voicemessages table
-- This is what holds the voicemail from Asterisk
--

CREATE TABLE voicemessages
(
  uniqueid serial PRIMARY KEY,
  msgnum int4,
  dir varchar(80),
  context varchar(80),
  macrocontext varchar(80),
  callerid varchar(40),
  origtime varchar(40),
  duration varchar(20),
  flag varchar(8),
  mailboxuser varchar(80),
  mailboxcontext varchar(80),
  recording lo,
  label varchar(30),
  "read" bool DEFAULT false
);

--
-- Let's not forget to make the voicemessages table use the trigger
--
```

```
CREATE TRIGGER vm_cleanup AFTER DELETE OR UPDATE ON voicemessages
FOR EACH ROW EXECUTE PROCEDURE vm_lo_cleanup();
```

- Just as a sanity check, make sure you check the voicemessages table via the isql utility.

```
[jsmith2@localhost ODBC]$ echo "SELECT uniqueid, msgnum, dir,
duration FROM voicemessages WHERE msgnum = 1" | isql testing
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL>
+-----------+-----------+-------------------------------------
uniqueid   | msgnum    | dir
| duration              |
+-----------+-----------+-------------------------------------
returns 0
```

- Now we can finally configure voicemail in Asterisk to use our database. Open /etc/asterisk/voicemail.conf, and look in the [general] section. I've changed the format to gsm (as I can't seem to get WAV or wav working), and specify both the odbc connection and database table to use.

```
[general]
; Default formats for writing Voicemail
;format=g723sf|wav49|wav
format=gsm
odbcstorage=postgres
odbctable=voicemessages
```

You'll also want to create a new voicemail context called "odbctest" to do some testing, and create a sample mailbox inside that context. Add the following to the very bottom of voicemail.conf:

```
[odbctest]
101 => 5555,Example Mailbox
```

- Once you've updated voicemail.conf, let's make the changes take effect:

```
localhost*CLI> unload app_voicemail.so
  == Unregistered application 'VoiceMail'
  == Unregistered application 'VoiceMailMain'
  == Unregistered application 'MailboxExists'
  == Unregistered application 'VMAuthenticate'
localhost*CLI> load app_voicemail.so
 Loaded /usr/lib/asterisk/modules/app_voicemail.so => (Comedian Mail
(Voicemail System))
  == Registered application 'VoiceMail'
  == Registered application 'VoiceMailMain'
  == Registered application 'MailboxExists'
  == Registered application 'VMAuthenticate'
  == Parsing '/etc/asterisk/voicemail.conf': Found
```

You can check to make sure your new mailbox exists by typing:

```
localhost*CLI> show voicemail users for odbctest
Context     Mbox  User                            Zone      NewMsg
odbctest    101   Example Mailbox                             0
```

- Now, let's add a new context called "odbc" to extensions.conf. We'll use these extensions to do some testing:

```
[odbc]
exten => 100,1,Voicemail(101@odbctest)
exten => 200,1,VoicemailMain(101@odbctest)
```

- Next, we need to point a phone at the odbc context. In my case, I've got a SIP phone called "linksys" that is registering to Asterisk, so I'm
  setting its context to the [odbc] context we created in the previous step. The relevant
  section of my sip.conf file looks like:

```
[linksys]
type=friend
secret=verysecret
disallow=all
allow=ulaw
allow=gsm
context=odbc
host=dynamic
qualify=yes
```

I can check to see that my linksys phone is registered with Asterisk correctly:

```
localhost*CLI> sip show peers like linksys
Name/username               Host            Dyn Nat ACL Port
Status
linksys/linksys             192.168.0.103    D           5060      OK
(9 ms)
1 sip peers [1 online , 0 offline]
```

- At last, we're finally ready to leave a voicemail message and have it stored in our database! (Who'd have guessed it would be this much trouble?!?) Pick up the phone, dial extension 100, and leave yourself a voicemail message.
  In my case, this is what appeared on the Asterisk CLI:

```
localhost*CLI>
    -- Executing VoiceMail("SIP/linksys-10228cac", "101@odbctest")
in new stack
    -- Playing 'vm-intro' (language 'en')
    -- Playing 'beep' (language 'en')
    -- Recording the message
    -- x=0, open writing:
/var/spool/asterisk/voicemail/odbctest/101/tmp/dlZunm format: gsm,
0x101f6534
    -- User ended message by pressing #
    -- Playing 'auth-thankyou' (language 'en')
  == Parsing
'/var/spool/asterisk/voicemail/odbctest/101/INBOX/msg0000.txt':
Found
```

Now, we can check the database and make sure the record actually made it into PostgreSQL, from within the psql utility.

```
[jsmith2@localhost ~]$ psql
Password:
Welcome to psql 8.1.4, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

asterisk=# SELECT * FROM voicemessages;
 uniqueid | msgnum |                           dir
| context | macrocontext |        callerid        | origtime |
duration | mailboxuser | mailboxcontext | recording | label | read |
sip_id | pabx_id | iax_id
----------+--------+--------------------------------------------------------
26 |       0 | /var/spool/asterisk/voicemail/odbctest/101/INBOX |
odbc     |                | "linksys" <linksys> | 1167794179 | 7
| 101          | odbctest       | 16599     |       | f     |      |
|
(1 row)
```

Did you notice the the recording column is just a number? When a recording gets stuck in the database, the audio isn't actually stored in the voicemessages table. It's stored in a system table called the large object table. We can look in the large object table and verify that the object actually exists there:

```
asterisk=# \lo_list
    Large objects
  ID   | Description
-------+-------------
 16599 |
(1 row)
```

In my case, the OID is 16599. Your OID will almost surely be different. Just make sure the OID number in the recording column in the voicemessages table corresponds with a record in the large object table. (The trigger we added to
our voicemessages table was designed to make sure this is always the case.)

We can also pull a copy of the voicemail message back out of the database and write it to a file, to help us as we debug things:

```
asterisk=# \lo_export 16599 /tmp/odcb-16599.gsm
lo_export
```

We can even listen to the file from the Linux command line:

```
[jsmith2@localhost tmp]$ play /tmp/odcb-16599.gsm

Input Filename : /tmp/odcb-16599.gsm
Sample Size    : 8-bits
Sample Encoding: gsm
Channels       : 1
Sample Rate    : 8000

Time: 00:06.22 [00:00.00] of 00:00.00 (  0.0%) Output Buffer:
298.36K

Done.
```

- Last but not least, we can pull the voicemail message back out of the database by dialing extension 200 and entering "5555" at the password prompt. You should see something like this on the Asterisk CLI:

```
localhost*CLI>
    -- Executing VoiceMailMain("SIP/linksys-10228cac",
"101@odbctest") in new stack
    -- Playing 'vm-password' (language 'en')
    -- Playing 'vm-youhave' (language 'en')
    -- Playing 'digits/1' (language 'en')
    -- Playing 'vm-INBOX' (language 'en')
    -- Playing 'vm-message' (language 'en')
    -- Playing 'vm-onefor' (language 'en')
    -- Playing 'vm-INBOX' (language 'en')
    -- Playing 'vm-messages' (language 'en')
    -- Playing 'vm-opts' (language 'en')
    -- Playing 'vm-first' (language 'en')
    -- Playing 'vm-message' (language 'en')
  == Parsing
'/var/spool/asterisk/voicemail/odbctest/101/INBOX/msg0000.txt':
Found
    -- Playing 'vm-received' (language 'en')
    -- Playing 'digits/at' (language 'en')
    -- Playing 'digits/10' (language 'en')
    -- Playing 'digits/16' (language 'en')
    -- Playing 'digits/p-m' (language 'en')
    -- Playing
'/var/spool/asterisk/voicemail/odbctest/101/INBOX/msg0000' (language
'en')
    -- Playing 'vm-advopts' (language 'en')
    -- Playing 'vm-repeat' (language 'en')
    -- Playing 'vm-delete' (language 'en')
    -- Playing 'vm-toforward' (language 'en')
    -- Playing 'vm-savemessage' (language 'en')
    -- Playing 'vm-helpexit' (language 'en')
    -- Playing 'vm-goodbye' (language 'en')
```

That's it!

Jared Smith
2 Jan 2006
(updated 11 Mar 2007)

Timing Interfaces

Asterisk Timing Interfaces

In the past, if internal timing were desired for an Asterisk system, then the only source
acceptable was from DAHDI. Beginning with Asterisk 1.6.1, a new timing API was introduced
which allows for various timing modules to be used.

Included with Asterisk are the following timing modules:

res_timing_pthread.so
res_timing_dahdi.so
res_timing_timerfd.so (Beginning with Asterisk 1.6.2)

res_timing_pthread uses the POSIX pthreads library in order to provide timing. Since the code uses a commonly-implemented set of functions, res_timing_pthread is portable to many types of systems. In fact, this is the only timing source currently usable on a non-Linux system. Due to the fact that a single userspace thread is used to provide timing for all users of the timer, res_timing_pthread is also the least efficient of the timing sources and has been known to lose its effectiveness in a heavily-loaded environment.

res_timing_dahdi uses timing mechanisms provided by DAHDI. This method of timing was previously the only means by which Asterisk could receive timing. It has the benefit of being efficient, and if a system is already going to use DAHDI hardware, then it makes good sense to use this timing source. If, however, there is no need for DAHDI other than as a timing source, this timing source may seem unattractive. For people who are upgrading from 1.4 and are used to the old ztdummy timing interface, res_timing_dahdi provides the interface to dahdi_dummy, which is ztdummy's replacement.

> ℹ️ dahdi_dummy is no longer required in DAHDI-linux versions 2.3.0+. If the dahdi kernel module is loaded, it will be able to provide timing regardless of the span configuration.

res_timing_timerfd uses a timing mechanism provided directly by the Linux kernel. This timing interface is only available on Linux systems using a kernel version at least 2.6.25 and a glibc version at least 2.8. This interface has the benefit of being very efficient, but at the time this is being written, it is a relatively new feature on Linux, meaning that its availability is not widespread.

What Asterisk does with the Timing Interfaces

By default, Asterisk will build and load all of the timing interfaces. These timing interfaces are "ordered" based on a hard-coded priority number defined in each of the modules. As of the time of this writing, the preferences for the
modules is the following: res_timing_timerfd.so, res_timing_dahdi.so, res_timing_pthread.so.

The only functionality that requires internal timing is IAX2 trunking. It may also be used when generating audio for playback, such as from a file. Even though internal timing is not a requirement for most Asterisk functionality, it may be advantageous to use it since the alternative is to use timing based on incoming frames of audio. If there are no incoming frames or if the incoming frames of audio are from an unreliable or jittery source, then the corresponding outgoing audio will also be unreliable, or even worse, nonexistent. Using internal timing prevents such unreliability.

Customizations/Troubleshooting

Now that you know Asterisk's default preferences for timing modules, you may decide that you have a different preference. Maybe you're on a timerfd-capable system but you would prefer to

get your timing from DAHDI since you already are
using DAHDI to drive your hardware.

Alternatively, you may have been directed to this document due to an error you are currently
experiencing with Asterisk. If you receive an error message regarding timing not working
correctly, then you can use one of the following suggestions to disable a faulty timing module.

1. Don't build the timing modules you know you will not use. You can disable the compilation of
any of the timing modules using menuselect. The modules are listed in the "Resource Modules"
section. Note that if you have already built Asterisk and have received an error about a timing
module not working properly, it is not sufficient to disable it from being built. You will need to
remove the module from your modules directory (by default, /usr/lib/asterisk/modules) to make
sure that it does not get loaded again.

2. Build, but do not load the timing modules you know you will not use. You can edit
modules.conf using "noload" lines to disable the loading of specific timing modules by default.
Based on the note in the section above, you may realize that your Asterisk setup does not
require internal timing at all. If this is the case, you can safely noload all timing modules.

> ⚠ Some confusion has arisen regarding the fact that non-DAHDI timing interfaces
> are available now. One common misconception which has arisen is that since
> timing can be provided elsewhere, DAHDI is no longer required for using the
> MeetMe application. Unfortunately, this is not the case. In addition to
> providing timing, DAHDI also provides a conferencing engine which the MeetMe
> application requires.
>
> Starting with Asterisk 1.6.2, however, there will be a new application,
> ConfBridge, which will be capable of conference bridging without the use
> of DAHDI's built-in mixing engine.

Using the Hoard Memory Allocator with Asterisk

Install the Hoard Memory Allocator

Download Hoard from http://www.hoard.org/ either via a package or the source tarball.

If downloading the source, unpack the tarball and follow the instructions in the README file to
build libhoard for your platform.

Configure asterisk

Run ./configure in the root of the asterisk source directory, passing the **--with-hoard** option
specifying the location of the libhoard shared library.

For example:

```
./configure --with-hoard=/usr/src/hoard-371/src/
```

Note that we don't specify the full path to libhoard.so, just the directory where it resides.

Enable Hoard in menuselect

Run 'make menuselect' in the root of the asterisk source distribution. Under 'Compiler Flags' select the 'USE_HOARD_ALLOCATOR' option. If the option is not available (shows up with XXX next to it) this means that configure was not able to find libhoard.so. Check that the path you passed to the **--with-hoard** option is correct. Re-run **./configure** with the correct option and then repeat this step.

Make and install asterisk

Run the standard build commands:

```
# make
# make install
```

Video Console

Video Console Support in Asterisk

Some console drivers (at the moment chan_oss.so) can be built with support for sending and receiving video. In order to have this working you need to perform the following steps:

Enable building the video_console support

The simplest way to do it is add this one line to channels/Makefile:

```
chan_oss.so: _ASTCFLAGS+=-DHAVE_VIDEO_CONSOLE
```

Install prerequisite packages

The video_console support relies on the presence of SDL, SDL_image and ffmpeg libraries, and of course on the availability of X11

On Linux, these are supplied by

- libncurses-dev
- libsdl1.2-dev
- libsdl-image1.2-dev
- libavcodec-dev
- libswcale-dev

On FreeBSD, you need the following ports:

- multimedia/ffmpeg (2007.10.04)
- devel/sdl12 graphics/sdl_image

Build and install asterisk with all the above

Make sure you do a 'make clean' and run configure again after you have installed the required

packages, to make sure that the required pieces are found.

Check that chan_oss.so is generated and correctly installed.

Update configuration files

Video support requires explicit configuration as described below:

oss.conf

You need to set various parameters for video console, the easiest way is to uncomment the following line in oss.conf by removing the leading ';'

```
;[general](+,my_video,skin2)
```

You also need to manually copy the two files

- images/kpad2.jpg
- images/font.png

into the places specified in oss.conf, which in the sample are set to

```
keypad = /tmp/kpad2.jpg
keypad_font = /tmp/font.png
```

other configuration parameters are described in oss.conf.sample

sip.conf

To actually run a call using SIP (the same probably applies to iax.conf) you need to enable video support as following

```
[general](+)
videosupport=yes
allow=h263       ; this or other video formats
allow=h263p      ; this or other video formats
```

You can add other video formats e.g. h261, h264, mpeg if they are supported by your version of libavcodec.

Run the Program

Run asterisk in console mode e.g. asterisk -vdc

If video console support has been successfully compiled in, then you will see the "console startgui" command available on the CLI interface. Run the command, and you should see a window like this http://info.iet.unipi.it/~luigi/asterisk_video_console.jpg

To exit from this window, in the console run "console stopgui".

If you want to start a video call, you need to configure your dialplan so that you can reach (or be reachable) by a peer who can support video. Once done, a video call is the same as an ordinary call:

"console dial ...", "console answer", "console hangup" all work the same.

To use the GUI, and also configure video sources, see the next section.

Video Sources

Video sources are declared with the "videodevice=..." lines in oss.conf where the ... is the name of a device (e.g. /dev/video0 ...) or a string starting with X11 which identifies one instance of an X11 grabber.

You can have up to 9 sources, displayed in thumbnails in the gui, and select which one to transmit, possibly using Picture-in-Picture.

For webcams, the only control you have is the image size and frame rate (which at the moment is the same for all video sources). X11 grabbers capture a region of the X11 screen (it can contain anything, even a live video) and use it as the source. The position of the grab region can be configured using the GUI below independently for each video source.

The actual video sent to the remote side is the device selected as "primary" (with the mouse, see below), possibly with a small 'Picture-in-Picture' of the "secondary" device (all selectable with the mouse).

GUI Commands and Video Sources

(most of the text below is taken from channels/console_gui.c)

The GUI is made of 4 areas: remote video on the left, local video on the right, keypad with all controls and text windows in the center, and source device thumbnails on the top. The top row is not displayed if no devices are specified in the config file.

```
  _____
 |    _____    _____    _____    _____    _____    _____    _____
 |
 |   | tn.1 |  | tn.2 |  | tn.3 |  | tn.4 |  | tn.5 |  | tn.6 |  | tn.7 |
 |
 |   |_____|  |_____|  |_____|  |_____|  |_____|  |_____|  |_____|
 |
 |    _____    _____    _____    _____    _____    _____    _____
 |
 |   |_____|  |_____|  |_____|  |_____|  |_____|  |_____|  |_____|
 |
 |    _____    _____    _____
 |
 | |                  | |                  | |                  |
 |
 | |                  | |                  | |                  |
 |
 | |                  | |                  | |                  |
 |
 | |   remote video   | |                  | |   local video    |
 |
 | |                  | |                  | |          _____  |
 |
 | |                  | |     keypad       | |         | PIP || 
 |
 | |                  | |                  | |         |_____|| 
 |
 | |_____| |                  | |_____|
 |
 |                      |                  |
 |
 |                      |                  |
 |
 |                      |_____|
 |
 |
 |_____|
```
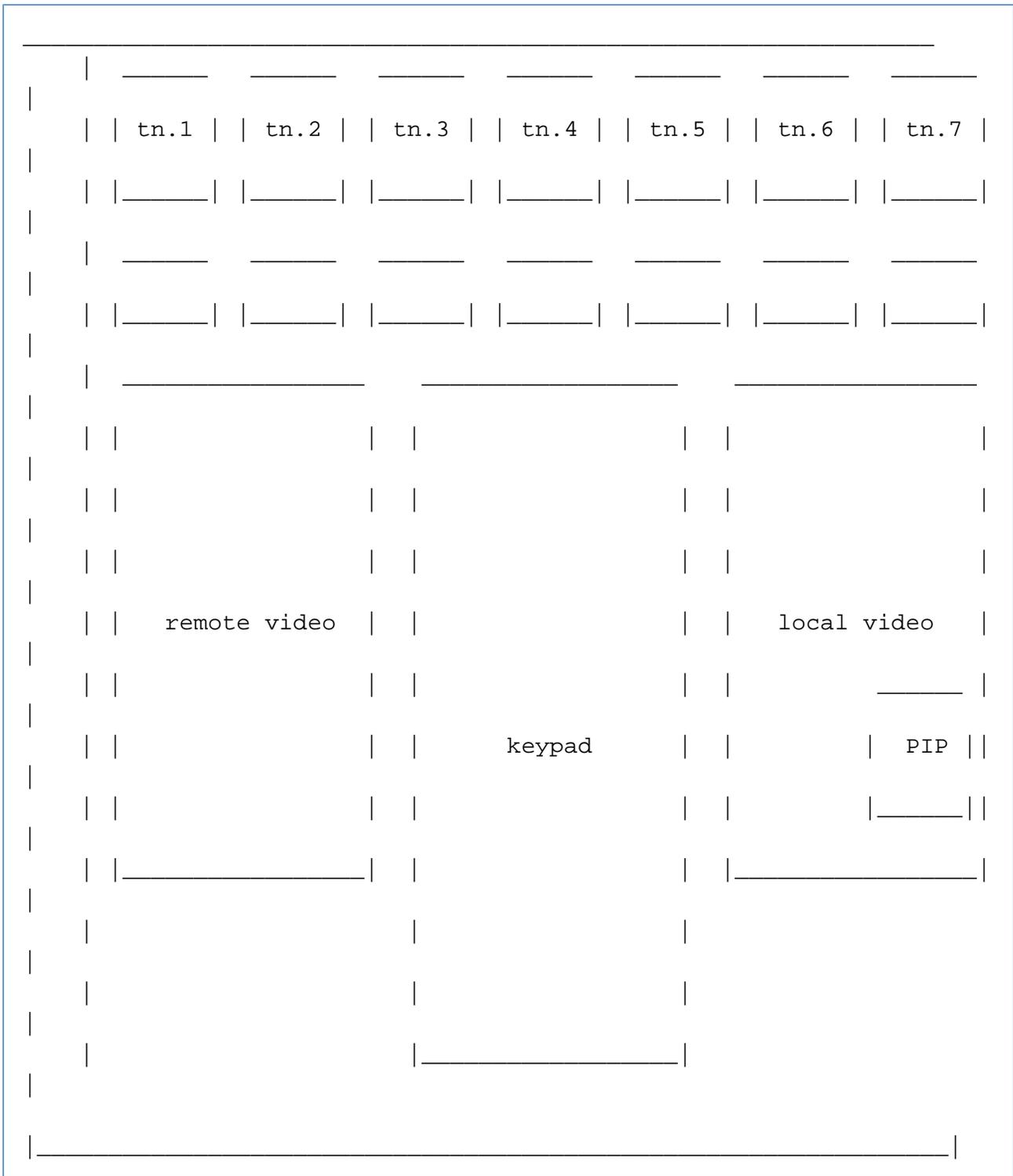
The central section is built using an image (jpg, png, maybe gif too) for the skin and other GUI elements. Comments embedded in the image indicate to what function each area is mapped to.

Another image (png with transparency) is used for the font.

Mouse and keyboard events are detected on the whole surface, and handled differently according to their location:

- Center/right click on the local/remote window are used to resize the corresponding window
- Clicks on the thumbnail start/stop sources and select them as primary or secondary video sources
- Drag on the local video window are used to move the captured area (in the case of X11 grabber) or the picture-in-picture position
- Keystrokes on the keypad are mapped to the corresponding key; keystrokes are used as keypad functions, or as text input if we are in text-input mode.
- Drag on some keypad areas (sliders etc.) are mapped to the corresponding functions (mute/unmute audio and video, enable/disable Picture-in-Picture, freeze the incoming video, dial numbers, pick up or hang up a call, ...)

Video Telephony

Asterisk and Video telephony

Asterisk supports video telephony in the core infrastructure. Internally, it's one audio stream and one video stream in the same call. Some channel drivers and applications has video support, but not all.

Codecs and formats

Asterisk supports the following video codecs and file formats. There's no video transcoding so you have to make sure that both ends support the same video format.

```
Codec Format
----- ----------
H.263 read/write
H.264 read/write
H.261 -  Passthrough only
```

Note that the file produced by Asterisk video format drivers is in no generic video format. Gstreamer has support for producing these files and converting from various video files to Asterisk video+audio files.

Note that H.264 is not enabled by default. You need to add that in the channel configuration file.

```
Channel drivers
--------------
SIP  The SIP channel driver (chan_sip.so) has support for video
IAX2  Supports video calls (over trunks too)
Local  Forwards video calls as a proxy channel
Agent  Forwards video calls as a proxy channel
oss  Has support for video display/decoding, see video_console.txt
```

Applications

This is not yet a complete list. These dialplan applications are known to handle video:

- Voicemail - Video voicemail storage (does not attach video to e-mail)
- Record - Records audio and video files (give audio format as argument)
- Playback - Plays a video while being instructed to play audio
- Echo - Echos audio and video back to the user

There is a development group working on enhancing video support for Asterisk.

If you want to participate, join the asterisk-video mailing list on http://lists.digium.com

Updates to this file are more than welcome!

Manipulating Party ID Information

Introduction

This chapter aims to explain how to use some of the features available to manipulate party ID information. It will not delve into specific channel configuration options described in the respective sample configuration files. The party ID information can consist of Caller ID, Connected Line ID, redirecting to party ID information, and redirecting from party ID information. Meticulous control is needed particularly when interoperating between different channel technologies.

- Caller ID: The Caller ID information describes who is originating a call.
- Connected Line ID: The Connected Line ID information describes who is connected to the other end of a call while a call is established. Unlike Caller ID, the connected line information can change over the life of a call when call transfers are performed. The connected line information can also change in either direction because either end could transfer the call. For ISDN it is known as Connected Line Identification Presentation (COLP), Connected Line Identification Restriction (COLR), and Explicit Call Transfer (ECT). For SIP it is known either as P-Asserted-Identity or Remote-Party-Id.
- Redirecting information: When a call is forwarded, the call originator is informed that the call is redirecting-to a new destination. The new destination is also informed that the incoming call is redirecting-from the forwarding party. A call can be forwarded repeatedly until a new destination answers it or a forwarding limit is reached.

Tools available

Asterisk contains several tools for manipulating the party ID information for a call. Additional information can be found by using the 'core show function' or 'core show application' console commands at the Asterisk CLI. The following list identifies some of the more common tools for manipulating the party ID information:

- `CALLERID(datatype[,caller-id])`
- `CONNECTEDLINE(datatype[,i])`
- `REDIRECTING(datatype[,i])`
- `Dial()` and `Queue()` dialplan application 'I' option
- Interception macros
- Channel driver specific configuration options.

CALLERID dialplan function

The CALLERID function has been around for quite a while and its use is straightforward. It is used to examine and alter the caller information that came into the dialplan with the call. Then

the call with it's caller information passes on to the destination using the Dial() or Queue() application.

The CALLERID information is passed during the initial call setup. However, depending on the channel technology, the caller name may be delayed. Q.SIG is an example where the caller name may be delayed so your dialplan may need to wait for it.

CONNECTEDLINE dialplan function

The CONNECTEDLINE function does the opposite of the CALLERID function. CONNECTEDLINE can be used to setup connected line information to be sent when the call is answered. You can use it to send new connected line information to the remote party on the channel when a call is transferred. The CONNECTEDLINE information is passed when the call is answered and when the call is transferred.

Since the connected line information can be sent while a call is connected, you may need to prevent the channel driver from acting on a partial update. The 'i' option is used to inhibit the channel driver from sending the changed information immediately.

REDIRECTING dialplan function

The REDIRECTING function allows you to report information about forwarded/deflected calls to the caller and to the new destination. The use of the REDIRECTING function is the most complicated of the party information functions.

The REDIRECTING information is passed during the initial call setup and while the call is being routed through the network. Since the redirecting information is sent before a call is answered, you need to prevent the channel driver from acting on a partial update. The 'i' option is used to inhibit the channel driver from sending the changed information immediately.

The incoming call may have already been redirected. An incoming call has already been redirected if the REDIRECTING(count) is not zero. (Alternate indications are if the REDIRECTING(from-num-valid) is non-zero or if the REDIRECTING(from-num) is not empty.)

There are several things to do when a call is forwarded by the dialplan:

- Setup the REDIRECTING(to-xxx) values to be sent to the caller.
- Setup the REDIRECTING(from-xxx) values to be sent to the new destination.
- Increment the REDIRECTING(count).
- Set the REDIRECTING(reason).
- Dial() the new destination.

Special REDIRECTING considerations for ISDN

Special considerations for Q.SIG and ISDN point-to-point links are needed to make the DivertingLegInformation1, DivertingLegInformation2, and DivertingLegInformation3 messages operate properly.

You should manually send the COLR of the redirected-to party for an incoming redirected call if the incoming call could experience further redirects. For chan_misdn, just set the REDIRECTING(to-num,i) = ${EXTEN} and set the REDIRECTING(to-num-pres) to the COLR. For chan_dahdi, just set the REDIRECTING(to-num,i) = CALLERID(dnid) and set the

REDIRECTING(to-num-pres) to the COLR. (Setting the REDIRECTING(to-num,i) value may not be necessary since the channel driver has already attempted to preset that value for automatic generation of the needed DivertingLegInformation3 message.)

For redirected calls out a trunk line, you need to use the 'i' option on all of the REDIRECTING statements before dialing the redirected-to party. The call will update the redirecting-to presentation (COLR) when it becomes available.

Dial() and Queue() dialplan application 'I' option

In the dialplan applications Dial() and Queue(), the 'I' option is a brute force option to block connected line and redirecting information updates while the application is running. Blocking the updates prevents the update from overwriting any CONNECTEDLINE or REDIRECTING values you may have setup before running the application.

The option blocks all redirecting updates since they should only happen before a call is answered. The option only blocks the connected line update from the initial answer. Connected line updates resulting from call transfers happen after the application has completed. Better control of connected line and redirecting information is obtained using the interception macros.

Interception macros

The interception macros give the administrator an opportunity to alter connected line and redirecting information before the channel driver is given the information. If the macro does not change a value then that is what is going to be passed to the channel driver.

The tag string available in CALLERID, CONNECTEDLINE, and REDIRECTING is useful for the interception macros to provide some information about where the information originally came from.

The 'i' option of the CONNECTEDLINE dialplan function should always be used in the CONNECTED_LINE interception macros. The interception macro always passes the connected line information on to the channel driver when the macro exits. Similarly, the 'i' option of the REDIRECTING dialplan function should always be used in the REDIRECTING interception macros.

- ${REDIRECTING_CALLEE_SEND_MACRO}
  Macro to call before sending a redirecting update to the callee. This macro may never be needed since the redirecting updates should only go from the callee to the caller direction. It is available for completeness.

- ${REDIRECTING_CALLEE_SEND_MACRO_ARGS}
  Arguments to pass to ${REDIRECTING_CALLEE_SEND_MACRO}.

- ${REDIRECTING_CALLER_SEND_MACRO}
  Macro to call before sending a redirecting update to the caller.

- ${REDIRECTING_CALLER_SEND_MACRO_ARGS}
  Arguments to pass to ${REDIRECTING_CALLER_SEND_MACRO}.

- ${CONNECTED_LINE_CALLEE_SEND_MACRO}
  Macro to call before sending a connected line update to the callee.

- ${CONNECTED_LINE_CALLEE_SEND_MACRO_ARGS}
  Arguments to pass to ${CONNECTED_LINE_CALLEE_SEND_MACRO}.

- ${CONNECTED_LINE_CALLER_SEND_MACRO}
  Macro to call before sending a connected line update to the caller.

- ${CONNECTED_LINE_CALLER_SEND_MACRO_ARGS}
  Arguments to pass to ${CONNECTED_LINE_CALLER_SEND_MACRO}.

Manipulation examples

The following examples show several common scenarios in which you may need to manipulate party ID information from the dialplan.

Simple recording playback

```
exten => 1000,1,NoOp
; The CONNECTEDLINE information is sent when the call is answered.
exten => 1000,n,Set(CONNECTEDLINE(name,i)="Company Name")
exten => 1000,n,Set(CONNECTEDLINE(name-pres,i)=allowed)
exten => 1000,n,Set(CONNECTEDLINE(num,i)=5551212)
exten => 1000,n,Set(CONNECTEDLINE(num-pres)=allowed)
exten => 1000,n,Answer
exten => 1000,n,Playback(tt-weasels)
exten => 1000,n,Hangup
```

Straightforward dial through

```
exten => 1000,1,NoOp
; The CONNECTEDLINE information is sent when the call is answered.
exten => 1000,n,Set(CONNECTEDLINE(name,i)="Company Name")
exten => 1000,n,Set(CONNECTEDLINE(name-pres,i)=allowed)
exten => 1000,n,Set(CONNECTEDLINE(num,i)=5551212)
exten => 1000,n,Set(CONNECTEDLINE(num-pres)=allowed)
; The I option prevents overwriting the CONNECTEDLINE information
; set above when the call is answered.
exten => 1000,n,Dial(SIP/1000,20,I)
exten => 1000,n,Hangup
```

Use of interception macro

```
[macro-add_pfx]
; ARG1 is the prefix to add.
; ARG2 is the number of digits at the end to add the prefix to.
; When the macro ends the CONNECTEDLINE data is passed to the
; channel driver.
exten => s,1,NoOp(Add prefix to connected line)
exten => s,n,Set(NOPREFIX=${CONNECTEDLINE(number):-${ARG2}})
exten => s,n,Set(CONNECTEDLINE(num,i)=${ARG1}${NOPREFIX})
exten => s,n,MacroExit

exten => 1000,1,NoOp
exten => 1000,n,Set(__CONNECTED_LINE_CALLER_SEND_MACRO=add_pfx)
exten => 1000,n,Set(__CONNECTED_LINE_CALLER_SEND_MACRO_ARGS=45,4)
exten => 1000,n,Dial(SIP/1000,20)
exten => 1000,n,Hangup
```

Simple redirection

```
exten => 1000,1,NoOp
; For Q.SIG or ISDN point-to-point we should determine the COLR for
this
; extension and send it if the call was redirected here.
exten =>
1000,n,GotoIf($[${REDIRECTING(count)}>0]?redirected:notredirected)
exten =>
1000,n(redirected),Set(REDIRECTING(to-num,i)=${CALLERID(dnid)})
exten => 1000,n,Set(REDIRECTING(to-num-pres)=allowed)
exten => 1000,n(notredirected),NoOp
; Determine that the destination has forwarded the call.
; ...
exten => 1000,n,Set(REDIRECTING(from-num,i)=1000)
exten => 1000,n,Set(REDIRECTING(from-num-pres,i)=allowed)
exten => 1000,n,Set(REDIRECTING(to-num,i)=2000)
; The DivertingLegInformation3 message is needed because at this
point
; we do not know the presentation (COLR) setting of the
redirecting-to
; party.
exten => 1000,n,Set(REDIRECTING(count,i)=$[${REDIRECTING(count)} +
1])
exten => 1000,n,Set(REDIRECTING(reason,i)=cfu)
; The call will update the redirecting-to presentation (COLR) when
it
; becomes available with a redirecting update.
exten => 1000,n,Dial(DAHDI/g1/2000,20)
exten => 1000,n,Hangup
```

Ideas for usage

The following is a list of ideas in which the manipulation of party ID information would be beneficial.

- IVR that updates connected name on each selection made.
- Disguise the true number of an individual with a generic company number.
- Use interception macros to make outbound connected number E.164 formatted.
- You can do a lot more in an interception macro than just manipulate party information...

Troubleshooting tips

- For CONNECTEDLINE and REDIRECTING, check the usage of the 'i' option.
- Check channel configuration settings. The default settings may not be what you want or expect.
- Check packet captures. Your equipment may not support what Asterisk sends.

For further reading...

- Relevant ETSI ISDN redirecting specification: EN 300 207-1
- Relevant ETSI ISDN COLP specification: EN 300 097-1

- Relevant ETSI ISDN ECT specification: EN 300 369-1
- Relevant Q.SIG ISDN redirecting specification: ECMA-174
- Relevant Q.SIG ISDN COLP specification: ECMA-148
- Relevant Q.SIG ISDN ECT specification: ECMA-178
- Relevant SIP RFC for P-Asserted-Id: RFC3325
- The expired draft (draft-ietf-sip-privacy-04.txt) defines Remote-Party-Id. Since Remote-Party-Id has not made it into an RFC at this time, its use is non-standard by definition.

Packet Loss Concealment (PLC)

What is PLC?

PLC stands for Packet Loss Concealment. PLC describes any method of generating new audio data when packet loss is detected. In Asterisk, there are two main flavors of PLC, generic and native. Generic PLC is a method of generating audio data on signed linear audio streams. Signed linear audio, often abbreviated "slin," is required since it is a raw format that has no companding, compression, or other transformations applied. Native PLC is used by specific codec implementations, such as iLBC and Speex, which generates the new audio in the codec's native format. Native PLC happens automatically when using a codec that supports native PLC. Generic PLC requires specific configuration options to be used and will be the focus of this document.

How does Asterisk detect packet loss?

Oddly, Asterisk does not detect packet loss when reading audio in. In order to detect packet loss, one must have a jitter buffer in use on the channel on which Asterisk is going to write missing audio using PLC. When a jitter buffer is in use, audio that is to be written to the channel is fed into the jitterbuffer. When the time comes to write audio to the channel, a bridge will request that the jitter buffer gives a frame of audio to the bridge so that the audio may be written. If audio is requested from the jitter buffer but the jitter buffer is unable to give enough audio to the bridge, then the jitter buffer will return an interpolation frame. This frame contains no actual audio data and indicates the number of samples of audio that should be inserted into the frame.

PLC Background on Translation

As stated in the introduction, generic PLC can only be used on slin audio. The majority of audio communication is not done in slin, but rather using lower bandwidth codecs. This means that for PLC to be used, there must be a translation step involving slin on the write path of a channel. This means that PLC cannot be used if the codecs on either side of the bridge are the same or do not require a translation to slin in order to translate between them. For instance, a ulaw - ulaw call will not use PLC since no translation is required. In addition, a ulaw - alaw call will also not use PLC since the translation path does not include any step involving slin. One item of note is that slin must be present on the write path of a channel since that is the path where PLC is applied. Consider that Asterisk is bridging channels A and B. A uses ulaw for audio and B uses GSM. This translation involves slin, so things are shaping up well for PLC. Consider, however if Asterisk sets up the translation paths like so:

Fig. 1

```
A ----------- B <-ulaw<-slin<-GSM| |GSM-> | Asterisk |
ulaw->slin->GSM->| |<-GSM -----------
```

The arrows indicate the direction of audio flow. Each channel has a write path (the top arrow) and a read path (the bottom arrow). In this setup, PLC can be used when sending audio to A, but it cannot be used when sending audio to B. The reason is simple, the write path to A's channel contains a slin step, but the write path to B contains no slin step. Such a translation setup is perfectly valid, and Asterisk can potentially set up such a path depending on circumstances. When we use PLC, however, we want slin audio to be present on the write paths of both A and B. A visual representation of what we want is the following:

Fig. 2

```
A ----------- B <-ulaw<-slin| |slin->GSM-> | Asterisk |
ulaw->slin->| |<-slin<-GSM -----------
```
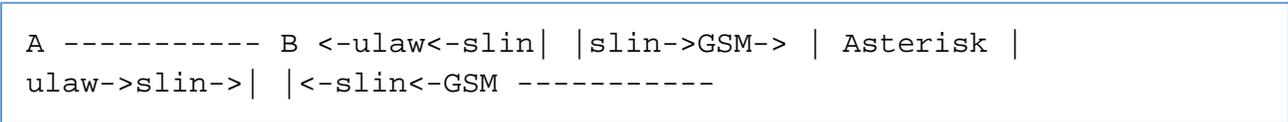
In this scenario, the write paths for both A and B begin with slin, and so PLC may be applied to either channel. This translation behavior has, in the past been doable with the transcode_via_sln option in asterisk.conf. Recent changes to the PLC code have also made the genericplc option in codecs.conf imply the transcode_via_sln option. The result is that by enabling genericplc in codecs.conf, the translation path set up in Fig. 2 should automatically be used.

PLC Restrictions and Caveats

One restriction that has not been spelled out so far but that has been hinted at is the presence of a bridge. The term bridge in this sense means two channels exchanging audio with one another. A bridge is required because use of a jitter buffer is a prerequisite for using PLC, and a jitter buffer is only used when bridging two channels. This means that one-legged calls, (e.g. calls to voicemail, to an IVR, to an extension that just plays back audio) will not use PLC. In addition, MeetMe and ConfBridge calls will not use PLC. It should be obvious, but it bears mentioning, that PLC cannot be used when using a technology's native bridging functionality. For instance, if two SIP channels can exchange RTP directly, then Asterisk will never be able to process the audio in the first place. Since translation of audio is a requirement for using PLC, and translation will not allow for a native bridge to be created, this is something that is not likely to be an issue, though. Since a jitter buffer is a requirement in order to use PLC, it should be noted that simply enabling the jitter buffer via the jbenable option may not be enough. For instance, if bridging two SIP channels together, the default behavior will not be to enable jitter buffers on either channel. The rationale is that the jitter will be handled at the endpoints to which Asterisk is sending the audio. In order to ensure that a jitter buffer is used in all cases, one must enable the jbforce option for channel types on which PLC is desired.

Requirements for PLC Use

The following are all required for PLC to be used:

1. Enable genericplc in the plc section of codecs.conf
2. Enable (and potentially force) jitter buffers on channels
3. Two channels must be bridged together for PLC to be used (no Meetme or one-legged calls)
4. The audio must be translated between the two channels and must have slin as a step in the translation process.

PLC Tips

One of the restrictions mentioned is that PLC will only be used when two audio channels are bridged together. Through the use of Local channels, you can create a bridge even if the call is, for all intents and purposes, one-legged. By using a combination of the /n and /j suffixes for a Local channel, one can ensure that the Local channel is not optimized out of the talk path and that a jitter buffer is applied to the Local channel as well. Consider the following simple dialplan:

```
[example]
exten => 1,1,Playback(tt-weasels)
exten => 2,1,Dial(Local/1@example/nj)
```

When dialing extension 1, PLC cannot be used because there will be only a single channel involved. When dialing extension 2, however, Asterisk will create a bridge between the incoming channel and the Local channel, thus allowing PLC to be used.

Phone Provisioning in Asterisk

Asterisk includes basic phone provisioning support through the res_phoneprov module. The current implementation is based on a templating system using Asterisk dialplan function and variable substitution and obtains information to substitute into those templates from phoneprov.conf and users.conf. A profile and set of templates is provided for provisioning Polycom phones. Note that res_phoneprov is currently limited to provisioning a single user per device.

Configuration of phoneprov.conf

The configuration file, phoneprov.conf, is used to set up the built-in variables SEVER and SERVER_PORT, to define a default phone profile to use, and to define different phone profiles available for provisioning.

The [general] section

Below is a sample of the general section of phoneprov.conf:

```
[general]
;serveriface=eth0
;serveraddr=192.168.1.1
;serverport=5060
default_profile=polycom
```

By default, res_phoneprov will set the SERVER variable to the IP address on the server that the requesting phone uses to contact the asterisk HTTP server. The SERVER_PORT variable will default to the bindport setting in sip.conf.

Should the defaults be insufficient, there are two choices for overriding the default setting of the SERVER variable. If the IP address of the server is known, or the hostname resolvable by the

phones, the appropriate serveraddr value should be set. Alternatively, the network interface that the server listens on can be set by specifying a serveriface and SERVER will be set to the IP address of that interface. Only one of these options should be set.

The default SERVER_PORT variable can be overridden by setting the serverport. If bindport is not set in sip.conf and serverport is not specified, it is set to a default value of 5060.

Any user set for auto-provisioning in users.conf without a specified profile will be assumed to belong to the profile set with default_profile.

Creating Phone Profiles

A phone profile is basically a list of files that a particular group of phones needs to function. For most phone types there are files that are identical for all phones (firmware, for instance) as well as a configuration file that is specific to individual phones. res_phoneprov breaks these two groups of files into static files and dynamic files, respectively. A sample profile:

```
[polycom]
staticdir => configs/
mime_type => text/xml
setvar =>
CUSTOM_CONFIG=/var/lib/asterisk/phoneprov/configs/custom.cfg
static_file => bootrom.ld,application/octet-stream
static_file => bootrom.ver,plain/text
static_file => sip.ld,application/octet-stream
static_file => sip.ver,plain/text
static_file => sip.cfg
static_file => custom.cfg
${TOLOWER(${MAC})}.cfg => 000000000000.cfg
${TOLOWER(${MAC})}-phone.cfg => 000000000000-phone.cfg config/
${TOLOWER(${MAC})} => polycom.xml
${TOLOWER(${MAC})}-directory.xml => 000000000000-directory.xml
```

A static_file is set by specifying the file name, relative to AST_DATA_DIR/phoneprov. The mime-type of the file can optionally be specified after a comma. If staticdir is set, all static files will be relative to the subdirectory of AST_DATA_DIR/phoneprov specified.

Since phone-specific config files generally have file names based on phone-specifc data, dynamic filenames in res_phoneprov can be defined with Asterisk dialplan function and variable substitution. In the above example, ${TOLOWER(${MAC})}.cfg = 000000000000.cfg would define a relative URI to be served that matches the format of MACADDRESS.cfg, all lower case. A request for that file would then point to the template found at AST_DATA_DIR/phoneprov/000000000000.cfg. The template can be followed by a comma and mime-type. Notice that the dynamic filename (URI) can contain contain directories. Since these files are dynamically generated, the config file itself does not reside on the filesystem-only the template. To view the generated config file, open it in a web browser. If the config file is XML, Firefox should display it. Some browsers will require viewing the source of the page requested.

A default mime-type for the profile can be defined by setting mime-type. If a custom variable is required for a template, it can be specified with setvar. Variable substitution on this value is done while building the route list, so ${USERNAME} would expand to the username of the users.conf user that registers the dynamic filename.

> ⚠ Any dialplan function that is used for generation of dynamic file names MUST be loaded before res_phoneprov. Add "preload = modulename.so" to modules.conf for required functions. In the example above, "preload = func_strings.so" would be required.

Configuration of users.conf

The asterisk-gui sets up extensions, SIP/IAX2 peers, and a host of other settings. User-specific settings are stored in users.conf. If the asterisk-gui is not being used, manual entries to users.conf can be made.

The [general] section

There are only two settings in the general section of users.conf that apply to phone provisioning: localextenlength which maps to template variable EXTENSION_LENGTH and vmexten which maps to the VOICEMAIL_EXTEN variable.

Individual Users

To enable auto-provisioning of a phone, the user in users.conf needs to have:

```
...
autoprov=yes
macaddress=deadbeef4dad
profile=polycom
```

The profile is optional if a default_profile is set in phoneprov.conf. The following is a sample users.conf entry, with the template variables commented next to the settings:

```
[6001]
callwaiting = yes
context = numberplan-custom-1
hasagent = no
hasdirectory = yes
hasiax = no
hasmanager = no
hassip = yes
hasvoicemail = yes
host = dynamic
mailbox = 6001
threewaycalling = yes
deletevoicemail = no
autoprov = yes
profile = polycom
directmedia = no
nat = no
fullname = User Two ; ${DISPLAY_NAME}
secret = test ; ${SECRET}
username = 6001 ; ${USERNAME}
macaddress = deadbeef4dad ; ${MAC}
label = 6001 ; ${LABEL}
cid_number = 6001 ; ${CALLERID}
```

The variables above, are the user-specfic variables that can be substituted into dynamic filenames and config templates.

Phone Provisioning Templates

Configuration templates are a generic way to configure phones with text-based configuration files. Templates can use any loaded dialplan function and all of the variables created by phoneprov.conf and users.conf. A short example is the included 000000000000.cfg Polycom template:

```
<?xml version="1.0" standalone="yes"?>
  <APPLICATION
    APP_FILE_PATH="sip.ld"
    CONFIG_FILES="${IF($[${STAT(e,${CUSTOM_CONFIG})}] ? "custom.cfg,
")}config/${TOLOWER(${MAC})}, sip.cfg"
    MISC_FILES="" LOG_FILE_DIRECTORY=""
 />
```

This template uses dialplan functions, expressions, and a couple of variables to generate a config file to instruct the Polycom where to pull other needed config files. If a phone with MAC address 0xDEADBEEF4DAD requests this config file, and the filename that is stored in variable

CUSTOM_CONFIG does not exist, then the generated output would be:

```
<?xml version="1.0" standalone="yes"?>
  <APPLICATION
    APP_FILE_PATH="sip.ld"
    CONFIG_FILES="config/deadbeef4dad, sip.cfg"
    MISC_FILES=""
    LOG_FILE_DIRECTORY=""
/>
```

The Polycom phone would then download both sip.cfg (which would be registered in phoneprov.conf as a static file) and config/deadbeef4dad (which would be registered as a dynamic file pointing to another template, polycom.xml).

res_phoneprov also registers its own dialplan function: PP_EACH_USER. This function was designed to be able to print out a particular string for each user that res_phoneprov knows about. An example use of this function is the template for a Polycom contact directory:

```
<?xml version="1.0" standalone="yes"?>
<directory>
  <item_list>

${PP_EACH_USER(<item><fn>%{DISPLAY_NAME}</fn><ct>%{CALLERID}</ct><bw>1
</item_list>
</directory>
```

PP_EACH_USER takes two arguments. The first is the string to be printed for each user. Any variables that are to be substituted need to be in the format %{VARNAME} so that Asterisk doesn't try to substitute the variable immediately before it is passed to PP_EACH_USER. The second, optional, argument is a MAC address to exclude from the list iterated over (so, in this case, a phone won't be listed in its own contact directory).

Phone Provisioning, Putting it all together

Make sure that manager.conf has:

```
[general]
enabled = yes
webenabled = yes
```

and that http.conf has:

```
[general]
enabled = yes
bindaddr = 192.168.1.1 ; Your IP here
bindport = 8088 ; Or port 80 if it is the only http server running
on the machine
```

With phoneprov.conf and users.conf in place, start Astersik. From the CLI, type "http show status". An example output:

```
HTTP Server Status:
Prefix: /asterisk
Server Enabled and Bound to 192.168.1.1:8088

Enabled URI's:
/asterisk/httpstatus => Asterisk HTTP General Status
/asterisk/phoneprov/... => Asterisk HTTP Phone Provisioning Tool
/asterisk/manager => HTML Manager Event Interface
/asterisk/rawman => Raw HTTP Manager Event Interface
/asterisk/static/... => Asterisk HTTP Static Delivery
/asterisk/mxml => XML Manager Event Interface
Enabled Redirects:
 None.
POST mappings:
 None.
```

There should be a phoneprov URI listed. Next, from the CLI, type "phoneprov show routes" and verify that the information there is correct. An example output for Polycom phones woud look like:

```
Static routes

Relative URI Physical location
sip.ver configs/sip.ver
sip.ld configs/sip.ld
bootrom.ver configs/bootrom.ver
sip.cfg configs/sip.cfg
bootrom.ld configs/bootrom.ld
custom.cfg configs/custom.cfg
Dynamic routes
Relative URI Template
deadbeef4dad.cfg 000000000000.cfg
deadbeef4dad-directory.xml
000000000000-directory.xml
deadbeef4dad-phone.cfg
000000000000-phone.cfg
config/deadbeef4dad polycom.xml
```

With the above examples, the phones would be pointed to:

http://192.168.1.1:8080/asterisk/phoneprov

for pulling config files.

Templates would all be placed in AST_DATA_DIR/phoneprov and static files would be placed in AST_DATA_DIR/phoneprov/configs. Examples of valid URIs would be:

- http://192.168.1.1:8080/asterisk/phoneprov/sip.cfg
- http://192.168.1.1:8080/asterisk/phoneprov/deadbeef4dad.cfg
- http://192.168.1.1:8080/asterisk/phoneprov/config/deadbeef4dad

Reference Information Introduction

Introductory section to the Asterisk Reference Manual. Read this section first.

License Information

License Information

Asterisk is distributed under the GNU General Public License version 2 and is also available under alternative licenses negotiated directly with Digium, Inc. If you obtained Asterisk under the GPL, then the GPL applies to all loadable Asterisk modules used on your system as well, except as defined below. The GPL (version 2) is included in this source tree in the file COPYING.

This package also includes various components that are not part of Asterisk itself; these components are in the 'contrib' directory and its subdirectories. These components are also distributed under the GPL version 2 as well.

Digium, Inc. (formerly Linux Support Services) holds copyright and/or sufficient licenses to all components of the Asterisk package, and therefore can grant, at its sole discretion, the ability for companies, individuals, or organizations to create proprietary or Open Source (even if not GPL) modules which may be dynamically linked at runtime with the portions of Asterisk which fall under our copyright/license umbrella, or are distributed under more flexible licenses than GPL.

If you wish to use our code in other GPL programs, don't worry - there is no requirement that you provide the same exception in your GPL'd products (although if you've written a module for Asterisk we would strongly encourage you to make the same exception that we do).

Specific permission is also granted to link Asterisk with OpenSSL, OpenH323 and/or the UW IMAP Toolkit and distribute the resulting binary files.

In addition, Asterisk implements two management/control protocols: the Asterisk Manager Interface (AMI) and the Asterisk Gateway Interface (AGI). It is our belief that applications using these protocols to manage or control an Asterisk instance do not have to be licensed under the GPL or a compatible license, as we believe these protocols do not create a 'derivative work' as referred to in the GPL. However, should any court or other judiciary body find that these protocols do fall under the terms of the GPL, then we hereby grant you a license to use these protocols in combination with Asterisk in external applications licensed under any license you wish.

The 'Asterisk' name and logos are trademarks owned by Digium, Inc., and use of them is subject to our trademark licensing policies. If you wish to use these trademarks for purposes other than simple redistribution of Asterisk source code obtained from Digium, you should contact our licensing department to determine the necessary steps you must take. For more information on this policy, please read: http://www.digium.com/en/company/profile/trademarkpolicy.php

If you have any questions regarding our licensing policy, please contact us:

+1.877.344.4861 (via telephone in the USA)
+1.256.428.6000 (via telephone outside the USA)
+1.256.864.0464 (via FAX inside or outside the USA)
IAX2/pbx.digium.com (via IAX2)
licensing@digium.com (via email)

Digium, Inc.
445 Jan Davis Drive NW
Huntsville, AL 35806
United States

Hold Music License

The Hold (on hold) music included with the Asterisk distribution has been sourced from opsound.org which itself distributes the music under Creative Commons Attribution-ShareAlike 2.5 license.

Important Security Considerations

The pages in this section provide specific warnings about security that are pertinent to Asterisk.

Just because you're already familiar with securing your Linux machine, doesn't mean you can skip this section.

> 🔴 PLEASE READ THE FOLLOWING IMPORTANT SECURITY RELATED INFORMATION. IMPROPER CONFIGURATION OF ASTERISK COULD ALLOW UNAUTHORIZED USE OF YOUR FACILITIES, POTENTIALLY INCURRING SUBSTANTIAL CHARGES.

Asterisk security involves both network security (encryption, authentication) as well as dialplan security (authorization - who can access services in your pbx). If you are setting up Asterisk in production use, please make sure you understand the issues involved.

Network Security

Network Security

If you install Asterisk and use the "make samples" command to install a demonstration configuration, Asterisk will open a few ports for accepting VoIP calls. Check the channel configuration files for the ports and IP addresses.

If you enable the manager interface in manager.conf, please make sure that you access manager in a safe environment or protect it with SSH or other VPN solutions.

For all TCP/IP connections in Asterisk, you can set ACL lists that will permit or deny network access to Asterisk services. Please check the "permit" and "deny" configuration options in manager.conf and the VoIP channel configurations - i.e. sip.conf and iax.conf.

The IAX2 protocol supports strong RSA key authentication as well as AES encryption of voice and signaling. The SIP channel supports TLS encryption of the signaling, as well as SRTP (encrypted media).

Dialplan Security

Dialplan Security

First and foremost remember this:

> ✅ USE THE EXTENSION CONTEXTS TO ISOLATE OUTGOING OR TOLL SERVICES FROM ANY INCOMING CONNECTIONS.

You should consider that if any channel, incoming line, etc can enter an extension context that it has the capability of accessing any extension within that context.

Therefore, you should NOT allow access to outgoing or toll services in contexts that are accessible (especially without a password) from incoming channels, be they IAX channels, FX or other trunks, or even untrusted stations within you network. In particular, never ever put outgoing toll services in the "default" context. To make things easier, you can include the "default" context within other private contexts by using:

```
include => default
```

in the appropriate section. A well designed PBX might look like this:

```
[longdistance]
exten => _91NXXNXXXXXX,1,Dial(DAHDI/g2/${EXTEN:1})
include => local

[local]
exten => _9NXXNXXX,1,Dial(DAHDI/g2/${EXTEN:1})
include => default

[default]
exten => 6123,Dial(DAHDI/1)
```

🛑 DON'T FORGET TO TAKE THE DEMO CONTEXT OUT OF YOUR DEFAULT CONTEXT. There isn't really a security reason, it just will keep people from wanting to play with your Asterisk setup remotely.

Log Security

Log Security

Please note that the Asterisk log files, as well as information printed to the Asterisk CLI, may contain sensitive information such as passwords and call history. Keep this in mind when providing access to these resources.

Telephony Hardware

A PBX is only really useful if you can get calls into it. Of course, you can use Asterisk with VoIP calls (SIP, H.323, IAX, etc.), but you can also talk to the real PSTN through various cards.

Supported Hardware is divided into two general groups: DAHDI devices and non-DAHDI devices. The DAHDI compatible hardware supports pseudo-TDM conferencing and all call features through chan_dahdi, whereas non-DAHDI compatible hardware may have different features.

DAHDI compatible hardware

Digium, Inc (Primary Developer of Asterisk) DAHDI compatible hardware

Analog Interfaces

TDM410 and AEX410 - The TDM410P and AEX410 are modular four-port half-length PCI 2.2 and PCI-Express x1, respectively, cards that supports FXS and FXO station interfaces for connecting analog telephones and analog POTS lines through a PC.

TDM800 and AEX800 - The TDM800 and AEX800 are modular eight-port half-length PCI 2.2 and PCI-Express x1, respectively, cards that support FXS and FXO station interfaces for

connecting analog telephones and analog POTS lines through a PC.

TDM2400 and AEX2400 - The TDM2400 and AEX2400 are modular twenty-four port full-length PCI 2.2 and PCI-Express x1, respectively, cards that support FXS and FXO station interfaces for connecting analog telephones and analog POTS lines through a PC.

Digital Interfaces

TE122 and TE121 - The TE122 and TE122 are half-length, half-height PCI 2.2 and PCI-Express x1, respectively, single-span E1/T1/J1 PRI/PRA software-configurable interfaces for terminating digital telephony through a PC.
TE205/7, TE210/12 and TE220 - The TE205, TE210 and TE220 are half-length PCI 2.2 5.0V, PCI 2.2 3.3V, and PCI-Express x1 dual-span E1/T1/J1 PRI/PRA software-configurable interfaces for terminating digital telephony through a PC.
TE405/7, TE410/12 and TE420 - The TE405/7, TE410/12 and TE420 are half-length PCI 2.2 5.0V, PCI 2.2 3.3V, and PCI-Express x1 quad-span E1/T1/J1 PRI/PRA software-configurable interfaces for terminating digital telephony through a PC.
HA8 and HB8 - The HA8 and HB8 are half-length PCI2.2 and PCI-Express x1, respectively, modular eight-port EuroISDN BRI and Analog FXS/FXO interfaces for terminating telephony through a PC.
B410P - The B410P is a half-length PCI-2.2 fixed four-port EuroISDN BRI interface for terminating digital telephony through a PC.

Packet Interfaces

TC400 and TCE400 - The TC400 and TCE400 are half-length, half-height PCI 2.2 and PCI-Express x1, respectively, codec translation cards for DSP-based translations of patent-encumbered codecs such as G.729a and G.723.1

Non-DAHDI compatible hardware

Non-DAHDI compatible hardware

QuickNet, Inc. http://www.quicknet.net

Internet PhoneJack - Single FXS interface. Supports Linux telephony interface. DSP compression built-in.

Internet LineJack - Single FXS or FXO interface. Supports Linux telephony interface.

mISDN compatible hardware

mISDN compatible hardware

Any adapter with an mISDN driver should be compatible with chan_misdn. See the mISDN section for more information.

- **Digium, Inc.**
  **B410P - also compatible with mISDN
- **beroNet** http://www.beronet.com
  **BN4S0 - 4 Port BRI card (TE/NT)
  **BN8S0 - 8 Port BRI card (TE/NT)

- **Billion Card - Single Port BRI card (TE (/NT with crossed cable))**

Miscellaneous other interfaces

Miscellaneous other interfaces

ALSA http://www.alsa-project.org
Any ALSA compatible full-duplex sound card
OSS http://www.opensound.com
Any OSS compatible full-duplex sound card

Secure Calling

Top-level page for articles about securing VoIP calls using encryption.

Secure Calling Specifics

Asterisk supports a channel-agnostic method for handling secure call requirements. Since there is no single meaning of what constitutes a "secure call," Asterisk allows the administrator the control to define "secure" for themselves via the dialplan and channel-specific configuration files.

Channel-specific configuration

Currently the IAX2 and SIP channels support the call security features in Asterisk. Both channel-specific configuration files (iax2.conf and sip.conf) support the encryption=yes setting. For IAX2, this setting causes Asterisk to offer encryption when placing or receiving a call. To force encryption with IAX2, the forceencrypt=yes option is required. Due to limitations of SDP, encryption=yes in sip.conf results in a call with only a secure media offer, therefor forceencrypt=yes would be redundant in sip.conf.

If a peer is defined as requiring encryption but the endpoint does not support it, the call will fail with a HANGUPCAUSE of 58 (bearer capability does not exist).

Security-based dialplan branching

Each channel that supports secure signaling or media can implement a CHANNEL read callback function that specifies whether or not that channel meets the specified criteria. Currently, chan_iax2 and chan_sip implement these callbacks. Channels that do not support secure media or signaling will return an empty string when queried. For example, to only allow an inbound call that has both secure signaling and media, see the following example.

```
exten => 123,1,GotoIf($["${CHANNEL(secure_signaling)}" = "1"]?:fail)

exten => 123,n,GotoIf($["${CHANNEL(secure_media)}" = "1"]?:fail)
exten => 123,n,Dial(SIP/123)
exten => 123,n,Hangup
exten => 123,n(fail),Playback(vm-goodbye)
exten => 123,n,Hangup
```

Administrators can force outbound channels that are to be bridged to a calling channel to conform to secure media and signaling policies. For example, to first make a call attempt that has both secure signaling and media, but gracefully fall back to non-secure signaling and media see the following example:

```
exten => 123,1,NoOp(We got a call)
exten => 123,n,Set(CHANNEL(secure_bridge_signaling)=1)
exten => 123,n,Set(CHANNEL(secure_bridge_media)=1)
exten => 123,n,Dial(SIP/somebody)
exten => 123,n,NoOp(HANGUPCAUSE=${HANGUPCAUSE})
exten => 123,n,GotoIf($["${HANGUPCAUSE}"="58"]?encrypt_fail)
exten => 123,n,Hangup ; notify user that retrying via insecure
channel (user-provided prompt)
exten => 123,n(encrypt_fail),Playback(secure-call-fail-retry)
exten => 123,n,Set(CHANNEL(secure_bridge_signaling)=0)
exten => 123,n,Set(CHANNEL(secure_bridge_media)=0)
exten => 123,n,Dial(SIP/somebody)
exten => 123,n,Hangup
```

Secure Calling Tutorial

Overview

So you'd like to make some secure calls.

Here's how to do it, using Blink, a SIP soft client for Mac OS X, Windows, and Linux.

These instructions assume that you're running as the root user (sudo su -).

Part 1 (TLS)

Transport Layer Security (TLS) provides encryption for call signaling. It's a practical way to prevent people who aren't Asterisk from knowing who you're calling. Setting up TLS between Asterisk and a SIP client involves creating key files, modifying Asterisk's SIP configuration to enable TLS, creating a SIP peer that's capable of TLS, and modifying the SIP client to connect to Asterisk over TLS.

Keys

First, let's make a place for our keys.

```
mkdir /etc/asterisk/keys
```

Next, use the "ast_tls_cert" script in the "contrib/scripts" Asterisk source directory to make a self-signed certificate authority and an Asterisk certificate.

```
./ast_tls_cert -C pbx.mycompany.com -O "My Super Company" -d
/etc/asterisk/keys
```

- The "-C" option is used to define our host - DNS name or our IP address.
- The "-O" option defines our organizational name.
- The "-d" option is the output directory of the keys.

1. You'll be asked to enter a pass phrase for /etc/asterisk/keys/ca.key, put in something that you'll remember for later.
2. This will create the /etc/asterisk/keys/ca.crt file.
3. You'll be asked to enter the pass phrase again, and then the /etc/asterisk/keys/asterisk.key file will be created.
4. The /etc/asterisk/keys/asterisk.crt file will be automatically generated.
5. You'll be asked to enter the pass phrase a third time, and the /etc/asterisk/keys/asterisk.pem will be created, a combination of the asterisk.key and asterisk.crt files.

Next, we generate a client certificate for our SIP device.

```
./ast_tls_cert -m client -c /etc/asterisk/keys/ca.crt -k
/etc/asterisk/keys/ca.key -C phone1.mycompany.com -O "My Super
Company" -d /etc/asterisk/keys -o malcolm
```

- The "-m client" option tells the script that we want a client certificate, not a server certificate.
- The "-c /etc/asterisk/keys/ca.crt" option specifies which Certificate Authority (ourselves) that we're using.
- The "-k /etc/asterisk/keys/ca.key" provides the key for the above-defined Certificate Authority.
- The "-C" option, since we're defining a client this time, is used to define the hostname or IP address of our SIP phone
- The "-O" option defines our organizational name.
- The "-d" option is the output directory of the keys."
- The "-o" option is the name of the key we're outputting.

1. You'll be asked to enter the pass phrase from before to unlock /etc/asterisk/keys/ca.key.

Now, let's check the keys directory to see if all of the files we've built are there. You should have:

```
asterisk.crt
asterisk.csr
asterisk.key
asterisk.pem
malcolm.crt
malcolm.csr
malcolm.key
malcolm.pem
ca.cfg
ca.crt
ca.key
tmp.cfg
```

Next, copy the malcolm.pem and ca.crt files to the computer running the Blink soft client.

The Asterisk SIP configuration

Now, let's configure Asterisk to use TLS.

In the **sip.conf** configuration file, set the following:

```
tlsenable=yes
tlsbindaddr=0.0.0.0
tlscertfile=/etc/asterisk/keys/asterisk.pem
tlscafile=/etc/asterisk/keys/ca.crt
tlscipher=ALL
tlsclientmethod=tlsv1 ;none of the others seem to work with Blink as
the client
```

Here, we're enabling TLS support.
We're binding it to our local IPv4 wildcard (the port defaults to 5061 for TLS).
We've set the TLS certificate file to the one we created above.
We've set the Certificate Authority to the one we created above.
TLS Ciphers have been set to ALL, since it's the most permissive.
And we've set the TLS client method to TLSv1, since that's the preferred one for RFCs and for most clients.

Configuring a TLS-enabled SIP peer within Asterisk

Next, you'll need to configure a SIP peer within Asterisk to use TLS as a transport type. Here's an example:

```
[malcolm]
type=peer
secret=malcolm ;note that this is NOT a secure password
host=dynamic
context=local
dtmfmode=rfc2833
disallow=all
allow=g722
transport=tls
context=local
```

Notice the **transport** option. The Asterisk SIP channel driver supports three types: udp, tcp and tls. Since we're configuring for TLS, we'll set that. It's also possible to list several supported transport types for the peer by separating them with commas.

Configuring a TLS-enabled SIP client to talk to Asterisk

Next, we'll configure Blink.

First, let's add a new account.
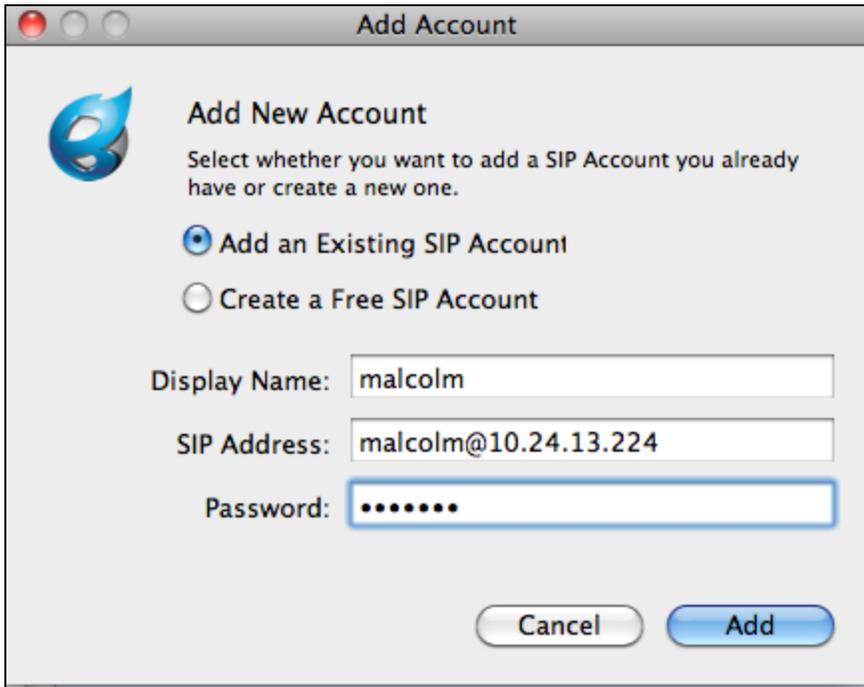
Then, we need to modify the Account Preferences, and under the SIP Settings, we need to set the outbound proxy to connect to the TLS port and transport type on our Asterisk server. In this case, there's an Asterisk server running on port 5061 on host 10.24.13.233.
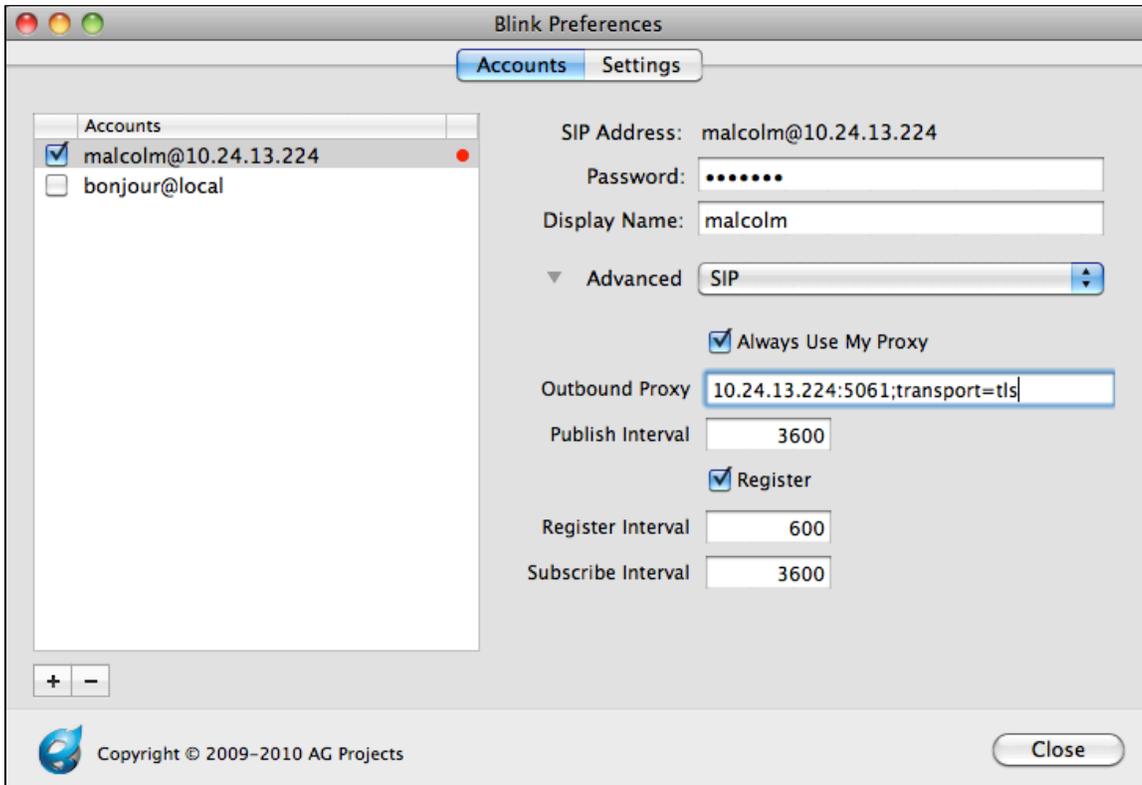
Now, we need to point the TLS account settings to the client certificate (malcolm.pem) that we copied to our computer.
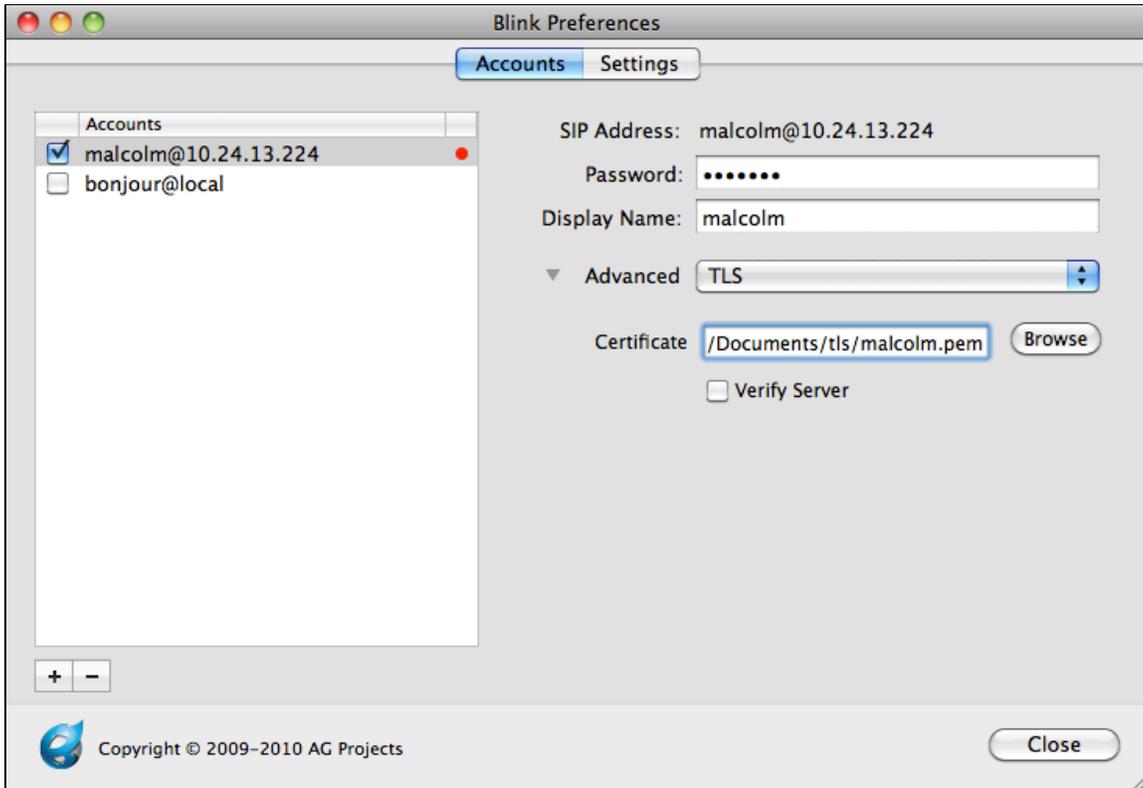
Then, we'll point the TLS server settings to the ca.crt file that we copied to our computer.
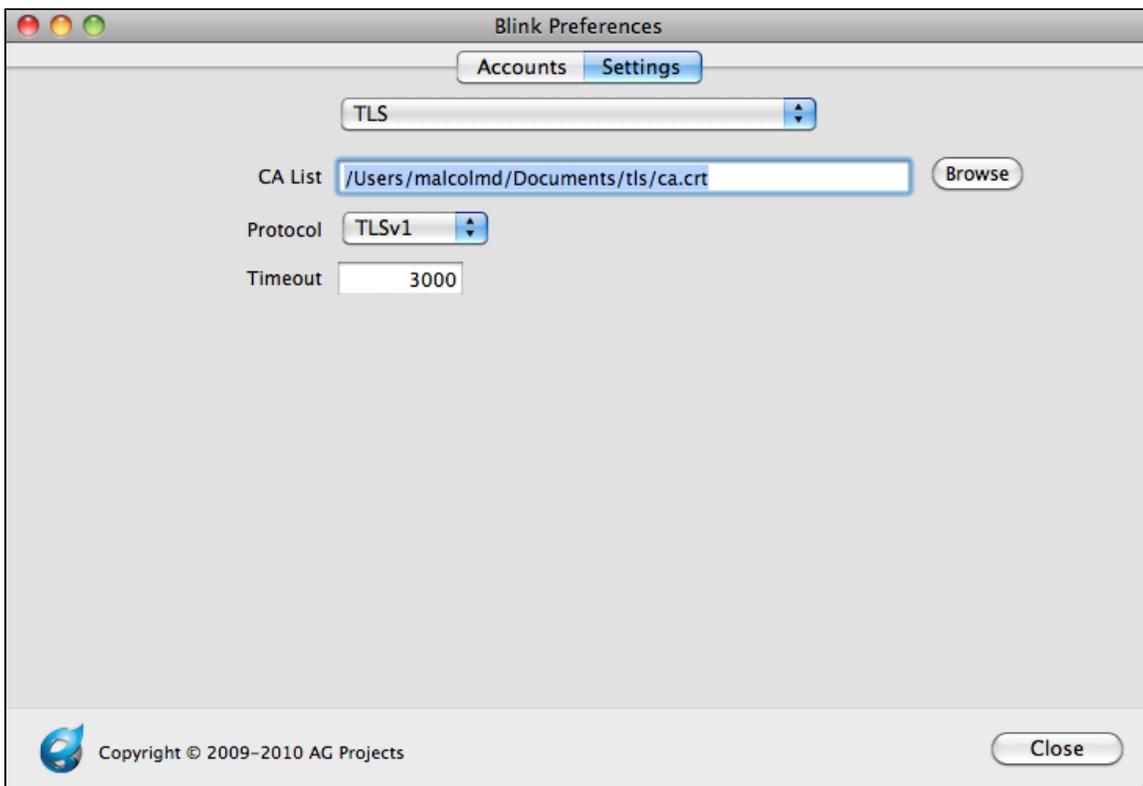
Press "close," and you should see Blink having successfully registered to Asterisk.
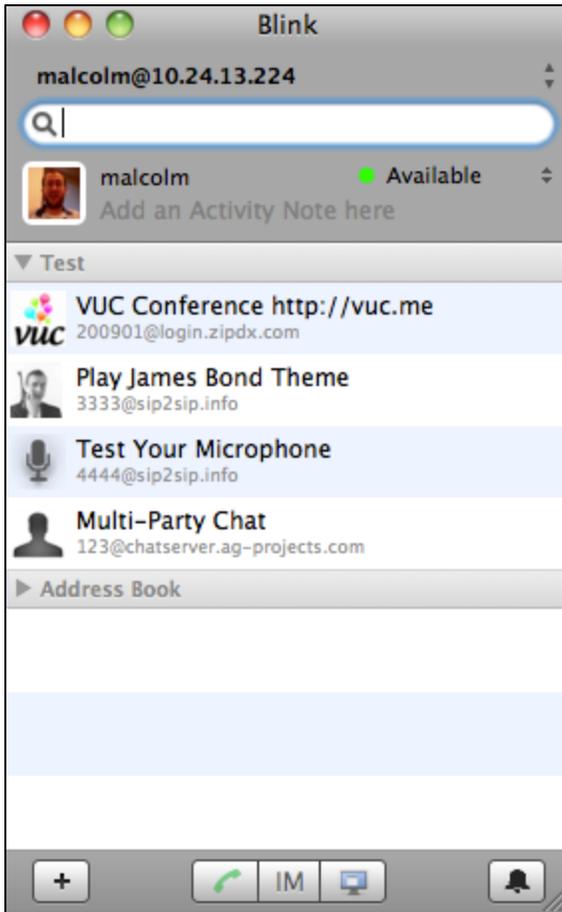
Depending on your Asterisk CLI logging levels, you should see something like:

```
-- Registered SIP 'malcolm' at 10.24.250.178:5061
    > Saved useragent "Blink 0.22.2 (MacOSX)" for peer malcolm
```

Notice that we registered on port 5061, the TLS port.

Now, make a call. You should see a small secure lockbox in your Blink calling window to indicate that the call was made using secure (TLS) signaling:
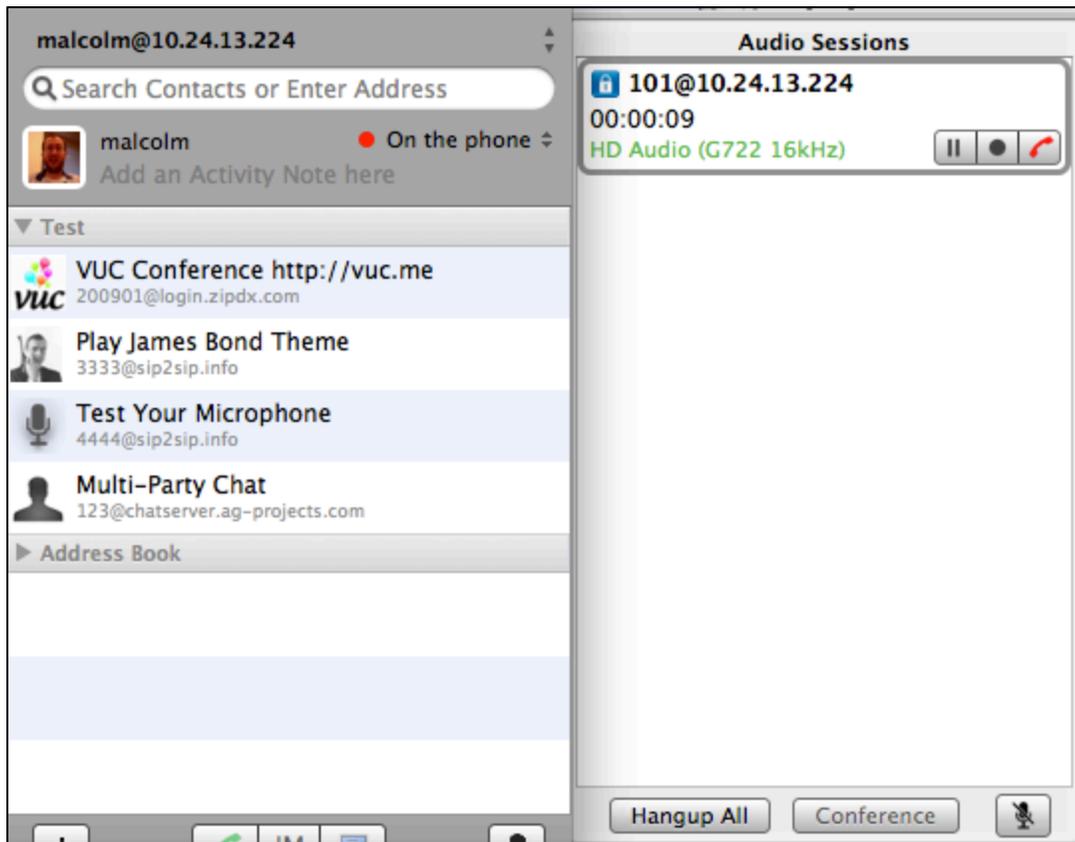
Part 2 (SRTP)

Now that we've got TLS enabled, our signaling is secure - so no one knows what extensions on the PBX we're dialing. But, our media is still not secure - so someone can snoop our RTP conversations from the wire. Let's fix that.

SRTP support is provided by libsrtp. libsrtp has to be installed on the machine before Asterisk is compiled, otherwise you're going to see something like:

```
[Jan 24 09:29:16] ERROR[10167]: chan_sip.c:27987 setup_srtp: No SRTP
module loaded, can't setup SRTP session.
```

on your Asterisk CLI. If you do see that, install libsrtp (and the development headers), and then reinstall Asterisk (./configure; make; make install).

With that complete, let's first go back into our peer definition in **sip.conf.** We're going to add a new encryption line, like:

```
[malcolm]
type=peer
secret=malcolm ;note that this is NOT a secure password
host=dynamic
context=local
dtmfmode=rfc2833
disallow=all
allow=g722
transport=tls
encryption=yes
context=local
```

Next, we'll set Blink to use SRTP:
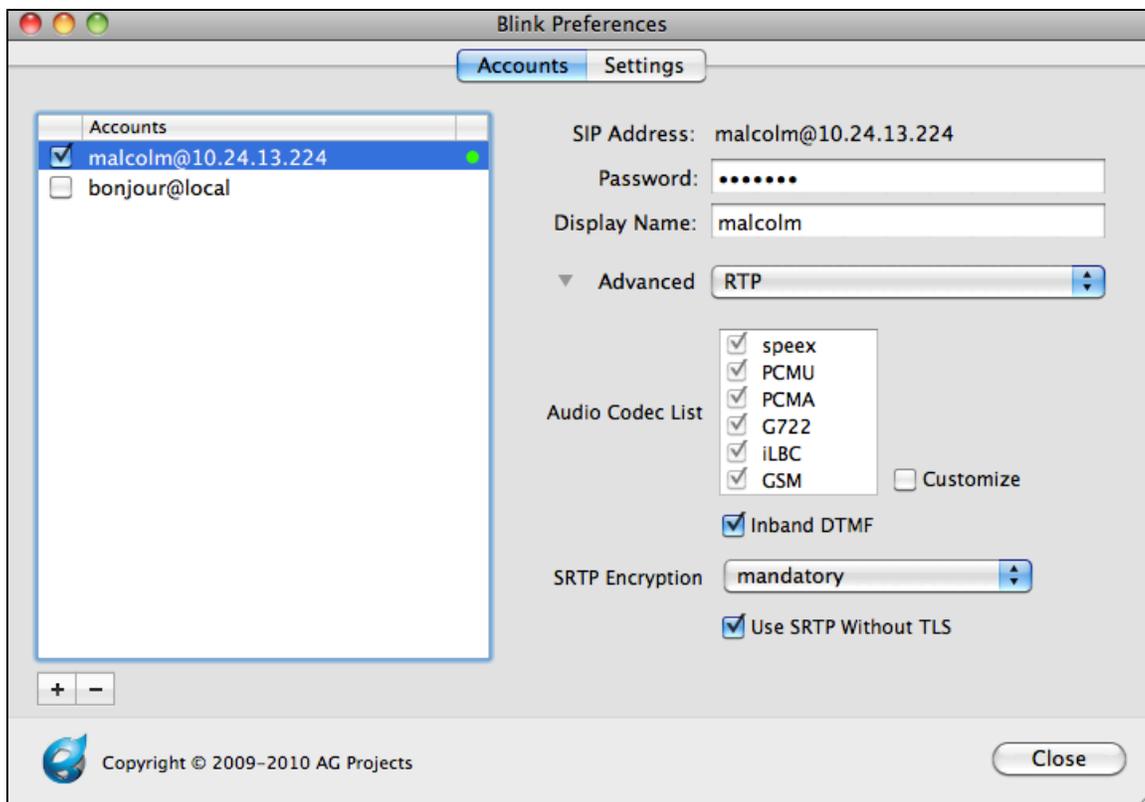
Reload Asterisk's SIP configuration (sip reload), make a call, and voilà:
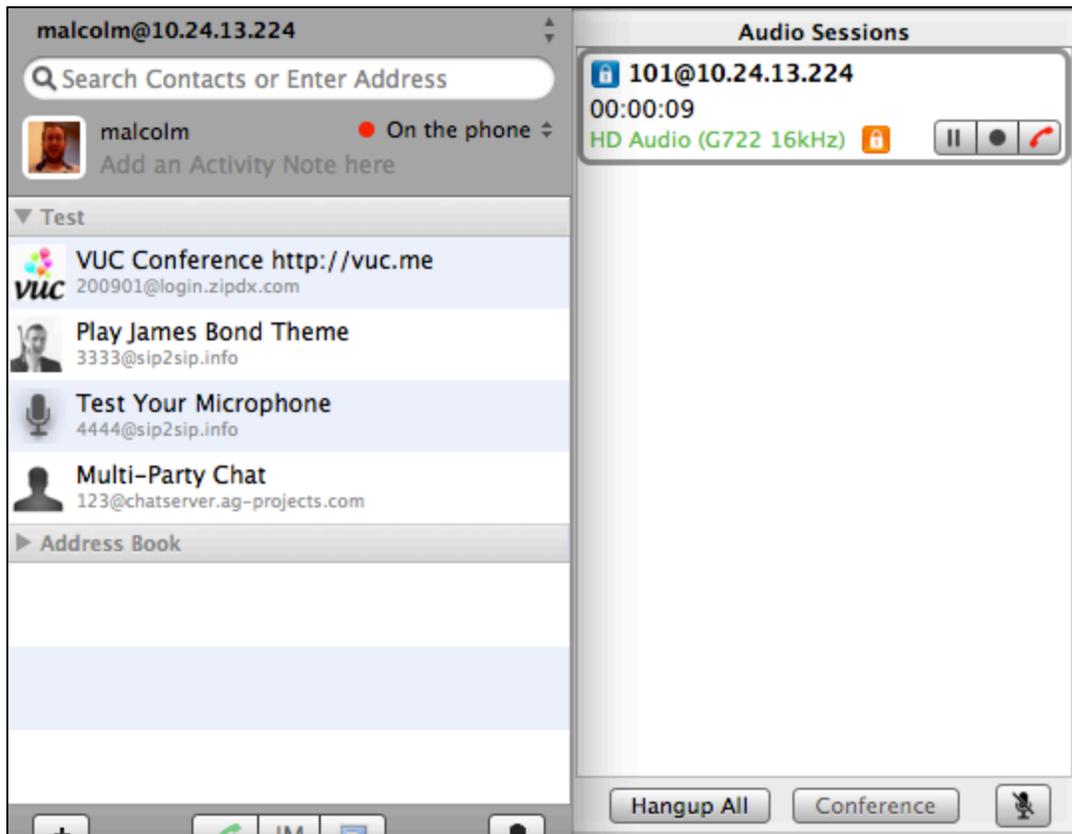
We're making secure calls with TLS (signaling) and SRTP (media).

Shared Line Appearances (SLA)

What are shared line appearances and how do they work in Asterisk? Read on...

Introduction to Shared Line Appearances (SLA)

The "SLA" functionality in Asterisk is intended to allow a setup that emulates a simple key system. It uses the various abstraction layers already built into Asterisk to emulate key system functionality across various devices, including IP channels.

SLA Configuration

How-to configure SLA in Asterisk

SLA Configuration Summary

An SLA system is built up of virtual trunks and stations mapped to real Asterisk devices. The configuration for all of this is done in three different files: extensions.conf, sla.conf, and the channel specific configuration file such as sip.conf or dahdi.conf.

SLA Dialplan Configuration

The SLA implementation can automatically generate the dialplan necessary for basic operation if the "autocontext" option is set for trunks and stations in sla.conf. However, for reference, here is

an automatically generated dialplan to help with custom building of the dialplan to include other features, such as voicemail.

However, note that there is a little bit of additional configuration needed if the trunk is an IP channel. This is discussed in the section on Trunks.

There are extensions for incoming calls on a specific trunk, which execute the SLATrunk application, as well as incoming calls from a station, which execute SLAStation. Note that there are multiple extensions for incoming calls from a station. This is because the SLA system has to know whether the phone just went off hook, or if the user pressed a specific line button.

Also note that there is a hint for every line on every station. This lets the SLA system control each individual light on every phone to ensure that it shows the correct state of the line. The phones must subscribe to the state of each of their line appearances.
Please refer to the examples section for full dialplan samples for SLA.

SLA Trunk Configuration

An SLA trunk is a mapping between a virtual trunk and a real Asterisk device. This device may be an analog FXO line, or something like a SIP trunk. A trunk must be configured in two places. First, configure the device itself in the channel specific configuration file such as dahdi.conf or sip.conf. Once the trunk is configured, then map it to an SLA trunk in sla.conf.

```
[line1]
type=trunk
device=DAHDI/1
```

Be sure to configure the trunk's context to be the same one that is set for the "autocontext" option in sla.conf if automatic dialplan configuration is used. This would be done in the regular device entry in dahdi.conf, sip.conf, etc. Note that the automatic dialplan generation creates the SLATrunk() extension at extension 's'. This is perfect for DAHDI channels that are FXO trunks, for example. However, it may not be good enough for an IP trunk, since the call coming in over the trunk may specify an actual number.

If the dialplan is being built manually, ensure that calls coming in on a trunk execute the SLATrunk() application with an argument of the trunk name, as shown in the dialplan example before.

IP trunks can be used, but they require some additional configuration to work.

For this example, let's say we have a SIP trunk called "mytrunk" that is going to be used as line4. Furthermore, when calls come in on this trunk, they are going to say that they are calling the number "12564286000". Also, let's say that the numbers that are valid for calling out this trunk are NANP numbers, of the form _1NXXNXXXXXX.

In sip.conf, there would be an entry for [mytrunk]. For [mytrunk], set context=line4.

```
[line4]
type=trunk
device=Local/disa@line4_outbound

[line4]
exten => 12564286000,1,SLATrunk(line4)

[line4_outbound]
exten => disa,1,Disa(no-password,line4_outbound)
exten => _1NXXNXXXXXX,1,Dial(SIP/${EXTEN}@mytrunk)
```

So, when a station picks up their phone and connects to line 4, they are connected to the local dialplan. The Disa application plays dialtone to the phone and collects digits until it matches an extension. In this case, once the phone dials a number like 12565551212, the call will proceed out the SIP trunk.

SLA Station Configuration

An SLA station is a mapping between a virtual station and a real Asterisk device. Currently, the only channel driver that has all of the features necessary to support an SLA environment is chan_sip. So, to configure a SIP phone to use as a station, you must configure sla.conf and sip.conf.

```
[station1]
type=station
device=SIP/station1
trunk=line1
trunk=line2
```

Here are some hints on configuring a SIP phone for use with SLA:

- Add the SIP channel as a [station] in sla.conf.

- Configure the phone in sip.conf. If automatic dialplan configuration was used by enabling the "autocontext" option in sla.conf, then this entry in sip.conf should have the same context setting.
- On the phone itself, there are various things that must be configured to make everything work correctly. Let's say this phone is called "station1" in sla.conf, and it uses trunks named "line1" and line2".
  - Two line buttons must be configured to subscribe to the state of the following extensions: - station1_line1 - station1_line2
  - The line appearance buttons should be configured to dial the extensions that they are subscribed to when they are pressed.
  - If you would like the phone to automatically connect to a trunk when it is taken off hook, then the phone should be automatically configured to dial "station1" when it is taken off hook.

SLA Configuration Examples

Example configurations for SLA

Basic SLA Configuration Example

This is an example of the most basic SLA setup. It uses the automatic dialplan generation so the

configuration is minimal.
sla.conf:

```
[line1]
type=trunk
device=DAHDI/1
autocontext=line1

[line2]
type=trunk
device=DAHDI/2
autocontext=line2

[station]
type=station
trunk=line1
trunk=line2
autocontext=sla_stations

[station1](station)
device=SIP/station1

[station2](station)
device=SIP/station2

[station3](station)
device=SIP/station3
```

With this configuration, the dialplan is generated automatically. The first DAHDI channel should have its context set to "line1" and the second should be set to "line2" in dahdi.conf. In sip.conf, station1, station2, and station3 should all have their context set to "sla_stations".
For reference, here is the automatically generated dialplan for this situation:

```
[line1]
exten => s,1,SLATrunk(line1)

[line2]
exten => s,2,SLATrunk(line2)

[sla_stations]
exten => station1,1,SLAStation(station1)
exten => station1_line1,hint,SLA:station1_line1
exten => station1_line1,1,SLAStation(station1_line1)
exten => station1_line2,hint,SLA:station1_line2
exten => station1_line2,1,SLAStation(station1_line2)
exten => station2,1,SLAStation(station2)
exten => station2_line1,hint,SLA:station2_line1
exten => station2_line1,1,SLAStation(station2_line1)
exten => station2_line2,hint,SLA:station2_line2
exten => station2_line2,1,SLAStation(station2_line2)
exten => station3,1,SLAStation(station3)
exten => station3_line1,hint,SLA:station3_line1
exten => station3_line1,1,SLAStation(station3_line1)
exten => station3_line2,hint,SLA:station3_line2
exten => station3_line2,1,SLAStation(station3_line2)
```

SLA and Voicemail Example

This is an example of how you could set up a single voicemail box for the phone system. The voicemail box number used in this example is 1234, which would be configured in voicemail.conf.

For this example, assume that there are 2 trunks and 3 stations. The trunks are DAHDI/1 and DAHDI/2. The stations are SIP/station1, SIP/station2, and SIP/station3.

In dahdi.conf, channel 1 has context=line1 and channel 2 has context=line2.

In sip.conf, all three stations are configured with context=sla_stations.

When the stations pick up their phones to dial, they are allowed to dial NANP numbers for outbound calls, or 8500 for checking voicemail.

sla.conf:

```
[line1]
type=trunk
device=Local/disa@line1_outbound

[line2]
type=trunk
device=Local/disa@line2_outbound

[station]
type=station
trunk=line1
trunk=line2

[station1](station)
device=SIP/station1

[station2](station)
device=SIP/station2

[station3](station)
device=SIP/station3
```

extensions.conf:

```
[macro-slaline]
exten => s,1,SLATrunk(${ARG1})
exten => s,n,Goto(s-${SLATRUNK_STATUS},1)
exten => s-FAILURE,1,Voicemail(1234,u)
exten => s-UNANSWERED,1,Voicemail(1234,u)

[line1]
exten => s,1,Macro(slaline,line1)

[line2]
exten => s,2,Macro(slaline,line2)

[line1_outbound]
exten => disa,1,Disa(no-password,line1_outbound)
exten => _1NXXNXXXXXX,1,Dial(DAHDI/1/${EXTEN})
exten => 8500,1,VoicemailMain(1234)

[line2_outbound]
exten => disa,1,Disa(no-password|line2_outbound)
exten => _1NXXNXXXXXX,1,Dial(DAHDI/2/${EXTEN})
exten => 8500,1,VoicemailMain(1234)

[sla_stations]
exten => station1,1,SLAStation(station1)
exten => station1_line1,hint,SLA:station1_line1
exten => station1_line1,1,SLAStation(station1_line1)
exten => station1_line2,hint,SLA:station1_line2
exten => station1_line2,1,SLAStation(station1_line2)
exten => station2,1,SLAStation(station2)
exten => station2_line1,hint,SLA:station2_line1
exten => station2_line1,1,SLAStation(station2_line1)
exten => station2_line2,hint,SLA:station2_line2
exten => station2_line2,1,SLAStation(station2_line2)
exten => station3,1,SLAStation(station3)
exten => station3_line1,hint,SLA:station3_line1
exten => station3_line1,1,SLAStation(station3_line1)
exten => station3_line2,hint,SLA:station3_line2
exten => station3_line2,1,SLAStation(station3_line2)
```

SLA and Call Handling

Read about call handling related to SLA

SLA Call Handling Summary

This section is intended to describe how Asterisk handles calls inside of the SLA system so that it is clear what behavior is expected.

SLA Station goes off hook (not ringing)

When a station goes off hook, it should initiate a call to Asterisk with the extension that indicates that the phone went off hook without specifying a specific line. In the examples in this document, for the station named "station1", this extension is simply named, "station1".

Asterisk will attempt to connect this station to the first available trunk that is not in use. Asterisk will check the trunks in the order that they were specified in the station entry in sla.conf. If all trunks are in use, the call will be denied.

If Asterisk is able to acquire an idle trunk for this station, then trunk is connected to the station and the station will hear dialtone. The station can then proceed to dial a number to call. As soon as a trunk is acquired, all appearances of this line on stations will show that the line is in use.

SLA Station goes off hook (ringing)

When a station goes off hook while it is ringing, it should simply answer the call that had been initiated to it to make it ring. Once the station has answered, Asterisk will figure out which trunk to connect it to. It will connect it to the highest priority trunk that is currently ringing. Trunk priority is determined by the order that the trunks are listed in the station entry in sla.conf.

SLA Line button on a station is pressed

When a line button is pressed on a station, the station should initiate a call to Asterisk with the extension that indicates which line button was pressed. In the examples given in this document, for a station named "station1" and a trunk named "line1", the extension would be "station1_line1".
If the specified trunk is not in use, then the station will be connected to it and will hear dialtone. All appearances of this trunk will then show that it is now in use.

If the specified trunk is on hold by this station, then this station will be reconnected to the trunk. The line appearance for this trunk on this station will now show in use. If this was the only station that had the call on hold, then all appearances of this trunk will now show that it is in use. Otherwise, all stations that are not currently connected to this trunk will show it on hold.

If the specified trunk is on hold by a different station, then this station will be connected to the trunk only if the trunk itself and the station(s) that have it on hold do not have private hold enabled. If connected, the appeareance of this trunk on this station will then show in use. All stations that are not currently connected to this trunk will show it on hold.

Short Message Service (SMS)

Information about Asterisk and SMS

Introduction to SMS

The SMS module for Asterisk was developed by Adrian Kennard, and is an implementation of the

ETSI specification for landline SMS, ETSI ES 201 912, which is available from
http://www.etsi.org. Landline SMS is starting to be available in various parts of Europe, and is
available from BT in the UK. However, Asterisk would allow gateways to be created in other
locations such as the US, and use of SMS capable phones such as the Magic Messenger. SMS
works using analogue or ISDN lines.

SMS and extensions.conf

The following contexts are recommended.

```
; Mobile Terminated, RX. This is used when an incoming call from the
SMS arrives
; with the queue (called number and sub address) in ${EXTEN}
; Running an app after receipt of the text allows the app to find
all messages
; in the queue and handle them, e.g. email them.
; The app may be something like smsq --process=somecommand
--queue=${EXTEN} to
; run a command for each received message
; See below for usage
[smsmtrx]
exten = _X.,1,SMS(${EXTEN},a)
exten = _X.,2,System("someapptohandleincomingsms ${EXTEN}")
exten = _X.,3,Hangup
;
; Mobile originated, RX. This is receiving a message from a device,
e.g.
; a Magic Messenger on a sip extension
; Running an app after receipt of the text allows the app to find
all messages
; in the queue and handle then, e.g. sending them to the public SMSC

; The app may be something like smsq --process=somecommand
--queue=${EXTEN}
; to run a command for each received message
; See below for example usage
[smsmorx]
exten = _X.,1,SMS(${EXTEN},sa)
exten = _X.,2,System("someapptohandlelocalsms ${EXTEN}")
exten = _X.,3,Hangup
```

smsmtrx is normally accessed by an incoming call from the SMSC. In the UK this call is from a
CLI of 080058752X0 where X is the sub address. As such a typical usage in the extensions.conf
at the point of handling an incoming call is:

```
exten = _X./8005875290,1,Goto(smsmtrx,${EXTEN},1)
exten =
_X./_80058752[0-8]0,1,Goto(smsmtrx,${EXTEN}-${CALLERID(num):8:1},1)
```

Alternatively, if you have the correct national prefix on incoming CLI, e.g. using dahdi_hfc, you might use:

```
exten = _X./08005875290,1,Goto(smsmtrx,${EXTEN},1)
exten =
_X./_080058752[0-8]0,1,Goto(smsmtrx,${EXTEN}-${CALLERID(num):9:1},1)
```

smsmorx is normally accessed by a call from a local sip device connected to a Magic Messenger. It could however by that you are operating Asterisk as a message centre for calls from outside. Either way, you look at the called number and goto smsmorx. In the UK, the SMSC number that would be dialed is 1709400X where X is the caller sub address. As such typical usage in extension.config at the point of handling a call from a sip phone is:

```
exten = 17094009,1,Goto(smsmorx,${CALLERID(num)},1)
exten = _1709400[0-8],1,Goto(smsmorx,${CALLERID(num)}-{EXTEN:7:1},1)
```

SMS Background

Short Message Service (SMS), or texting is very popular between mobile phones. A message can be sent between two phones, and normally contains 160 characters. There are ways in which various types of data can be encoded in a text message such as ring tones, and small graphic, etc. Text messaging is being used for voting and competitions, and also SPAM...

Sending a message involves the mobile phone contacting a message centre (SMSC) and passing the message to it. The message centre then contacts the destination mobile to deliver the message. The SMSC is responsible for storing the message and trying to send it until the destination mobile is available, or a timeout.

Landline SMS works in basically the same way. You would normally have a suitable text capable landline phone, or a separate texting box such as a Magic Messenger on your phone line. This sends a message to a message centre your telco provides by making a normal call and sending the data using 1200 Baud FSK signaling according to the ETSI spec. To receive a message the message centre calls the line with a specific calling number, and the text capable phone answers the call and receives the data using 1200 Baud FSK signaling. This works particularly well in the UK as the calling line identity is sent before the first ring, so no phones in the house would ring when a message arrives.

SMS Delivery Reports

The SMS specification allows for delivery reports. These are requested using the srr bit. However, as these do not work in the UK yet they are not fully implemented in this application. If

anyone has a telco that does implement these, please let me know. BT in the UK have a non standard way to do this by starting the message with *0#, and so this application may have a UK specific bodge in the near future to handle these.
The main changes that are proposed for delivery report handling are :

- New queues for sent messages, one file for each destination address and message reference.
- New field in message format, user reference, allowing applications to tie up their original message with a report.
- Handling of the delivery confirmation/rejection and connecting to the outgoing message - the received message file would then have fields for the original outgoing message and user reference allowing applications to handle confirmations better.

SMS File Formats

By default all queues are held in a director /var/spool/asterisk/sms. Within this directory are sub directories mtrx, mttx, morx, motx which hold the received messages and the messages ready to send. Also, /var/log/asterisk/sms is a log file of all messages handled.

The file name in each queue directory starts with the queue parameter to SMS which is normally the CLI used for an outgoing message or the called number on an incoming message, and may have -X (X being sub address) appended. If no queue ID is known, then 0 is used by smsq by default. After this is a dot, and then any text. Files are scanned for matching queue ID and a dot at the start. This means temporary files being created can be given a different name not starting with a queue (we recommend a . on the start of the file name for temp files). Files in these queues are in the form of a simple text file where each line starts with a keyword and an = and then data. udh and ud have options for hex encoding, see below.

UTF-8.

The user data (ud) field is treated as being UTF-8 encoded unless the DCS is specified indicating 8 bit format. If 8 bit format is specified then the user data is sent as is. The keywords are as follows:

- oa - Originating address The phone number from which the message came Present on mobile terminated messages and is the CLI for morx messages
- da - Destination Address The phone number to which the message is sent Present on mobile originated messages
- scts - The service centre time stamp Format YYYY-MM-DDTHH:MM:SS Present on mobile terminated messages
- pid - One byte decimal protocol ID See GSM specs for more details Normally 0 or absent
- dcs - One byte decimal data coding scheme If omitted, a sensible default is used (see below) See GSM specs for more details
- mr - One byte decimal message reference Present on mobile originated messages, added by default if absent
- srr - 0 or 1 for status report request Does not work in UK yet, not implemented in app_sms yet
- rp - 0 or 1 return path See GSM specs for details
- vp - Validity period in seconds Does not work in UK yet
- udh - Hex string of user data header prepended to the SMS contents, excluding initial length byte. Consistent with ud, this is specified as udh# rather than udh= If blank, this means that the udhi flag will be set but any user data header must be in the ud field
- ud - User data, may be text, or hex, see below

udh is specified as as udh# followed by hex (2 hex digits per byte). If present, then the user data header indicator bit is set, and the length plus the user data header is added to the start of the user data, with padding if necessary (to septet boundary in 7 bit format). User data can hold an USC character codes U+0000 to U+FFFF. Any other characters are coded as U+FEFF

ud can be specified as ud= followed by UTF-8 encoded text if it contains no control characters, i.e. only (U+0020 to U+FFFF). Any invalid UTF-8 sequences are treated as is (U+0080-U+00FF).

ud can also be specified as ud# followed by hex (2 hex digits per byte) containing characters U+0000 to U+00FF only.

ud can also be specified as ud## followed by hex (4 hex digits per byte) containing UCS-2 characters.

When written by app_sms (e.g. incoming messages), the file is written with ud= if it can be (no control characters). If it cannot, the a comment line ;ud= is used to show the user data for human readability and ud# or ud## is used.

SMS Sub Address

When sending a message to a landline, you simply send to the landline number. In the UK, all of the mobile operators (bar one) understand sending messages to landlines and pass the messages to the BTText system for delivery to the landline.

The specification for landline SMS allows for the possibility of more than one device on a single landline. These can be configured with Sub addresses which are a single digit. To send a message to a specific device the message is sent to the landline number with an extra digit appended to the end. The telco can define a default sub address (9 in the UK) which is used when the extra digit is not appended to the end. When the call comes in, part of the calling line ID is the sub address, so that only one device on the line answers the call and receives the message.

Sub addresses also work for outgoing messages. Part of the number called by the device to send a message is its sub address. Sending from the default sub address (9 in the UK) means the message is delivered with the sender being the normal landline number. Sending from any other sub address makes the sender the landline number with an extra digit on the end.

Using Asterisk, you can make use of the sub addresses for sending and receiving messages. Using DDI (DID, i.e. multiple numbers on the line on ISDN) you can also make use of many different numbers for SMS.

SMS Terminology

- SMS - Short Message Service i.e. text messages
- SMSC - Short Message Service Centre The system responsible for storing and forwarding messages
- MO - Mobile Originated A message on its way from a mobile or landline device to the SMSC
- MT - Mobile Terminated A message on its way from the SMSC to the mobile or landline device
- RX - Receive A message coming in to the Asterisk box
- TX - Transmit A message going out of the Asterisk box

SMS Typical Use with Asterisk

Sending messages from an Asterisk box can be used for a variety of reasons, including notification from any monitoring systems, email subject lines, etc.

Receiving messages to an Asterisk box is typically used just to email the messages to someone appropriate - we email and texts that are received to our direct numbers to the appropriate person. Received messages could also be used to control applications, manage competitions, votes, post items to IRC, anything.

Using a terminal such as a magic messenger, an Asterisk box could ask as a message centre sending messages to the terminal, which will beep and pop up the message (and remember 100 or so messages in its memory).

Using SMSq

smsq is a simple helper application designed to make it easy to send messages from a command line. it is intended to run on the Asterisk box and have direct access to the queue directories for SMS and for Asterisk.

In its simplest form you can send an SMS by a command such as smsq 0123456789 This is a test to 0123456789 This would create a queue file for a mobile originated TX message in queue 0 to send the text "This is a test to 0123456789" to 0123456789. It would then place a file in the /var/spool/asterisk/outgoing directory to initiate a call to 17094009 (the default message centre in smsq) attached to application SMS with argument of the queue name (0).

Normally smsq will queue a message ready to send, and will then create a file in the Asterisk outgoing directory causing Asterisk to actually connect to the message centre or device and actually send the pending message(s).

Using --process, smsq can however be used on received queues to run a command for each file (matching the queue if specified) with various environment variables set based on the message (see below); smsq options:

- --help Show help text
- --usage Show usage
- --queue -q Specify a specific queue In no specified, messages are queued under queue "0"
- --da -d Specify destination address
- --oa -o Specify originating address This also implies that we are generating a mobile terminated message
- --ud -m Specify the actual message
- --ud-file -f Specify a file to be read for the context of the message A blank filename (e.g. --ud-file= on its own) means read stdin. Very useful when using via ssh where command line parsing could mess up the message.
- --mt -t Mobile terminated message to be generated
- --mo Mobile originated message to be generated Default
- --tx Transmit message Default
- --rx -r Generate a message in the receive queue
- --UTF-8 Treat the file as UTF-8 encoded (default)
- --UCS-1 Treat the file as raw 8 bit UCS-1 data, not UTF-8 encoded
- --UCS-2 Treat the file as raw 16 bit bigendian USC-2 data
- --process Specific a command to process for each file in the queue Implies --rx and --mt if not otherwise specified. Sets environment variables for every possible variable, and also ud, ud8 (USC-1 hex), and ud16 (USC-2 hex) for each call. Removes files.
- --motx-channel Specify the channel for motx calls May contain X to use sub address based on queue name or may be full number Default is Local/1709400X
- --motx-callerid Specify the caller ID for motx calls The default is the queue name without -X suffix
- --motx-wait Wait time for motx call Default 10
- --motx-delay Retry time for motx call Default 1
- --motx-retries Retries for motx call Default 10
- --mttx-channel Specify the channel for mttx calls Default is Local/ and the queue name without -X suffix
- --mtttx-callerid Specify the callerid for mttx calls May include X to use sub address based on queue name or may be full number Default is 080058752X0
- --mttx-wait Wait time for mttx call Default 10
- --mttx-delay Retry time for mttx call Default 30
- --mttx-retries Retries for mttx call Default 100
- --default-sub-address The default sub address assumed (e.g. for X in CLI and dialled numbers as above) when none added (-X) to queue Default 9
- --no-dial -x Create queue, but do not dial to send message
- --no-wait Do not wait if a call appears to be in progress This could have a small window where a message is queued but not sent, so regular calls to smsq should be done to pick up any missed messages
- --concurrent How many concurrent calls to allow (per queue), default 1
- --mr -n Message reference
- --pid -p Protocol ID
- --dcs Data coding scheme
- --udh Specific hex string of user data header specified (not including the initial length byte) May be a blank string to indicate header is included in the user data already but user data header indication to be set.
- --srr Status report requested
- --rp Return path requested
- --vp Specify validity period (seconds)
- --scts Specify timestamp (YYYY-MM-DDTHH:MM:SS)

- --spool-dir Spool dir (in which sms and outgoing are found) Default /var/spool/asterisk

Other arguments starting '' or '' are invalid and will cause an error. Any trailing arguments are processed as follows:

- If the message is mobile originating and no destination address has been specified, then the first argument is assumed to be a destination address
- If the message is mobile terminating and no destination address has been specified, then the first argument is assumed to be the queue name
- If there is no user data, or user data file specified, then any following arguments are assumed to be the message, which are concatenated.
- If no user data is specified, then no message is sent. However, unless --no-dial is specified, smsq checks for pending messages and generates an outgoing anyway

> ⚠ When smsq attempts to make a file in /var/spool/asterisk/outgoing, it checks if there is already a call queued for that queue. It will try several filenames, up to the --concurrent setting. If these files exist, then this means Asterisk is already queued to send all messages for that queue, and so Asterisk should pick up the message just queued. However, this alone could create a race condition, so if the files exist then smsq will wait up to 3 seconds to confirm it still exists or if the queued messages have been sent already. The --no-wait turns off this behaviour. Basically, this means that if you have a lot of messages to send all at once, Asterisk will not make unlimited concurrent calls to the same message centre or device for the same queue. This is because it is generally more efficient to make one call and send all of the messages one after the other.

smsq can be used with no arguments, or with a queue name only, and it will check for any pending messages and cause an outgoing if there are any. It only sets up one outgoing call at a time based on the first queued message it finds. A outgoing call will normally send all queued messages for that queue. One way to use smsq would be to run with no queue name (so any queue) every minute or every few seconds to send pending message. This is not normally necessary unless --no-dial is selected. Note that smsq does only check motx or mttx depending on the options selected, so it would need to be called twice as a general check.

UTF-8 is used to parse command line arguments for user data, and is the default when reading a file. If an invalid UTF-8 sequence is found, it is treated as UCS-1 data (i.e, as is). The --process option causes smsq to scan the specified queue (default is mtrx) for messages (matching the queue specified, or any if queue not specified) and run a command and delete the file. The command is run with a number of environment variables set as follows. Note that these are unset if not needed and not just taken from the calling environment. This allows simple processing of incoming messages

- - $queue Set if a queue specified $?srr srr is set (to blank) if srr defined and has value 1. $?rp rp is set (to blank) if rp defined and has value 1. $ud User data, UTF-8 encoding, including any control characters, but with nulls stripped out Useful for the content of emails, for example, as it includes any newlines, etc. $ude User data, escaped UTF-8, including all characters, but control characters \n, \r, \t, \f, \xxx and \ is escaped as

Useful guaranteed one line printable text, so useful in Subject lines of emails, etc $ud8 Hex UCS-1 coding of user data (2 hex digits per character) Present only if all user data is in range U+0000 to U+00FF $ud16 Hex UCS-2 coding of user data (4 hex digits per character) other Other fields set using their field name, e.g. mr, pid, dcs, etc. udh is a hex byte string

Voicemail

All things voicemail

ODBC Voicemail Storage

ODBC Storage allows you to store voicemail messages within a database instead of using a file.

This is not a full realtime engine and only supports ODBC. The table description for the voicemessages table is as follows:

Field	Type	Null	Key	Default	Extra
msgnum	int(11)	Yes		NULL	
dir	varchar(80)	Yes	MUL		NULL
context	varchar(80)	Yes		NULL	
macrocontext	varchar(80)	Yes		NULL	
callerid	varchar(40)	Yes		NULL	
origtime	varchar(40)	Yes		NULL	
duration	varchar(20)	Yes		NULL	
flag	varchar(8)	Yes		NULL	
mailboxuser	varchar(80)	Yes		NULL	
mailboxcontext	varchar(80)	Yes		NULL	
recording	longblob	Yes		NULL	

The database name (from /etc/asterisk/res_odbc.conf) is in the odbcstorage variable in the general section of voicemail.conf.

You may modify the voicemessages table name by using odbctable=table_name in voicemail.conf.

IMAP Voicemail Storage

By enabling IMAP Storage, Asterisk will use native IMAP as the storage mechanism for voicemail messages instead of using the standard file structure.

Tighter integration of Asterisk voicemail and IMAP email services allows additional voicemail functionality, including:

- Listening to a voicemail on the phone will set its state to "read" in a user's mailbox automatically.
- Deleting a voicemail on the phone will delete it from the user's mailbox automatically.
- Accessing a voicemail recording email message will turn off the message waiting indicator (MWI) on the user's phone.
- Deleting a voicemail recording email will also turn off the message waiting indicator, and delete the message from the voicemail system.

IMAP VM Storage Installation Notes

University of Washington IMAP C-Client

If you do not have the University of Washington's IMAP c-client installed on your system, you will need to download the c-client source distribution (http://www.washington.edu/imap/) and compile it. Asterisk supports the 2007 version of c-client as there appears to be issues with older versions which cause Asterisk to crash in certain scenarios. It is highly recommended that you utilize a current version of the c-client libraries. Additionally, mail_expunge_full is enabled in the 2006 and later versions.

Note that Asterisk only uses the 'c-client' portion of the UW IMAP toolkit, but building it also builds an IMAP server and various other utilities. Because of this, the build instructions for the

IMAP toolkit are somewhat complicated and can lead to confusion about what is needed.
If you are going to be connecting Asterisk to an existing IMAP server, then you don't need to care about the server or utilities in the IMAP toolkit at all. If you want to also install the UW IMAPD server, that is outside the scope of this document.

Building the c-client library is fairly straightforward; for example, on a Debian system there are two possibilities:

If you will not be using SSL to connect to the IMAP server:

```
$ make slx SSLTYPE=none
```

If you will be using SSL to connect to the IMAP server:

```
$ make slx EXTRACFLAGS="-I/usr/include/openssl"
```

Additionally, you may wish to build on a 64-bit machine, in which case you need to add -fPIC to EXTRACFLAGS. So, building on a 64-bit machine with SSL support would look something like:

```
$ make slx EXTRACFLAGS="-fPIC -I/usr/include/openssl"
```

Or without SSL support:

```
$ make slx SSLTYPE=none EXTRACFLAGS=-fPIC
```

Once this completes you can proceed with the Asterisk build; there is no need to run 'make install'.

Compiling Asterisk

Configure with ./configure -with-imap=/usr/src/imap or wherever you built the UWashington IMAP Toolkit. This directory will be searched for a source installation. If no source installation is found there, then a package installation of the IMAP c-client will be searched for in this directory. If one is not found, then configure will fail.

A second configure option is to not specify a directory (i.e. ./configure -with-imap). This will assume that you have the imap-2007e source installed in the ../imap directory relative to the Asterisk source. If you do not have this source, then configure will default to the "system" option defined in the next paragraph

A third option is ./configure -with-imap=system. This will assume that you have installed a dynamically linked version of the c-client library (most likely via a package provided by your distro). This will attempt to link agains -lc-client and will search for c-client headers in your include path starting with the imap directory, and upon failure, in the c-client directory.

When you run 'make menuselect', choose 'Voicemail Build Options' and the IMAP_STORAGE option should be available for selection.

After selecting the IMAP_STORAGE option, use the 'x' key to exit menuselect and save your changes, and the build/install Asterisk normally.

IMAP VM Storage Voicemail.conf Modifications

The following directives have been added to voicemail.conf:

- imapserver=<name or IP address of IMAP mail server>
- imapport=<IMAP port, defaults to 143>
- imapflags=<IMAP flags, "novalidate-cert" for example>
- imapfolder=<IMAP folder to store messages to>
- imapgreetings=<yes or no>
- greetingsfolder=<IMAP folder to store greetings in if imapgreetings is enabled>
- expungeonhangup=<yes or no>
- authuser=<username>
- authpassword=<password>
- opentimeout=<TCP open timeout in seconds>
- closetimeout=<TCP close timeout in seconds>
- readtimeout=<TCP read timeout in seconds>
- writetimeout=<TCP write timeout in seconds>

The "imapfolder" can be used to specify an alternative folder on your IMAP server to store voicemails in. If not specified, the default folder 'INBOX' will be used.

The "imapgreetings" parameter can be enabled in order to store voicemail greetings on the IMAP server. If disabled, then they will be stored on the local file system as normal.

The "greetingsfolder" can be set to store greetings on the IMAP server when "imapgreetings" is enabled in an alternative folder than that set by "imapfolder" or the default folder for voicemails.

The "expungeonhangup" flag is used to determine if the voicemail system should expunge all messages marked for deletion when the user hangs up the phone.

Each mailbox definition should also have imapuser=imap username. For example:

```
4123=>4123,James
Rothenberger,jar@onebiztone.com,,attach=yes|imapuser=jar
```

The directives "authuser" and "authpassword" are not needed when using Kerberos. They are defined to allow Asterisk to authenticate as a single user that has access to all mailboxes as an alternative to Kerberos.

Voicemail and IMAP Folders

Besides INBOX, users should create "Old", "Work", "Family" and "Friends" IMAP folders at the same level of hierarchy as the INBOX. These will be used as alternate folders for storing voicemail messages to mimic the behavior of the current (file-based) voicemail system.

Please note that it is not recommended to store your voicemails in the top level folder where your users will keep their emails, especially if there are a large number of emails. A large number of emails in the same folder(s) that you're storing your voicemails could cause a large delay as Asterisk must parse through all the emails. For example a mailbox with 100 emails in it could take up to 60 seconds to receive a response.

Separate vs. Shared E-mail Accounts

As administrator you will have to decide if you want to send the voicemail messages to a separate IMAP account or use each user's existing IMAP mailbox for voicemail storage. The IMAP storage mechanism will work either way.

By implementing a single IMAP mailbox, the user will see voicemail messages appear in the same INBOX as other messages. The disadvantage of this method is that if the IMAP server does NOT support UIDPLUS, Asterisk voicemail will expunge ALL messages marked for deletion when the user exits the voicemail system, not just the VOICEMAIL messages marked for deletion.

By implementing separate IMAP mailboxes for voicemail and email, voicemail expunges will not remove regular email flagged for deletion.

IMAP Server Implementations

There are various IMAP server implementations, each supports a potentially different set of features.

UW IMAP-2005 or earlier

UIDPLUS is currently NOT supported on these versions of UW-IMAP. Please note that without UID_EXPUNGE, Asterisk voicemail will expunge ALL messages marked for deletion when a user exits the voicemail system (hangs up the phone).
This version is **not recommended for Asterisk.**

UW IMAP-2006

This version supports UIDPLUS, which allows UID_EXPUNGE capabilities. This feature allow the system to expunge ONLY pertinent messages, instead of the default behavior, which is to expunge ALL messages marked for deletion when EXPUNGE is called. The IMAP storage mechanism is this version of Asterisk will check if the UID_EXPUNGE feature is supported by the server, and use it if possible.
This version is **not recommended for Asterisk.**

UW IMAP-2007

This is the currently recommended version for use with Asterisk.

Cyrus IMAP

Cyrus IMAP server v2.3.3 has been tested using a hierarchy delimiter of '/'.

IMAP Voicemail Quota Support

If the IMAP server supports quotas, Asterisk will check the quota when accessing voicemail. Currently only a warning is given to the user that their quota is exceeded.

IMAP Voicemail Application Notes

Since the primary storage mechanism is IMAP, all message information that was previously stored in an associated text file, AND the recording itself, is now stored in a single email message. This means that the .gsm recording will ALWAYS be attached to the message (along with the user's preference of recording format if different - ie. .WAV). The voicemail message information is stored in the email message headers. These headers include:

- X-Asterisk-VM-Message-Num
- X-Asterisk-VM-Server-Name
- X-Asterisk-VM-Context
- X-Asterisk-VM-Extension
- X-Asterisk-VM-Priority
- X-Asterisk-VM-Caller-channel
- X-Asterisk-VM-Caller-ID-Num
- X-Asterisk-VM-Caller-ID-Name
- X-Asterisk-VM-Duration
- X-Asterisk-VM-Category
- X-Asterisk-VM-Orig-date
- X-Asterisk-VM-Orig-time

Asterisk SIP Connections

Fishook: We need a single spot to compile the definitive guide to Asterisk's SIP capabilities.

Content coming. WIP beginning here:
[~mdavenport:1.8 SIP Scratchpad]

Asterisk GUI

Introduction to Asterisk GUI

Asterisk GUI is a framework for the creation of graphical interfaces for configuring Asterisk. Some sample graphical interfaces for specific vertical markets are included for reference or for actual use and extension.

Software License

Asterisk GUI HTML and Javascript files Copyright (C) 2006-2011 Digium, Inc.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 2 only. This software is also available under commercial terms from Digium, Inc.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Please contact Digium for information on alternative licensing arrangements for Asterisk GUI.

Download

While package release is inconsistent and infrequent, you can always get a current copy of Asterisk GUI from subversion. The current stable version will always be under `branches` and is currently located in `branches/2.0`.

Please note that Asterisk GUI is not officially supported, though bugs, patches, and feature requests may be submitted at http://issues.asterisk.org and should reference the Asterisk GUI project. You may also find peer support in the #asterisk-gui IRC channel and the Asterisk GUI forum.

Installation and Configuration

Installation

```
$ ./configure
$ make
$ make install
$ make checkconfig
```

Configuration

You may install sample configuration files by doing "make samples". Also you will need to edit your Asterisk configuration files to enable Asterisk GUI properly, specifically:

http.conf:

Enable http.

```
[general]
enabled=yes
enablestatic=yes
#bindaddr=0.0.0.0 # allow GUI to be accessible from all IP
addresses.
bindaddr=127.0.0.1 # require access from the machine Asterisk is
running on
bindport=8088
```

manager.conf

Enable manager access.

```
[general]
enabled = yes
webenabled = yes
```

Create an appropriate entry in manager.conf for the administrative user (PLEASE READ THE security.txt FILE!)

```
[admin]
secret = thiswouldbeaninsecurepassword
read =
system,call,log,verbose,command,agent,config,read,write,originate
write =
system,call,log,verbose,command,agent,config,read,write,originate
```

Troubleshooting

Check your filesystem permissions:

```
$ chown -R asterisk:asterisk /etc/asterisk/ /var/lib/asterisk # if
asterisk runs as the user "asterisk"
$ chmod 644 /etc/asterisk/*
```

Check that the bindaddr value in /etc/asterisk/http.conf matches the IP address of the machine you're using to access Asterisk GUI.

Check that the dahdi_genconf script runs correctly and creates /etc/asterisk/dahdi_guiread.conf.

Check the existence of /etc/asterisk/guipreferences.conf and inside that file, the existence of the following line:

```
config_upgraded = yes
```

Check the last modified date of /etc/asterisk/http.conf. Asterisk GUI modifies this file every time it is loaded.

Check that the user Asterisk runs as has a login shell.

```
su - asterisk
```

Development

Debugging

To turn on debug messages, open config/js/session.js. On line 30, set "log" to "true":

```
log: true, /**< boolean toggling logging */
```

Hide Menu Categories

Hide menu categories by changing the HTML class attribute to "AdvancedMode" in index.html. Show them by enabling Advanced Options in the GUI.

Development

This section includes some information related to the core development of Asterisk. Most of the documentation related to the source code is embedded in the source files and is processed with Doxygen. The latest versions of the source documentation can be found on asterisk.org.

Policies and Procedures

Commit Messages

1. Commit Message Formatting

The following illustrates the basic outline for commit messages:

```
<One-liner summary of changes>

<Empty Line>

<Verbose description of the changes>

<Empty Line>

<Special Tags>
```

Some commit history viewers treat the first line of commit messages as the summary for the commit. So, an effort should be made to format our commit messages in that fashion. The verbose description may contain multiple paragraphs, itemized lists, etc. Always end the first sentence (and any subsequent sentences) with punctuation.

Commit messages should be wrapped at 80 columns.

Note that for trivial commits, such as "fix the build", or "fix spelling mistake", the verbose description may not be necessary.

2. Special Tags for Commit Messages

2.1. JIRA (https://issues.asterisk.org/jira/)

To have a commit noted in an issue, use a tag of the form:

```
(issue ASTERISK-1234)
```

To have a commit automatically close an issue, use a tag of the form:

```
(closes issue ASTERISK-1234)
```

Also include information about who reported an issue, or who contributed testing feedback:

```
(closes issue ASTERISK-1234)
Reported by: SomeGuy
Tested by: SomeOtherGuy, SomeOtherGuy2
Patches:
      fix_bug_1234.diff uploaded by SomeDeveloper (license 5678)
```

If the patch being committed was written by the person doing the commit, and is not available to reference as an upload to the issue, there is no need to include something like "fixed by me", as that will be the default assumption when a specific patch is not referenced.

2.2. Review Board (https://reviewboard.asterisk.org/)

To have a commit set a review request as submitted, include the full URL to the review request. For example:

```
Review: https://reviewboard.asterisk.org/r/95/
```

3. Commit Messages with svnmerge

When using the svnmerge tool for merging changes between branches, use the *commit message generated by svnmerge.* The '-F' option to svnmerge allows you to specify a file for *svnmerge* to write out a commit message to. The '-F' option to svn commit allows you to specify a file that contains the commit message.

> ✅ If you are using the expect script wrappers for svnmerge from repotools, a commit message is automatically placed in the file '../merge.msg'.

For more detailed information about working with branches and merging, see Subversion Usage.

Issue Tracker Workflow

Description of the Issue Tracker Workflow

(This document is most beneficial for Asterisk bug marshals, however it is good reading for

anyone who may be filing issues or wondering how the Asterisk Open Source project moves issues through from filing to completion.)

The workflow in the issue tracker should be handled in the following way:

1. A bug is reported and is automatically placed in the 'New' status.
2. The Bug Marshall team should go through bugs in the 'New' status to determine whether the report is valid (not a duplicate, hasn't already been fixed, not a Digium tech support issue, etc.). Invalid reports should be set to 'Closed' with the appropriate resolution set. Categories and descriptions should be corrected at this point.[~russell:Note1]
   Issues should also have enough information for a developer to either reproduce the issue or determine where an issue exists (or both). If this is not the case then the issue should be moved to 'Feedback' prior to moving forward in the workflow.
3. The next step is to determine whether the report is about a bug or a submission of a new feature:
   a. BUG: A bug should be moved into the status 'Acknowledged' if enough information has been provided by the reporter to either reproduce the issue or clearly see where an issue may lie. The bug may also be assigned to a developer for the creation of the initial patch, or review of the issue.
      Once a patch has been created for the issue and attached, the issue can then be moved to the 'Confirmed' status. At this point, initial code review and discussion about the patch will take place. Once an adequate amount of support for the implementation of the patch is acquired, then the bug can be moved to the 'Ready for Testing' status for wider testing by the community. After the testing phase is complete and it appears the issue is resolved, the patch can be committed by a developer and closed.
   b. FEATURE: As new features should be filed with a patch, it can be immediately moved to the 'confirmed' status, making it ready for basic formatting and code review. From there any changes to style or feel of the patch based on feedback from the community can be discussed, and changes to the patch made. It can then be moved forward to the 'Ready for Testing' status. Once the feature has been merged, or a decision has been made that it will not be merged, the issue should be taken to 'Closed' with the appropriate resolution.[~russell:Note2]
4. If at any point in the workflow, an issue requires feedback from the original poster of the issue, the status should be changed to 'Feedback'. Once the required information has been provided, it should be placed back in the appropriate point of the workflow.
5. If at any point in the workflow, a developer or bug marshal would like to take responsibility for doing the work that is necessary to progress an issue, the status can be changed to 'Assigned'. At that point the developer assigned to the issue will be responsible for moving the issue to completion.

Workflow Summary

The following is a list of valid statuses and what they mean to the work flow.

New

This issue is awaiting review by bug marshals. Categorization and summaries should be fixed as appropriate.

Feedback

This issue requires feedback from the poster of the issue before any additional progress in the workflow can be made. This may include providing additional debugging information, or a backtrace with DONT_OPTIMIZE enabled, for example. (See the doc/HOWTO_collect_debug_information.txt file in your Asterisk source.)

Acknowledged

This is a submitted bug which has no patch associated with it, but appears to be a valid bug based on the description and provided debugging information.

Confirmed

The patch associated with this issue requires initial formatting and code review, and may have some initial testing done. It is waiting for a developer to confirm the patch will no longer need large changes made to it, and is ready for wider testing from the community. This stage is used for discussing the feel and style of a patch, in addition to the coding style utilized.

For Testing

This is an issue which has a patch that is waiting for testing feedback from the community after it has been deemed to no longer need larger changes.

Assigned

A developer or bug marshal has taken responsibility for taking the necessary steps to move forward in the workflow. Once the issue is ready to be reviewed and feedback provided, it should be placed back into the appropriate place of the workflow.

Resolved

A resolution for this issue has been reached. This issue should immediately be Closed.

Closed

No further action is necessary for this issue.

Severity Levels

Severity levels generally represent the number of users who are potentially affected by the reported issue.

Feature

This issue is a new feature and will only be committed to Asterisk trunk. Asterisk trunk is where future branches will be created and thus this feature will only be found in future branches of Asterisk and not merged into existing branches. (See Release Branch Commit Policy below.)

Trivial

A trivial issue is something that either affects an insignificant number of Asterisk users, or is a minimally invasive change that does not affect functionality.

Text

A text issue is typically something like a spelling fix, a clarifying of a debugging or verbose message, or changes to documentation.

Tweak

A tweak to the code the has the potential to either make code clearer to read, or a change that could speed up processing in certain circumstances. These changes are typically only a couple of lines

.

Minor

An issue that does not affect a large number of Asterisk users, but not an insignificant number.

The number of lines of code and development effort to resolve this issue could be non-trivial.

Major

As issue that affects the majority of Asterisk users. The number of lines of code and development effort required to resolve this issue could be non-trivial.

Crash

An issue marked as a Crash is something that would cause Asterisk to be unusable for a majority of Asterisk users and is an issue that causes a deadlock or crash of the Asterisk process.

Block

A blocking issue is an issue that must be resolved before the next release of Asterisk as would affect a significant number of Asterisk users, or could be a highly visible regression. A severity of block should only be set by Asterisk bug marshals at their discretion.

** USERS SHOULD NOT FILE ISSUES WITH A SEVERITY OF BLOCK **

Priority Levels

Currently, the following priority levels are listed on the issue tracker:

- None
- Low
- Normal
- High
- Urgent
- Immediate

However, at this time they are not utilized and all new issue should have a priority of 'Normal'.

Notes

1. Using the "Need Triage" filter is useful for finding these issues quickly.
2. The issue tracker now has the ability to monitor the commits list, and if the commit message contains something like, "(Closes issue #9999)", the bug will be automatically closed.
   See http://www.asterisk.org/doxygen/trunk/CommitMessages.html for more information on commit messages.

Release Branch Commit Policy

The code in the release branches should be changed as little as possible. The only time the release branches will be changed is to fix a bug. New features will never be included in the release branch unless a special exception is made by the release branch maintainers.

Sometimes it is difficult to determine whether a patch is considered to fix a bug or if it is a new feature. Patches that are considered code cleanup, or to improve performance, are NOT to be included in the release branches. Performance issues will only be considered for the release branch if they are considered significant, and should be approved by the maintainers.

If there is ever a question about what should be included in the release branch, the maintainers should be allowed to make the decision.

Reviewboard Usage

Usage Guidelines

Mantis and Reviewboard are both utilities that the Asterisk development community uses to help track and review code being written for Asterisk. Since both systems are used for posting patches, it is worth discussing when it is appropriate to use reviewboard and when not.

Here are the situations in which it is appropriate to post code to reviewboard:

- A committer has a patch that they would like to get some feedback on before merging into one of the main branches.
- A committer or bug marshal has requested a contributor to post their patch from Mantis on reviewboard to aid in the review process. This typically happens with complex code contributions where reviewboard can help aid in providing feedback.

We do encourage all interested parties to participate in the review process. However, aside from the cases mentioned above, we prefer that all code submissions first go through Mantis.

> ⚠ It is acceptable for a committer to post patches to reviewboard before they are complete to get some feedback on the approach being taken. However, if the code is not yet ready to be merged, it must be documented as such. A review request with a patch proposed for merging should have documented testing and should not have blatant coding guidelines violations. Lack of these things is careless and shows disrespect for those reviewing your code.

Posting Code to Reviewboard

Using post-review

The easiest way to post a patch to reviewboard is by using the post-review tool. Install it using `easy_install`.

> ⚠ If you do not already have `easy_install`, install the `python-setuptools` package.

```
$ sudo easy_install -U RBTools
```

Essentially, post-review is a script that will take the output of "svn diff" and create a review request out of it for you. So, once you have a working copy with the changes you expect in the output of "svn diff", you just run the following command:

```
$ post-review
```

If it complains about not knowing which reviewboard server to use, add the server option:

```
$ post-review --server=https://reviewboard.asterisk.org
```

Dealing with New Files

I have one final note about an oddity with using post-review. If you maintain your code in a team branch, and the new code includes new files, there are some additional steps you must take to

get post-review to behave properly.

You would start by getting your changes applied to a trunk working copy:

```
$ cd .../trunk
```

Then, apply the changes from your branch:

```
$ svn merge .../trunk .../team/group/my_new_code
```

Now, the code is merged into your working copy. However, for a new file, subversion treats it as a copy of existing content and not new content, so new files don't show up in "svn diff" at this point. To get it to show up in the diff, use the following commands so svn treats it as new content and publishes it in the diff:

```
$ svn revert my_new_file.c
    $ svn add my_new_file.c
```

Now, it should work, and you can run "post-review" as usual.

Updating Patch on Existing Review Request

Most of the time, a patch on reviewboard will require multiple iterations before other sign off on it being ready to be merged. To update the diff for an existing review request, you can use post-review and the -r option. Apply the current version of the diff to a working copy as described above, and then run the following command:

```
$ post-review -r <review request number>
```

Debugging

Collecting Debug Information

Collecting Debug Information for the Asterisk Issue Tracker

This document will provide instructions on how to collect debugging logs from an Asterisk machine, for the purpose of helping bug marshals troubleshoot an issue on https://issues.asterisk.org

PREREQUISITES

- Asterisk 1.4.30 or greater.

STEPS

1. Edit the logger.conf file to enable debug output to your filesystem. Add the following line. The word "myDebugLog" can be changed to anything you want, as that is the filename the logging will be written to. A good example might be something like: issue_12345_full_log

logger.conf
``` [logfiles] myDebugLog => notice,warning,error,debug,verbose,dtmf ```

2. From the Asterisk CLI, restart the logger module:

```
*CLI> core set verbose 15
*CLI> core set debug 15
*CLI> module reload logger
```

Optionally, if you've used this file to record data previously, then rotate the logs:

```
*CLI> logger rotate
```

2.1. Depending on your issue, be sure to enable the channel driver logging.

SIP (1.6.0 or higher)

```
*CLI> sip set debug on
```

SIP (1.4)

```
*CLI> sip set debug
```

IAX2 (1.6.0 or higher)

```
*CLI> iax2 set debug on
```

IAX2 (1.4)

```
*CLI> iax2 set debug
```

3. Reproduce your issue.

4. Once finished, be sure to disable the extra debugging:

```
*CLI> core set verbose 0
*CLI> core set debug 0
```

4.1. Again, remember to disable any extra logging if you enabled it in the channel driver.

SIP (1.4 or higher)

```
*CLI> sip set debug off
```

IAX2 (1.4 or higher)

```
*CLI> iax2 set debug off
```

5. Upload the file located in /var/log/asterisk/myDebugLog to the issue tracker.

> **Important**
> Do **NOT** post the output of your file as a comment. This clutters the issue and will only result in your comment being deleted.

6. Disable logging to the filesystem. Edit the logger.conf file and comment out or delete the line you added in step 1. Using a semi-colon as the first character on the line will comment out the line.

logger.conf
[logfiles] ;myDebugLog => notice,warning,error,debug,verbose,dtmf

Then reload the logger module like in step 2:

```
*CLI> module reload logger
```

Getting a Backtrace

This document is intended to provide information on how to obtain the backtraces required on the asterisk bug tracker, available at https://issues.asterisk.org.

- Overview
- Preparing Asterisk To Produce Core Files On Crash
- Getting Information After A Crash
- Getting Information For A Deadlock
- Verify Your Backtraces
- Uploading Your Information To The Issue Tracker

Overview

The backtrace information is required by developers to help fix problems with bugs of any kind. Backtraces provide information about what was wrong when a program crashed; in our case, Asterisk.

Preparing Asterisk To Produce Core Files On Crash

First of all, when you start Asterisk, you MUST start it with option -g. This tells Asterisk to produce a core file if it crashes.

If you start Asterisk with the safe_asterisk script, it automatically starts using the option -g.

If you're not sure if Asterisk is running with the -g option, type the following command in your shell:

```
# ps aux | grep asterisk
root      17832  0.0  1.2   2348    788 pts/1    S    Aug12   0:00
/bin/sh /usr/sbin/safe_asterisk
root      26686  0.0  2.8  15544   1744 pts/1    S    Aug13   0:02
asterisk -vvvg -c
[...]
```

The interesting information is located in the last column.

Second, your copy of Asterisk must have been built without optimization or the backtrace will be (nearly) unusable. This can be done by selecting the 'DONT_OPTIMIZE' option in the Compiler Flags submenu in the 'make menuselect' tree before building Asterisk.

> **Using BETTER_BACKTRACES**
>
> As of Asterisk versions 1.4.40, 1.6.2.17, and 1.8.3, the option BETTER_BACKTRACES which uses libbfd, will provide better symbol information within both the Asterisk binary, as well as loaded modules, to assist when using inline backtraces to track down problems. **It is recommended that you enable both DONT_OPTIMIZE and BETTER_BACKTRACES**

Running a production server with DONT_OPTIMIZE is generally safe. You'll notice the binary files may be a bit larger, but in terms of Asterisk performance, impact should be negligible.

After Asterisk crashes, a file named "core" will be dumped in the present working directory of the Linux shell from which Asterisk was started.

In the event that there are multiple core files present, it is important to look at the file timestamps in order to determine which one you really intend to look at.

Getting Information After A Crash

There are two kind of backtraces (aka 'bt') which are useful: bt and bt full.

Now that we've verified the core file has been written to disk, the final part is to extract 'bt' from the core file. Core files are pretty big, don't be scared, it's normal.

For extraction, we use a really nice tool, called gdb. To verify that you have gdb installed on your system:

```
# gdb -v
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.  Type "show
copying"
and "show warranty" for details.

This GDB was configured as "i486-linux-gnu"...
```

If you don't have gdb installed, go install gdb. You should be able to install using something like: apt-get install gdb -or- yum install gdb

✅ Just run the following command to get the output into the backtrace.txt file, ready for uploading to the issue tracker. Be sure to change the name of the core file to your actual core dump file:

```
# gdb -se "asterisk" -ex "bt full" -ex "thread apply all bt" --batch
-c core > /tmp/backtrace.txt
```

Now load the core file in gdb with the following command. This will also save the output of gdb to the /tmp/backtract.txt file.

```
# gdb -se "asterisk" -c core | tee /tmp/backtrace.txt
[...]
(You would see a lot of output here.)
[...]
Reading symbols from
/usr/lib/asterisk/modules/app_externalivr.so...done.
Loaded symbols for /usr/lib/asterisk/modules/app_externalivr.so
#0  0x29b45d7e in ?? ()
(gdb)
```

In order to make extracting the gdb output easier, you may wish to turn on logging using "set logging on". This command will save all output to the default file of gdb.txt, which in the end can be uploaded as an attachment to the bug tracker.

Now at the gdb prompt, type: bt You would see output similar to:

```
(gdb) bt
#0   0x29b45d7e in ?? ()
#1   0x08180bf8 in ?? ()
#2   0xbcdffa58 in ?? ()
#3   0x08180bf8 in ?? ()
#4   0xbcdffa60 in ?? ()
#5   0x08180bf8 in ?? ()
#6   0x180bf894 in ?? ()
#7   0x0bf80008 in ?? ()
#8   0x180b0818 in ?? ()
#9   0x08068008 in ast_stopstream (tmp=0x40758d38) at file.c:180
#10  0x000000a0 in ?? ()
#11  0x000000a0 in ?? ()
#12  0x00000000 in ?? ()
#13  0x407513c3 in confcall_careful_stream (conf=0x8180bf8,
filename=0x8181de8 "DAHDI/pseudo-1324221520") at app_meetme.c:262
#14  0x40751332 in streamconfthread (args=0x8180bf8) at
app_meetme.c:1965
#15  0xbcdffbe0 in ?? ()
#16  0x40028e51 in pthread_start_thread () from /lib/libpthread.so.0
#17  0x401ec92a in clone () from /lib/libc.so.6
(gdb)
```

The bt's output is the information that we need on the bug tracker.

Now do a bt full as follows:

```
(gdb) bt full
#0  0x29b45d7e in ?? ()
No symbol table info available.
#1  0x08180bf8 in ?? ()
No symbol table info available.
#2  0xbcdffa58 in ?? ()
No symbol table info available.
#3  0x08180bf8 in ?? ()
No symbol table info available.
#4  0xbcdffa60 in ?? ()
No symbol table info available.
#5  0x08180bf8 in ?? ()
No symbol table info available.
#6  0x180bf894 in ?? ()
No symbol table info available.
#7  0x0bf80008 in ?? ()
No symbol table info available.
#8  0x180b0818 in ?? ()
No symbol table info available.
#9  0x08068008 in ast_stopstream (tmp=0x40758d38) at file.c:180
No locals.
#10 0x000000a0 in ?? ()
No symbol table info available.
#11 0x000000a0 in ?? ()
No symbol table info available.
#12 0x00000000 in ?? ()
No symbol table info available.
#13 0x407513c3 in confcall_careful_stream (conf=0x8180bf8,
filename=0x8181de8 "DAHDI/pseudo-1324221520") at app_meetme.c:262
        f = (struct ast_frame *) 0x8180bf8
        trans = (struct ast_trans_pvt *) 0x0
#14 0x40751332 in streamconfthread (args=0x8180bf8) at
app_meetme.c:1965
No locals.
#15 0xbcdffbe0 in ?? ()
No symbol table info available.
#16 0x40028e51 in pthread_start_thread () from /lib/libpthread.so.0
No symbol table info available.
#17 0x401ec92a in clone () from /lib/libc.so.6
No symbol table info available.
(gdb)
```

The final "extraction" would be to know all traces by all threads. Even if Asterisk runs on the same thread for each call, it could have created some new threads.

To make sure we have the correct information, just do:

```
(gdb) thread apply all bt

Thread 1 (process 26252):
#0   0x29b45d7e in ?? ()
#1   0x08180bf8 in ?? ()
#2   0xbcdffa58 in ?? ()
#3   0x08180bf8 in ?? ()
#4   0xbcdffa60 in ?? ()
#5   0x08180bf8 in ?? ()
#6   0x180bf894 in ?? ()
#7   0x0bf80008 in ?? ()
#8   0x180b0818 in ?? ()
#9   0x08068008 in ast_stopstream (tmp=0x40758d38) at file.c:180
#10  0x000000a0 in ?? ()
#11  0x000000a0 in ?? ()
#12  0x00000000 in ?? ()
#13  0x407513c3 in confcall_careful_stream (conf=0x8180bf8,
filename=0x8181de8 "DAHDI/pseudo-1324221520") at app_meetme.c:262
#14  0x40751332 in streamconfthread (args=0x8180bf8) at
app_meetme.c:1965
#15  0xbcdffbe0 in ?? ()
#16  0x40028e51 in pthread_start_thread () from /lib/libpthread.so.0
#17  0x401ec92a in clone () from /lib/libc.so.6
(gdb)
```

That output tells us crucial information about each thread.

Getting Information For A Deadlock

Whenever supplying information about a deadlock (i.e. when you run the 'core show locks'
command on the Asterisk console), it is useful to also have additional information about the
threads. We can generate this information by attaching to a running Asterisk process and
gathering that information.

> ⚠ You need **DEBUG_THREADS** enabled in the Compiler Flags menu of menuselect. Be sure you recompile, install, and restart
> Asterisk prior to running 'core show locks'.

You can easily attach to a running Asterisk process, gather the output required and then detach
from the process all in a single step. Execute the following command and upload the resulting
backtrace-threads.txt file to the Asterisk issue tracker:

```
# gdb -ex "thread apply all bt" --batch /usr/sbin/asterisk `pidof
asterisk` > /tmp/backtrace-threads.txt
```

Note that this gathers information from the running Asterisk process, so you want to make sure you run this command immediately before or after gathering the output of 'core show locks'. You can gather that information by running the following command:

```
# asterisk -rx "core show locks" > /tmp/core-show-locks.txt
```

Verify Your Backtraces

Before uploading your backtraces to the issue tracker, you should double check to make sure the data you have is of use to the developers. Check your backtrace files to make sure you're not seeing several of the following:

```
<value optimized out>
```

If you are, then you likely haven't compiled with DONT_OPTIMIZE. The impact of DONT_OPTIMIZE is negligible on most systems. Be sure you've enabled the DONT_OPTIMIZE flag within the Compiler Flags section of menuselect. After doing so, be sure to run 'make install' and restart Asterisk.

> ⛔ Double check your backtrace! It is important you do not see <value optimized out>, otherwise the backtrace does not provide enough information for a developer.

Uploading Your Information To The Issue Tracker

You're now ready to upload your files to the Asterisk issue tracker (located at https://issues.asterisk.org).

> ⚠️ Please ATTACH your output! DO NOT paste it as a note!

If you have questions or comments regarding this documentation, feel free to pass by the #asterisk-bugs channel on irc.freenode.net.

Valgrind

If you're having certain types of crashes, such as those associated with memory corruption, a bug marshal may ask you to run Asterisk under valgrind. You should follow these steps, to give the bug marshal the maximum amount of information about the crash.

1. Run 'make menuselect' and in the Compiler Options, enable MALLOC_DEBUG and DONT_OPTIMIZE. A bug marshal may also ask you to enable additional compiler flags, such as DEBUG_THREADS, depending upon the nature of the issue.
2. Rebuild and install Asterisk.
3. Run Asterisk as follows, where /usr/src/asterisk/ is location of asterisk source code.

```
valgrind --suppressions=/usr/src/asterisk/contrib/valgrind.supp --log-fd=9 asterisk -vvvvcg
9>valgrind.txt
```

4. Reproduce the issue. Following the manifestation of the issue (or when the process crashes), upload the valgrind.txt to the issue tracker.

Please note that even if valgrind prevents Asterisk from crashing, the information logged may STILL be of use to developers, so please upload the resulting log file whether Asterisk crashes or not.

Subversion Usage

1. Commit Access

1.1. Configuration

The subversion server uses SSL client certificates to handle authentication of users. When you are granted commit access, you will be provided two files. These files should be placed in your `~/.subversion/` directory.

1. Digium_SVN-cacert-sha1.pem
2. <name>-cert.p12

The following should be placed in the `~/.subversion/servers` file:

~/.subversion/servers

```
[groups]
digium = *.digium.com

[digium]
ssl-client-cert-file =
/home/_<username>_/.subversion/_<name>_-cert.p12

[global]
ssl-authority-files =
/home/_<username>_/.subversion/Digium_SVN-cacert-sha1.pem
```

It is also recommended that you add the following to the `~/.subversion/config` file. Otherwise, you will have problems using our tools for merging changes between branches.

<table>
<tr><th colspan="1" style="text-align:center">~/.subversion/config</th></tr>
</table>

```
[miscellany]
interactive-conflicts=no
```

1.2. SVN Checkouts

Checkouts that come from `http://svn.asterisk.org/` are read-only copies of the repositories. When doing a checkout that you intend to commit to, it must be from `https://origsvn.digium.com/`. For example:

```
$ svn co https://origsvn.digium.com/svn/asterisk/trunk
```

2. Using `svnmerge` for Cross-Branch Merging

2.1. Tools Installation

You must install `svnmerge` and the related wrappers from our `repotools` repository. The wrapper scripts use `expect`, so be sure to install that, too.

```
$ svn co http://svn.asterisk.org/svn/repotools
$ cd repotools
$ ./configure
$ sudo make install
```

2.2. `svnmerge` Properties

If you do a `svn pl -v` while you are located in an svn checkout, you will see all the properties currently attached to the root directory. For instance, on a checked out copy of Asterisk trunk, you will see something like this:

```
branch-1.8-blocked
     /branches/1.8:
  branch-1.8-merged

/branches/1.8:1-279056,279113,279227,279273,279280,...............,286
```

and on the 1.8 branch, you will see these sort of properties:

```
branch-1.6.2-blocked
     /branches/1.6.2:279852,279883,280227,280556,280812,282668
   branch-1.6.2-merged


/branches/1.6.2:1-279056,279207,279501,279561,279597,279609,.........
```

These properties identify the following things:

1. The branch changes are being merged from
   - `branch-<branch>-...`
2. The revisions merged from that branch
   - `branch-<branch>-merged:/branches/<branch>:<revisions>`
3. The revision explicitly not merged, or blocked, from that branch
   - `branch-<branch>-blocked:/branches/<branch>:<revisions>`

2.3. Branch Merging Order

When committing a change that applies to more than one branch, the change first goes into the oldest branch and is then merged up to the next one. If a branch is reached where the change should not be merged up, it should be explicitly blocked. The following diagram shows the current branch merge order.
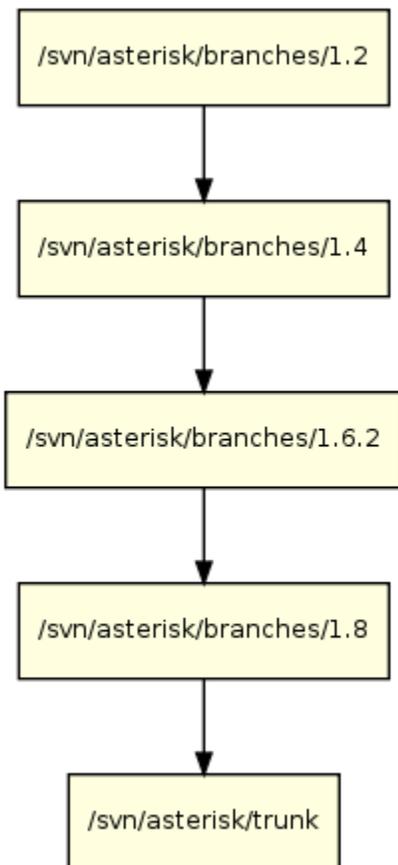
- 1.2 -> 1.4
  - `merge24 <revision>`
  - `block24 <revision>`

- 1.4 -> 1.6.2

- merge46 *<revision>*
- block46 *<revision>*

- 1.6.2 -> 1.8
  - merge68 *<revision>*
  - block68 *<revision>*

- 1.8 -> trunk
  - merge8trunk *<revision>*
  - block8trunk *<revision>*

> ✅ All of these scripts create a commit message for you in the file `../merge.msg`. Run `svn commit` and use that commit message with the following command:
>
> ```
> $ svn commit -F ../merge.msg
> ```

> ✅ Sometimes when you go to commit your changes after merging from another branch, you will end up with a conflict. The conflict will typically be against `.` (period). To resolve the conflict, run `svn resolved .` prior to committing.

2.4. Backporting Changes

Sometimes a change is made in a branch and later it is decided that it should be backported to an older branch. For example, a change may have gone into the 1.8 branch and later needs to be backported to the 1.6.2 branch. To handle this, first manually make the change and commit to the 1.6.2 branch. Then, there is another wrapper similar to `merge68` and `block68` to record that the code from a revision already exists in the 1.8 branch. The wrapper is `record68`.

```
$ cd 1.8
$ record68 <revision>
$ svn commit -F ../merge.msg
```

3. Developer Branches

> ℹ️ If you have been granted workspace on the server, you will have read and electronically signed the Open Source Contributor License found at https://issues.asterisk.org (upon signing in) and have been given an SSL client certificate.

Developer branches are stored in the `/team/<name>` directory of each project repository (and `/team/<name>/private` for private branches).

3.1. Creating a Developer Branch

Use the following commands to create a branch and prepare it for future merge tracking of the branch you created it from. This example creates a branch off of Asterisk trunk.

```
$ svn copy https://origsvn.digium.com/svn/asterisk/trunk
https://origsvn.digium.com/svn/asterisk/team/jdoe/my-fun-branch
$ svn checkout
https://origsvn.digium.com/svn/asterisk/team/jdoe/my-fun-branch
$ cd my-fun-branch
$ svnmerge init
$ svn commit -m "initialize branch for merge tracking"
```

3.2. Group Branches

Group branches are developer branches intended to be worked on by more than one developer. Instead of putting them in `/team/<name>`, they go in the `/team/group` directory, instead. Otherwise, they're managed in the exact same way as other developer branches.

3.3. Automatically Keeping Branches Up to Date Using `automerge`

Our subversion server provides the ability to automatically keep developer branches up to date with their parent. To enable this feature, set the `automerge` and `automerge-email` properties on the root directory. Changes from the parent branch will be periodically (once an hour) merged into your branch. If a change from upstream conflicts with changes in the branch, the `automerge` process will stop and the address(es) listed in the `automerge-email` property will be notified.

> ⚠ Running `svnmerge init` and committing those properties is a prerequisite for `automerge` to work for a developer branch.

Use the following commands to enable automerge on a developer branch:

```
$ cd my-fun-branch
$ svn ps automerge '*' .
$ svn ps automerge-email 'me@example.com' .
$ svn commit "initialize automerge"
```

3.3.1. Setting `automerge-email` on a Group Branch

For a branch with multiple developers working on it, it may be useful to have automerge emails sent to more than one email address. To do so, just separate the email addresses in the property with commas. The value of this property is literally used as the content for the `To:` header of the email.

```
$ svn ps automerge-email
'me@example.com,you@example.com,him@example.com' .
```

3.3.2. Resolving `automerge` Conflicts

If your developer branch goes into conflict with `automerge` on, and the `automerge-email` property has been set, you will receive an email notifying you of the conflict and `automerge` will be disabled. To resolve it, use the following commands:

```
$ cd my-branch
$ svn update
$ svnmerge merge
```

Running the `svnmerge` tool will merge in the changes that cause your branch to go into conflict into your local copy. Edit the files that are in conflict to resolve the problems as appropriate. Finally, tell SVN that you have resolved the problem, re-enable automerge, and commit.

```
$ svn resolved path/to/conflicted/file
$ svn ps automerge '*' .
$ svn commit -m "resolve conflict, enable automerge"
```

3.4. Private Branches

A private developer branch is only visible to Digium and the branch owner. Management of a private branch is exactly the same as any other developer branch. The only difference is branch location. Instead of putting the branch in `/team/<name>/` the branch goes in `/team/<name>/private/`.

3.5. Merging a Developer Branch into `trunk`

> ℹ️ If your branch contains new functionality, please make sure you have made the appropriate modifications to `CHANGES` and/or `UPGRADE.txt`.

If a developer has a branch that is ready to be merged back into the trunk, here is the process:

```
$ svn co https://origsvn.digium.com/svn/asterisk/trunk
$ cd trunk
$ svn merge --ignore-ancestry
https://origsvn.digium.com/svn/asterisk/trunk
https://origsvn.digium.com/svn/asterisk/team/jdoe/bug12345 .
```

> ⚠️ This is NOT using the svnmerge script; this is just a normal SVN merge.

Once this is done, the working copy will contain the trunk plus the changes from the developer branch. If you follow the above instructions for creating branches, you have probably introduced properties to the root of the branch that need to be removed.

```
$ svn revert .
```

If you are purposely introducing new properties, or purposely introducing new values for existing properties, then you might do the following instead, so as not to destroy your properties:

```
$ svn pd svnmerge-integrated .
$ svn pd automerge .
$ svn pd automerge-email .
```

If everything merged cleanly, you can test compile and then:

```
$ svn commit -m "Merge branch for issue 12345"
```

Once the contents of your branch has been merged, please use `svn remove` to remove it from the repository. It will still be accessible if needed by looking back in the repository history if needed.

Other Reference Information

Asterisk Channel Data Stores

Asterisk Channel Data Stores

What is a data store?

A data store is a way of storing complex data (such as a structure) on a channel so it can be retrieved at a later time by another application, or the same application.

If the data store is not freed by said application though, a callback to a destroy function occurs which frees the memory used by the data in the data store so no memory loss occurs.

A datastore info structure

```
static const struct example_datastore {
        .type = "example",
        .destroy = callback_destroy
};
```

This is a needed structure that contains information about a datastore, it's used by many API calls.

How do you create a data store?

1. Use ast_datastore_alloc function to return a pre-allocated structure
Ex: datastore = ast_datastore_alloc(&example_datastore, "uid");
This function takes two arguments: (datastore info structure, uid)

2. Attach data to pre-allocated structure.

Ex: datastore->data = mysillydata;

3. Add datastore to the channel
Ex: ast_channel_datastore_add(chan, datastore);
This function takes two arguments: (pointer to channel, pointer to data store)

Full Example:

```
void callback_destroy(void *data)
{
 ast_free(data);
}

struct ast_datastore *datastore = NULL;
datastore = ast_datastore_alloc(&example_datastore, NULL);
datastore->data = mysillydata;
ast_channel_datastore_add(chan, datastore);
```

> ⚠ Because you're passing a pointer to a function in your module, you'll want to include this in your use count. When allocated increment, when destroyed decrement.

How do you remove a data store?

1. Find the data store
Ex: datastore = ast_channel_datastore_find(chan, &example_datastore, NULL);
This function takes three arguments: (pointer to channel, datastore info structure, uid)

2. Remove the data store from the channel
Ex: ast_channel_datastore_remove(chan, datastore);
This function takes two arguments: (pointer to channel, pointer to data store)

3. If we want to now do stuff to the data on the data store

4. Free the data store (this will call the destroy call back)
Ex: ast_channel_datastore_free(datastore);
This function takes one argument: (pointer to data store)

How do you find a data store?

1. Find the data store
Ex: datastore = ast_channel_datastore_find(chan, &example_datastore, NULL);
This function takes three arguments: (pointer to channel, datastore info structure, uid)

Asterisk Soundfiles Submission Process

Asterisk Soundfiles Submission Process v1.5 2009-09-01
jtodd@digium.com

The Asterisk project is looking for contributions of soundfiles to expand the usefulness of Asterisk across as many languages as
possible.

However, Digium is not able to maintain more than a few basic language sets and the project relies upon the community for the many additional languages which may be added to the soundfile list. To this end, the following criteria are to enable the administrative and technical inclusion of soundfiles in multiple languages.

Criteria for language submission:

1. Creative Commons Attribution-Share Alike 3.0 (http://creativecommons.org/licenses/by-sa/3.0/us/) for all soundfiles, and the actual artist who performs the prompts must be the one who approves (in writing) the CCBYSAv3 license. Print out the license, have the artist sign, and fax to Digium (+1-256-428-6217 Attn: Contracts) or mail it to Digium (445 Jan Davis Dr Nw, Huntsville, AL 35806, USA Attn: Contracts)
2. You must duplicate all existing sounds that are currently in the SVN trunk version of core-sounds-en.txt - no partial libraries. If you are submitting sounds for extra-sounds, a full duplicate set of the English prompts is not necessary, and alternate words may be added.
3. The artist who performed the original sounds must be available for additional work at a "reasonable" market price, and contact data must be provided with the sounds submission.
4. Soundfiles must be in ".wav" 48kHz 16-bit signed linear format - Digium will do all conversions to various codecs.
5. A soundfile map must be included with all submissions, in the same style as core-sounds-en.txt but for your specific language. This may be derived from Step 6 in the "Process" section, below.
6. Periodically and infrequently, there may be additional words added to the English "core-sounds" repository. At such time as those words are added, you would be responsible for submitting those additional sounds in your submitted language by the same artist who performed the recordings for the primary sounds, and creating an issue on issues.asterisk.org containing the new sounds.
7. If you duplicate the words/phrases in the current core-sounds-en.txt file, this does NOT guarantee complete functionality with your language. Notably, numbers and dates often require additional work with code in various parts of Asterisk to comply with specific language syntax issues. You are welcome to create patches to fully support your language - see issues.asterisk.org for more details of how to contribute code to Asterisk.
8. If you wish to update a sound set, any updates must be created by the same artist that created the base sound set.

Process:

Every time you submit sounds, even if partial sounds, you will need to execute all of these steps. That includes #8 and #9 - each submission requires a signed license from the actual artist, no matter how trivial the file set.

1. Create your language soundfile archive as a directory. Put all soundfiles into the directory.
2. Determine the two-digit ISO639-1 language code (en=english, ru=russian, etc.) and replace the "xx" in other steps with this two digit code.
3. Determine the most obvious nation in which the dialect of the artist is appropriate, if any. Select the country code from the ISO-3166-1 list. Replace "_YY" in the other steps with this two digitcode. This two-digit code should be upper-case.
4. Create a file called "CREDITS" and put in this file a short free-form block of text that you wish to include for anyone who wants to learn more about you or the artist who performed the sounds. Please minimally include the name and contact information for the artist as a URL and/or email address.

```
Example:
 bash-3.2# more CREDITS
 Recorded by:
 Allison Smith (http://www.theivrvoice.com)

 Financial Contributions by: Digium, Inc. (http://www.digium.com)
 bash-3.2#
```

5. Create a file called "LICENSE" and include the license text (listed below these instructions) which are the Creative Commons Attribution-ShareAlike v3 license terms.
6. Ensure you have a .txt file that describes all of the soundfiles by name and by textual description. UTF-8 is mandatory. See "core-sounds-en.txt" as a template. Another option (possibly easier and preferred) is to submit an OpenOffice ODS formatted spreadsheet with three columns (Col 1: module name most associated with file, col 2: filename in ASCII, col 3: text of soundfile in your localized language - see doc/lang/*.ods for examples) and it will be converted during import to UTF-8 format.
7. Use "tar" to create a single file of your sounds and the three text files.
8. Create an account on http://issues.asterisk.org/ if you do not already have one, and submit the file via that interface. In order to eliminate confusion, you should also complete the license agreement on that system, which is an electronic license agreement. You will

obtain an issue number from the issues.asterisk.org system - save this number.

9. Have the artist (NOT YOU, UNLESS YOU ARE THE ARTIST) sign a copy of the Creative Commons license and send it to Digium (see contact data, above.) Have the artist WRITE THE ISSUE NUMBER from step 8 on the Creative Commons license agreement. This is mandatory for acceptance.

10. Project staff will evaluate your sounds submission and communicate with you via the issue maanger as to the progress and status of inclusion. If your soundfiles are included, they will receive an ordinal number according to the number of other archives in this language/dialect group, a gender key, a freeform text description, the codec for that file, and the version of Asterisk for which the files were released. (Format for filenames suffix is: languagecode[_territorycode[_gender[_variant]]]) So your submission may have an archive name that after processing looks like this:

asterisk-core-sounds-en_US_female_allison-gsm-1.6.23

or

asterisk-core-sounds-en_AU_male_alex-gsm-1.6.23

Forms:

Print out everything below this block, have artist sign, and then fax or mail signed original to Digium legal:

+1 256 428-6217 Attn: Contracts

or

Attn: Contracts
445 Jan Davis Drive NW
Huntsville, AL 35806
United States

```
LICENSE FOR VOICE PROMPT FILES ------------------------------

The voice prompt files distributed herewith are Copyright (C)
2008-20[XX] [ARTIST NAME OR COPYRIGHT HOLDER NAME], and provided
under
terms of the following License.  For more information, or to
purchase
custom voice prompt files, please visit:

  [URL or contact data for artist]

I am releasing the listed recordings (delivered as sound files) as
being under the Creative Commons 3.0 Attribution/ShareAlike license,
as follows.

  [include list of files on printout]


  Signed: _____
```

```
Printed:  _____
 Name

Issue  : _[see step 9 of "Process"]_____
 Number

Date  :  _____
```

LICENSE -------

Creative Commons Legal Code

Attribution 3.0 Unported

    CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT
    PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT
    CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS
PROVIDES
    THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO
    WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS
    LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

    THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS
    CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK
IS
    PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF
THE
    WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW
    IS PROHIBITED.

    BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT
AND
    AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT
THIS
    LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS
YOU
    THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF
    SUCH TERMS AND CONDITIONS.

    1. Definitions a. "Adaptation" means a work based upon the Work,
or

upon the Work and other pre-existing works, such as a
translation,
adaptation, derivative work, arrangement of music or other
alterations of a literary or artistic work, or phonogram or
performance and includes cinematographic adaptations or any
other
form in which the Work may be recast, transformed, or adapted
including in any form recognizably derived from the original,
except that a work that constitutes a Collection will not be
considered an Adaptation for the purpose of this License. For
the
avoidance of doubt, where the Work is a musical work,
performance
or phonogram, the synchronization of the Work in timed-relation
with a moving image ("synching") will be considered an
Adaptation
for the purpose of this License. b. "Collection" means a
collection of literary or artistic works, such as encyclopedias
and anthologies, or performances, phonograms or broadcasts, or
other works or subject matter other than works listed in Section
1(f) below, which, by reason of the selection and arrangement of
their contents, constitute intellectual creations, in which the
Work is included in its entirety in unmodified form along with
one
or more other contributions, each constituting separate and
independent works in themselves, which together are assembled
into
a collective whole. A work that constitutes a Collection will
not
be considered an Adaptation (as defined above) for the purposes
of
this License. c. "Distribute" means to make available to the
public the original and copies of the Work or Adaptation, as
appropriate, through sale or other transfer of ownership.
d. "Licensor" means the individual, individuals, entity or
entities that offer(s) the Work under the terms of this License.
e. "Original Author" means, in the case of a literary or
artistic
work, the individual, individuals, entity or entities who
created
the Work or if no individual or entity can be identified, the
publisher; and in addition (i) in the case of a performance the
actors, singers, musicians, dancers, and other persons who act,
sing, deliver, declaim, play in, interpret or otherwise perform
literary or artistic works or expressions of folklore; (ii) in
the

case of a phonogram the producer being the person or legal entity
who first fixes the sounds of a performance or other sounds; and,
(iii) in the case of broadcasts, the organization that transmits
the broadcast. f. "Work" means the literary and/or artistic work
offered under the terms of this License including without
limitation any production in the literary, scientific and artistic
domain, whatever may be the mode or form of its expression
including digital form, such as a book, pamphlet and other
writing; a lecture, address, sermon or other work of the same
nature; a dramatic or dramatico-musical work; a choreographic work
or entertainment in dumb show; a musical composition with or
without words; a cinematographic work to which are assimilated
works expressed by a process analogous to cinematography; a work
of drawing, painting, architecture, sculpture, engraving or
lithography; a photographic work to which are assimilated works
expressed by a process analogous to photography; a work of applied
art; an illustration, map, plan, sketch or three-dimensional work
relative to geography, topography, architecture or science; a
performance; a broadcast; a phonogram; a compilation of data to
the extent it is protected as a copyrightable work; or a work
performed by a variety or circus performer to the extent it is not
otherwise considered a literary or artistic work. g. "You" means
an individual or entity exercising rights under this License who
has not previously violated the terms of this License with respect
to the Work, or who has received express permission from the
Licensor to exercise rights under this License despite a previous
violation. h. "Publicly Perform" means to perform public
recitations of the Work and to communicate to the public those
public recitations, by any means or process, including by wire or
wireless means or public digital performances; to make available
to the public Works in such a way that members of the public may
access these Works from a place and at a place individually chosen
by them; to perform the Work to the public by any means or

process
   and the communication to the public of the performances of the
   Work, including by public digital performance; to broadcast and
   rebroadcast the Work by any means including signs, sounds or
   images. i. "Reproduce" means to make copies of the Work by any
   means including without limitation by sound or visual recordings
   and the right of fixation and reproducing fixations of the Work,
   including storage of a protected performance or phonogram in
   digital form or other electronic medium.

   2. Fair Dealing Rights. Nothing in this License is intended to
   reduce, limit, or restrict any uses free from copyright or rights
   arising from limitations or exceptions that are provided for in
   connection with the copyright protection under copyright law or
   other applicable laws.

   3. License Grant. Subject to the terms and conditions of this
   License, Licensor hereby grants You a worldwide, royalty-free,
   non-exclusive, perpetual (for the duration of the applicable
   copyright) license to exercise the rights in the Work as stated
   below: a. to Reproduce the Work, to incorporate the Work into one
   or more Collections, and to Reproduce the Work as incorporated in
   the Collections; b. to create and Reproduce Adaptations provided
   that any such Adaptation, including any translation in any
medium,
   takes reasonable steps to clearly label, demarcate or otherwise
   identify that changes were made to the original Work. For
example,
   a translation could be marked "The original work was translated
   from English to Spanish," or a modification could indicate "The
   original work has been modified."; c. to Distribute and Publicly
   Perform the Work including as incorporated in Collections; and,
   d. to Distribute and Publicly Perform Adaptations. e. For the
   avoidance of doubt: i. Non-waivable Compulsory License Schemes.
In
   those jurisdictions in which the right to collect royalties
through
   any statutory or compulsory licensing scheme cannot be waived,
the
   Licensor reserves the exclusive right to collect such royalties
for
   any exercise by You of the rights granted under this License;
   ii. Waivable Compulsory License Schemes. In those jurisdictions
in
   which the right to collect royalties through any statutory or
   compulsory licensing scheme can be waived, the Licensor waives

the
    exclusive right to collect such royalties for any exercise by You
    of the rights granted under this License; and, iii. Voluntary
    License Schemes. The Licensor waives the right to collect
    royalties, whether individually or, in the event that the
Licensor
    is a member of a collecting society that administers voluntary
    licensing schemes, via that society, from any exercise by You of
    the rights granted under this License.

    The above rights may be exercised in all media and formats
whether
    now known or hereafter devised. The above rights include the
right
    to make such modifications as are technically necessary to
exercise
    the rights in other media and formats. Subject to Section 8(f),
all
    rights not expressly granted by Licensor are hereby reserved.

    4. Restrictions. The license granted in Section 3 above is
    expressly made subject to and limited by the following
    restrictions: a. You may Distribute or Publicly Perform the Work
    only under the terms of this License. You must include a copy of,
    or the Uniform Resource Identifier (URI) for, this License with
    every copy of the Work You Distribute or Publicly Perform. You
may
    not offer or impose any terms on the Work that restrict the terms
    of this License or the ability of the recipient of the Work to
    exercise the rights granted to that recipient under the terms of
    the License. You may not sublicense the Work. You must keep
intact
    all notices that refer to this License and to the disclaimer of
    warranties with every copy of the Work You Distribute or Publicly
    Perform. When You Distribute or Publicly Perform the Work, You
may
    not impose any effective technological measures on the Work that
    restrict the ability of a recipient of the Work from You to
    exercise the rights granted to that recipient under the terms of
    the License. This Section 4(a) applies to the Work as
incorporated
    in a Collection, but this does not require the Collection apart
    from the Work itself to be made subject to the terms of this
    License. If You create a Collection, upon notice from any
Licensor
    You must, to the extent practicable, remove from the Collection

any
    credit as required by Section 4(b), as requested. If You create
an
    Adaptation, upon notice from any Licensor You must, to the extent
    practicable, remove from the Adaptation any credit as required by
    Section 4(b), as requested. b. If You Distribute, or Publicly
    Perform the Work or any Adaptations or Collections, You must,
    unless a request has been made pursuant to Section 4(a), keep
    intact all copyright notices for the Work and provide, reasonable
    to the medium or means You are utilizing: (i) the name of the
    Original Author (or pseudonym, if applicable) if supplied, and/or
    if the Original Author and/or Licensor designate another party or
    parties (e.g., a sponsor institute, publishing entity, journal)
for
    attribution ("Attribution Parties") in Licensor's copyright
notice,
    terms of service or by other reasonable means, the name of such
    party or parties; (ii) the title of the Work if supplied; (iii)
to
    the extent reasonably practicable, the URI, if any, that Licensor
    specifies to be associated with the Work, unless such URI does
not
    refer to the copyright notice or licensing information for the
    Work; and (iv) , consistent with Section 3(b), in the case of an
    Adaptation, a credit identifying the use of the Work in the
    Adaptation (e.g., "French translation of the Work by Original
    Author," or "Screenplay based on original Work by Original
    Author"). The credit required by this Section 4 (b) may be
    implemented in any reasonable manner; provided, however, that in
    the case of a Adaptation or Collection, at a minimum such credit
    will appear, if a credit for all contributing authors of the
    Adaptation or Collection appears, then as part of these credits
and
    in a manner at least as prominent as the credits for the other
    contributing authors. For the avoidance of doubt, You may only
use
    the credit required by this Section for the purpose of
attribution
    in the manner set out above and, by exercising Your rights under
    this License, You may not implicitly or explicitly assert or
imply
    any connection with, sponsorship or endorsement by the Original
    Author, Licensor and/or Attribution Parties, as appropriate, of
You
    or Your use of the Work, without the separate, express prior
    written permission of the Original Author, Licensor and/or

Attribution Parties. c. Except as otherwise agreed in writing by
the Licensor or as may be otherwise permitted by applicable law,
if
You Reproduce, Distribute or Publicly Perform the Work either by
itself or as part of any Adaptations or Collections, You must not
distort, mutilate, modify or take other derogatory action in
relation to the Work which would be prejudicial to the Original
Author's honor or reputation. Licensor agrees that in those
jurisdictions (e.g. Japan), in which any exercise of the right
granted in Section 3(b) of this License (the right to make
Adaptations) would be deemed to be a distortion, mutilation,
modification or other derogatory action prejudicial to the
Original
Author's honor and reputation, the Licensor will waive or not
assert, as appropriate, this Section, to the fullest extent
permitted by the applicable national law, to enable You to
reasonably exercise Your right under Section 3(b) of this License
(right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING,
LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR
WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED,
STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES
OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE,
NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS,
ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT
DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF
IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY
APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY
LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE
OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF
THE
WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH
DAMAGES.

7. Termination a. This License and the rights granted hereunder
 will terminate automatically upon any breach by You of the terms
 of this License. Individuals or entities who have received
 Adaptations or Collections from You under this License, however,
 will not have their licenses terminated provided such
individuals

or entities remain in full compliance with those
licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any
termination of this License. b. Subject to the above terms and
conditions, the license granted here is perpetual (for the
duration of the applicable copyright in the Work).
Notwithstanding
the above, Licensor reserves the right to release the Work under
different license terms or to stop distributing the Work at any
time; provided, however that any such election will not serve to
withdraw this License (or any other license that has been, or is
required to be, granted under the terms of this License), and
this
License will continue in full force and effect unless terminated
as stated above.

8. Miscellaneous a. Each time You Distribute or Publicly Perform
the Work or a Collection, the Licensor offers to the recipient a
license to the Work on the same terms and conditions as the
license granted to You under this License. b. Each time You
Distribute or Publicly Perform an Adaptation, Licensor offers to
the recipient a license to the original Work on the same terms
and
conditions as the license granted to You under this License.
c. If any provision of this License is invalid or unenforceable
under applicable law, it shall not affect the validity or
enforceability of the remainder of the terms of this License,
and
without further action by the parties to this agreement, such
provision shall be reformed to the minimum extent necessary to
make such provision valid and enforceable. d. No term or
provision of this License shall be deemed waived and no breach
consented to unless such waiver or consent shall be in writing
and
signed by the party to be charged with such waiver or consent.
e. This License constitutes the entire agreement between the
parties with respect to the Work licensed here. There are no
understandings, agreements or representations with respect to
the
Work not specified here. Licensor shall not be bound by any
additional provisions that may appear in any communication from
You. This License may not be modified without the mutual written
agreement of the Licensor and You. f. The rights granted under,
and the subject matter referenced, in this License were drafted
utilizing the terminology of the Berne Convention for the
Protection of Literary and Artistic Works (as amended on
September

28, 1979), the Rome Convention of 1961, the WIPO Copyright
Treaty
of 1996, the WIPO Performances and Phonograms Treaty of 1996 and
the Universal Copyright Convention (as revised on July 24,
1971). These rights and subject matter take effect in the
relevant
jurisdiction in which the License terms are sought to be
enforced
according to the corresponding provisions of the implementation
of
those treaty provisions in the applicable national law. If the
standard suite of rights granted under applicable copyright law
includes additional rights not granted under this License, such
additional rights are deemed to be included in the License; this
License is not intended to restrict the license of any rights
under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no
warranty whatsoever in connection with the Work. Creative
Commons
will not be liable to You or any party on any legal theory for
any damages whatsoever, including without limitation any
general,
special, incidental or consequential damages arising in
connection to this license. Notwithstanding the foregoing two
(2)
sentences, if Creative Commons has expressly identified itself
as
the Licensor hereunder, it shall have all rights and
obligations
of Licensor.

Except for the limited purpose of indicating to the public that
the Work is licensed under the CCPL, Creative Commons does not
authorize the use by either party of the trademark "Creative
Commons" or any related trademark or logo of Creative Commons
without the prior written consent of Creative Commons. Any
permitted use will be in compliance with Creative Commons'
then-current trademark usage guidelines, as may be published on
its website or otherwise made available upon request from time
to
time. For the avoidance of doubt, this trademark restriction
does not form part of this License.

Creative Commons may be contacted at
http://creativecommons.org/

```
    -- end printout --
```

Build System Architecture

1. Build Architecture

The asterisk build architecture relies on autoconf to detect the system configuration, and on a locally developed tool (menuselect) to select build options and modules list, and on gmake to do the build.

The first step, usually to be done soon after a checkout, is running "./configure", which will store its findings in two files:

- include/asterisk/autoconfig.h
    - contains C macros, normally #define HAVE_FOO or HAVE_FOO_H , for all functions and headers that have been detected at build time. These are meant to be used by C or C++ source files.

- makeopts
    - contains variables that can be used by Makefiles. In addition to the usual CC, LD, ... variables pointing to the various build tools, and prefix, includedir ... which are useful for generic compiler flags, there are variables for each package detected. These are normally of the form FOO_INCLUDE=... FOO_LIB=... FOO_DIR=... indicating, for each package, the useful libraries and header files.

The next step is to run "make menuselect", to extract the dependencies existing between files and modules, and to store build options.
menuselect produces two files, both to be read by the Makefile:

- menuselect.makeopts
    - Contains for each subdirectory a list of modules that must be excluded from the build, plus some additional informatiom.

- menuselect.makedeps
    - Contains, for each module, a list of packages it depends on. For each of these packages, we can collect the relevant INCLUDE and LIB files from makeopts. This file is based on information in the .c source code files for each module.

The top level Makefile is in charge of setting up the build environment, creating header files with build options, and recursively invoking the
subdir Makefiles to produce modules and the main executable.

The sources are split in multiple directories, more or less divided by module type (apps/ channels/ funcs/ res/ ...) or by function, for the main binary (main/ pbx/).

⊖ This section is not complete...

2. Adding new modules

2.1. Modules with no dependencies

If your module has no dependencies on external libraries, it can be dropped into the appropriate directory and the build system will automatically detect it and build it for you. For example, if you write `app_custom.c`, drop it in the `apps/` directory.

2.2. Modules with dependencies

If your module has an external dependency, you have a bit more work to do to integrate it into the build system. First, you must determine whether the library you are using is used by any other Asterisk module. If not, you must add the library to the following files.

2.2.1. `configure.ac`

This is the source of the configure script. There are *many* examples to draw from in this file. Search for the two instances of `ALSA` in this file to find one of the simpler examples of checking for a library.

2.2.2. `build_tools/menuselect-deps.in`

Add a line in this file for your new library. The configure script uses this file as input and outputs build_tools/menuselect-deps. The `menuselect` utility reads in this file to know which libraries have been found on the system.

2.2.3. `makeopts.in`

Add `LIBNAME_LIB` and `LIBNAME_INCLUDE` lines into this file. The configure script will use this file as input and output the `makeopts` file. When this library is found, these lines will have the appropriate `CFLAGS` and `LDFLAGS` needed to build and link a module with the library. Follow the other examples that are already in this file for formatting.

2.2.4. Update your module

Modules that have a dependency must have a special comment block in them that is used by the build system.

```
/*** MODULEINFO
        <depend>libname</depend>
 ***/
```

Coding Guidelines

1. Introduction

This document gives some basic indication on how the asterisk code is structured. Please read it to the end to understand in detail how the asterisk code is organized, and to know how to extend asterisk or contribute new code.

We are looking forward to your contributions to Asterisk - the Open Source PBX! As Asterisk is a large and in some parts very time-sensitive application, the code base needs to conform to a common set of coding rules so that many developers can enhance and maintain the code. Code also needs to be reviewed and tested so that it works and follows the general architecture and guide-lines, and is well documented.

Asterisk is published under a dual-licensing scheme by Digium. To be accepted into the codebase, all non-trivial changes must be licensed to Digium. For more information, see the electronic license agreement on https://issues.asterisk.org/.

Patches should be in the form of a unified (-u) diff, made from a checkout from subversion.

```
/usr/src/asterisk$ svn diff > mypatch
```

If you would like to only include changes to certain files in the patch, you can list them in the "svn diff" command:

```
/usr/src/asterisk$ svn diff somefile.c someotherfile.c > mypatch
```

2. Guidelines

2.1. General Rules

- Indent code using tabs, not spaces.

- All code, filenames, function names and comments must be in ENGLISH.

- Don't annotate your changes with comments like "/* JMG 4/20/04 */"; Comments should explain what the code does, not when something was changed or who changed it. If you have done a larger contribution, make sure that you are added to the CREDITS file.

- Don't make unnecessary whitespace changes throughout the code. If you make changes, submit them to the tracker as separate patches that only include whitespace and formatting changes.

- Don't use C++ type (//) comments.

- Try to match the existing formatting of the file you are working on.

- Use spaces instead of tabs when aligning in-line comments or #defines (this makes your comments aligned even if the code is viewed with another tabsize)

2.2. File structure and header inclusion

Every C source file should start with a proper copyright and a brief description of the content of the file. Following that, you should immediately put the following lines:

```
#include "asterisk.h"
ASTERISK_FILE_VERSION(__FILE__, "$Revision: 231369 $")
```

"asterisk.h" resolves OS and compiler dependencies for the basic set of unix functions (data types, system calls, basic I/O libraries) and the basic Asterisk APIs. ASTERISK_FILE_VERSION() stores in the executable information about the file.

Next, you should #include extra headers according to the functionality that your file uses or implements. For each group of functions that you use there is a common header, which covers OS header dependencies and defines the 'external' API of those functions (the equivalent of 'public' members of a class). As an example:

- asterisk/module.h
  **if you are implementing a module, this should be included in one of the files that are linked with the module.

- asterisk/io.h
  - access to extra file I/O functions (stat, fstat, playing with directories etc)

- asterisk/network.h
  - basic network I/O - all of the socket library, select/poll, and asterisk-specific (usually either thread-safe or reentrant or both) functions to play with socket addresses.

- asterisk/app.h
  - parsing of application arguments

- asterisk/channel.h
  - struct ast_channel and functions to manipulate it

For more information look at the headers in include/asterisk/. These files are usually self-sufficient, i.e. they recursively #include all the extra headers they need.

The equivalent of 'private' members of a class are either directly in the C source file, or in files named asterisk/mod_*.h to make it clear that they are not for inclusion by generic code.

Keep the number of header files small by not including them unnecessarily. Don't cut&paste list of header files from other sources, but only include those you really need. Apart from obvious

cases (e.g. module.h which is almost always necessary) write a short comment next to each #include to explain why you need it.

2.3. Declaration of functions and variables

- Do not declare variables mid-block (e.g. like recent GNU compilers support) since it is harder to read and not portable to GCC 2.95 and others.

- Functions and variables that are not intended to be used outside the module must be declared static. If you are compiling on a Linux platform that has the 'dwarves' package available, you can use the 'pglobal' tool from that package to check for unintended global variables or functions being exposed in your object files. Usage is very simple:

```
$ pglobal \-vf <path to .o file>
```

- When reading integer numeric input with scanf (or variants), do *NOT* use '%i' unless you specifically want to allow non-base-10 input; '%d' is always a better choice, since it will not silently turn numbers with leading zeros into base-8.

- Strings that are coming from input should not be used as the format argument to any printf-style function.

2.4. Structure alignment and padding

On many platforms, structure fields (in structures that are not marked 'packed') will be laid out by the compiler with gaps (padding) between them, in order to satisfy alignment requirements. As a simple example:

```
struct foo {
        int bar;
        void *xyz;
}
```

On nearly every 64-bit platform, this will result in 4 bytes of dead space between 'bar' and 'xyz', because pointers on 64-bit platforms must be aligned on 8-byte boundaries. Once you have your code written and tested, it may be worthwhile to review your structure definitions to look for problems of this nature. If you are on a Linux platform with the 'dwarves' package available, the 'pahole' tool from that package can be used to both check for padding issues of this type and also propose reorganized structure definitions to eliminate it. Usage is quite simple; for a structure named 'foo', the command would look something like this:

```
$ pahole \--reorganize \--show_reorg_steps \-C foo <path to module>
```

The 'pahole' tool has many other modes available, including some that will list all the structures declared in the module and the amount of padding in each one that could possibly be recovered.

2.5. Use the internal API

Make sure you are aware of the string and data handling functions that exist within Asterisk to enhance portability and in some cases to produce more secure and thread-safe code. Check utils.c/utils.h for these.

If you need to create a detached thread, use the ast_pthread_create_detached() normally or ast_pthread_create_detached_background() for a thread with a smaller stack size. This reduces the replication of the code to handle the pthread_attr_t structure.

2.6. Code formatting

Roughly, Asterisk code formatting guidelines are generally equivalent to the following:

```
\# indent \-i4 \-ts4 \-br \-brs \-cdw \-lp \-ce \-nbfda \-npcs
\-nprs \-npsl \-nbbo \-saf \-sai \-saw \-cs \-l90 foo.c
```

this means in verbose:

```
-i4:     indent level 4
 -ts4:    tab size 4
 -br:     braces on if line
 -brs:    braces on struct decl line
 -cdw:    cuddle do while
 -lp:     line up continuation below parenthesis
 -ce:     cuddle else
 -nbfda: dont break function decl args
 -npcs:  no space after function call names
 -nprs:  no space after parentheses
 -npsl:  dont break procedure type
 -saf:    space after for
 -sai:    space after if
 -saw:    space after while
 -cs:     space after cast
 -l90:   line length 90 columns
```

Function calls and arguments should be spaced in a consistent way across the codebase.

```
GOOD: foo(arg1, arg2);
 BAD: foo(arg1,arg2);
 BAD: foo (arg1, arg2);
 BAD: foo( arg1, arg2 );
 BAD: foo(arg1, arg2,arg3);
```

Don't treat keywords (if, while, do, return) as if they were functions; leave space between the keyword and the expression used (if any). For 'return', don't even put parentheses around the expression, since they are not required.

There is no shortage of whitespace characters 🙂 Use them when they make the code easier to read. For example:

```
for (str=foo;str;str=str->next)
```

is harder to read than

```
for (str = foo; str; str = str->next)
```

Following are examples of how code should be formatted.

2.6.1. Functions:

```
int foo(int a, char *s)
{
 return 0;
}
```

2.6.2. If statements:

```
if (foo) {
 bar();
} else {
 blah();
}
```

2.6.3. Case statements:

```
switch (foo) {
case BAR:
 blah();
 break;
case OTHER:
 other();
 break;
}
```

2.6.4. No nested statements without braces

```
for (x = 0; x < 5; x++)
 if (foo)
   if (bar)
    baz();
```

Instead, do:

```
for (x = 0; x < 5; x++) {
 if (foo) {
   if (bar) {
    baz();
   }
  }
 }
```

Always use braces around the statements following an if/for/while construct, even if not strictly necessary, as it reduces future possible problems.

Don't build code like this:

```
if (foo) {
 /* .... 50 lines of code ... */
} else {
 result = 0;
 return;
}
```

Instead, try to minimize the number of lines of code that need to be indented, by only indenting the shortest case of the 'if' statement, like so:

```
if (!foo) {
 result = 0;
 return;
}

.... 50 lines of code ....
```

When this technique is used properly, it makes functions much easier to read and follow, especially those with more than one or two 'setup' operations that must succeed for the rest of the function to be able to execute.

2.7. Labels/goto are acceptable

Proper use of this technique may occasionally result in the need for a label/goto combination so that error/failure conditions can exit the
function while still performing proper cleanup. This is not a bad thing! Use of goto in this situation is encouraged, since it removes the need
for excess code indenting without requiring duplication of cleanup code.

2.8. Do not cast 'void *'

Do not explicitly cast 'void *' into any other type, nor should you cast any other type into 'void *'.

Implicit casts to/from 'void *' are explicitly allowed by the C specification. This means the results of malloc(), calloc(), alloca(), and similar functions do not *ever* need to be cast to a specific type, and when you are passing a pointer to (for example) a callback function that accepts a 'void *' you do not need to cast into that type.

2.9. Function naming

All public functions (those not marked 'static'), must be named "ast_<something>" and have a descriptive name.

As an example, suppose you wanted to take a local function "find_feature", defined as static in a file, and used only in that file, and make it public, and use it in other files. You will have to remove the "static" declaration and define a prototype in an appropriate header file (usually in include/asterisk). A more specific name should be given, such as "ast_find_call_feature".

2.10. Variable function argument parsing

Functions with a variable amount of arguments need a 'sentinel' when called. Newer GNU C compilers are fine if you use NULL for this. Older versions (pre 4) don't like this. You should use the constant SENTINEL. This one is defined in include/asterisk/compiler.h

2.11. Variable naming

2.11.1. Global variables

Name global variables (or local variables when you have a lot of them or are in a long function) something that will make sense to aliens who
find your code in 100 years. All variable names should be in lower case, except when following external APIs or specifications that normally
use upper- or mixed-case variable names; in that situation, it is preferable to follow the external API/specification for ease of understanding.

Make some indication in the name of global variables which represent options that they are in fact intended to be global.
e.g.: `static char global_something[80]`

2.12. Don't use unnecessary typedef's

Don't use 'typedef' just to shorten the amount of typing; there is no substantial benefit in this:
`struct foo { int bar; }; typedef struct foo foo_t;`

In fact, don't use 'variable type' suffixes at all; it's much preferable to just type 'struct foo' rather than 'foo_s'.

2.13. Use enums instead of #define where possible

Use enums rather than long lists of #define-d numeric constants when possible; this allows structure members, local variables and function arguments to be declared as using the enum's type. For example:

```
enum option {
  OPT_FOO = 1,
  OPT_BAR = 2,
  OPT_BAZ = 4,
};

static enum option global_option;

static handle_option(const enum option opt)
{
  ...
}
```

Note: The compiler will *not* force you to pass an entry from the enum as an argument to this function; this recommendation serves only to make
the code clearer and somewhat self-documenting. In addition, when using switch/case blocks that switch on enum values, the compiler will warn you if you forget to handle one or more of the enum values, which can be handy.

2.14. String handling

Don't use strncpy for copying whole strings; it does not guarantee that the output buffer will be null-terminated. Use ast_copy_string instead, which is also slightly more efficient (and allows passing the actual buffer size, which makes the code clearer).

Don't use ast_copy_string (or any length-limited copy function) for copying fixed (known at compile time) strings into buffers, if the buffer is something that has been allocated in the function doing the copying. In that case, you know at the time you are writing the code whether the buffer is large enough for the fixed string or not, and if it's not, your code won't work anyway! Use strcpy() for this operation, or directly set the first two characters of the buffer if you are just trying to store a one character string in the buffer. If you are trying to 'empty' the buffer, just store a single NULL character ('\0') in the first byte of the buffer; nothing else is needed, and any other method is wasteful.

In addition, if the previous operations in the function have already determined that the buffer in use is adequately sized to hold the string
you wish to put into it (even if you did not allocate the buffer yourself), use a direct strcpy(), as it can be inlined and optimized to simple
processor operations, unlike ast_copy_string().

2.15. String conversions

When converting from strings to integers or floats, use the sscanf function in preference to the atoi and atof family of functions, as sscanf detects errors. Always check the return value of sscanf to verify that your numeric variables successfully scanned before using them. Also, to

avoid a potential libc bug, always specify a maximum width for each conversion specifier, including integers and floats. A good length for both integers and floats is 30, as this is more than generous, even if you're using doubles or long integers.

2.16. Use of functions

For the sake of uclibc, do not use index, bcopy or bzero; use strchr(), memset(), and memmove() instead. uclibc can be configured to supply these functions, but we can save these users time and consternation if we abstain from using these functions.

When making applications, always ast_strdupa(data) to a local pointer if you intend to parse the incoming data string.

```
if (data) {
  mydata = ast_strdupa(data);
 }
```

Use the argument parsing macros to declare arguments and parse them, i.e.:

```
AST_DECLARE_APP_ARGS(args,
   AST_APP_ARG(arg1);
   AST_APP_ARG(arg2);
   AST_APP_ARG(arg3);
 );
 parse = ast_strdupa(data);
 AST_STANDARD_APP_ARGS(args, parse);
```

Make sure you are not duplicating any functionality already found in an API call somewhere. If you are duplicating functionality found in
another static function, consider the value of creating a new API call which can be shared.

2.17. Handling of pointers and allocations

2.17.1. Use const on pointer arguments if possible

Use const on pointer arguments which your function will not be modifying, as this allows the compiler to make certain optimizations. In general, use 'const' on any argument that you have no direct intention of modifying, as it can catch logic/typing errors in your code when you use the argument variable in a way that you did not intend.

2.17.2. Do not create your own linked list code - reuse!

As a common example of this point, make an effort to use the lockable linked-list macros found in include/asterisk/linkedlists.h. They are
efficient, easy to use and provide every operation that should be necessary for managing a singly-linked list (if something is missing,
let us know!). Just because you see other open-coded list implementations in the source tree is

no reason to continue making new copies of
that code... There are also a number of common string manipulation and timeval manipulation
functions in asterisk/strings.h and asterisk/time.h;
use them when possible.

2.17.3. Allocations for structures

When allocating/zeroing memory for a structure, use code like this:

```
struct foo *tmp;

...

tmp = ast_calloc(1, sizeof(*tmp));
```

Avoid the combination of ast_malloc() and memset(). Instead, always use ast_calloc(). This will
allocate and zero the memory in a single operation. In the case that uninitialized memory is
acceptable, there should be a comment in the code that states why this is the case.

Using sizeof(*tmp) instead of sizeof(struct foo) eliminates duplication of the 'struct foo' identifier,
which makes the code easier to read and also ensures that if it is copy-and-pasted it won't
require as much editing.

The ast_* family of functions for memory allocation are functionally the same. They just add an
Asterisk log error message in the case that the allocation fails for some reason. This eliminates
the need to generate custom messages throughout the code to log that this has occurred.

2.17.4. String Duplications

The functions strdup and strndup can **not** accept a NULL argument. This results in having code
like this:

```
if (str) {
  newstr = strdup(str);
 } else {
  newstr = NULL;
 }
```

However, the ast_strdup and ast_strdupa functions will happily accept a NULL
argument without generating an error. The same code can be written as:

```
newstr = ast_strdup(str);
```

Furthermore, it is unnecessary to have code that malloc/calloc's for the length of a string (+1 for
the terminating '\0') and then using strncpy to copy the copy the string into the resulting buffer.
This is the exact same thing as using ast_strdup.

2.18. CLI Commands

New CLI commands should be named using the module's name, followed by a verb and then
any parameters that the command needs. For example:

```
*CLI> iax2 show peer <peername>
```

not

```
*CLI> show iax2 peer <peername>
```

2.19. New dialplan applications/functions

There are two methods of adding functionality to the Asterisk dialplan: applications and functions.
Applications (found generally in
the apps/ directory) should be collections of code that interact with a channel and/or user in some
significant way. Functions (which can be
provided by any type of module) are used when the provided functionality is simple...
getting/retrieving a value, for example. Functions should also be used when the operation is in no
way related to a channel (a computation or string operation, for example).

Applications are registered and invoked using the ast_register_application function; see the
apps/app_skel.c file for an example.

Functions are registered using 'struct ast_custom_function' structures and the
ast_custom_function_register function.

2.20. Doxygen API Documentation Guidelines

When writing Asterisk API documentation the following format should be followed. Do not use the
javadoc style.

```
/*!
 * \brief Do interesting stuff.
 *
 * \param thing1 interesting parameter 1.
 * \param thing2 interesting parameter 2.
 *
 * This function does some interesting stuff.
 *
 * \retval zero on success
 * \retval -1 on error.
 */
int ast_interesting_stuff(int thing1, int thing2)
{
 return 0;
}
```

Notice the use of the \param, \brief, and \return constructs. These should be used to describe the corresponding pieces of the function being documented. Also notice the blank line after the last \param directive. All doxygen comments must be in one /*! */ block. If the function or struct does not need an extended description it can be left out.

Please make sure to review the doxygen manual and make liberal use of the \a, \code, \c, \b, \note, \li and \e modifiers as appropriate.

When documenting a 'static' function or an internal structure in a module, use the \internal modifier to ensure that the resulting documentation explicitly says 'for internal use only'.

When adding new API you should also attach a \since note because this will indicate to developers that this API did not exist before this version. It also has the benefit of making the resulting HTML documentation to group the changes for a single version.

Structures should be documented as follows.

```
/*!
 * \brief A very interesting structure.
 */
struct interesting_struct
{
 /*! \brief A data member. */
 int member1;

 int member2; /*!< \brief Another data member. */
}
```

Note that **/! */ blocks document the construct immediately following them unless they are**

written, /!< */, in which case they document the construct preceding them.

It is very much preferred that documentation is not done inline, as done in the previous example for member2. The first reason for this is that it tends to encourage extremely brief, and often pointless, documentation since people try to keep the comment from making the line extremely long. However, if you insist on using inline comments, please indent the documentation with spaces! That way, all of the comments are properly aligned, regardless of what tab size is being used for viewing the code.

2.21. Finishing up before you submit your code

Before submitting a patch, **read** the actual patch file to be sure that all the changes you expect to be there are, and that there are no
surprising changes you did not expect. During your development, that part of Asterisk may have changed, so make sure you compare with the latest SVN.

If you are asked to make changes to your patch, there is a good chance the changes will introduce bugs, check it even more at this stage.
Also remember that the bug marshal or co-developer that adds comments is only human, they may be in error 🙂

2.22. Creating new manager events?

If you create new AMI events, please read manager.txt. Do not re-use existing headers for new purposes, but please re-use existing headers
for the same type of data.

Manager events that signal a status are required to have one event name, with a status header that shows the status. The old style, with one event named "ThisEventOn" and another named "ThisEventOff", is no longer approved.

Check manager.txt for more information on manager and existing headers. Please update this file if you add new headers.

Janitor Projects

What is this page?

This page contains janitor (small, cleanup) projects for Asterisk that are waiting for a kind developer to take them on and bring them to completion.

- Audit uses of usleep() to ensure that the argument is never greater than 1 million. On some systems, that is considered an error. In any such cases, convert the usage ver to use nanosleep(), instead.

- Convert all existing uses of astobj.h to astobj2.h. chan_sip already in progress in a branch

- There are many places where large character buffers are allocated in structures. There is a new system for string handling that uses dynamically allocatted memory pools which is documented in include/asterisk/stringfields.h. Examples of where they are currently used are the ast_channel structure defined in include/asterisk/channel.h, some structures in chan_sip.c, and chan_dahdi.c.

- There is a convenient set of macros defined in include/asterisk/linkedlists.h for handling linked lists. However, there are some open-coded

lists throughout the code. Converting linked lists to use these macros will make list handling more consistent and reduce the possibility of coding errors.

- Clean up and add Doxygen Documentation. When generating the documentation with make progdocs, a lot of warnings are generated. All of these need to be fixed. There is also plenty of code that still needs to be documented. All public API functions should be documented. That is pretty much anything in include/asterisk/*.h.

- Check all ast_copy_string() usage to ensure that buffers are not being unnecessarily zeroed before or after calling it.

- Find any remaining open-coded struct timeval manipulation and convert to use new time library functions.

- Use the ast_str API in strings.h to replace multiple calls to strncat(), snprintf() with funky math, etc.

- Audit all channel/res/app/etc. modules to ensure that they do not register any entrypoints with the Asterisk core until after they are ready to service requests; all config file reading/processing, structure allocation, etc. must be completed before Asterisk is made aware of any services the module offers.

- Ensure that Realtime-enabled modules do not depend on the order of columns returned by the database lookup (example: outboundproxy and host settings in chan_sip).

- Convert all usage of the signal(2) system API to the more portable sigaction(2) system API.

- Find options and arguments in Asterisk which specify a time period in seconds or milliseconds and convert them to use the new ast_app_parse_timelen() function.

- Find applications and functions in Asterisk that would benefit from being able to encode control characters and extended ASCII and embed calls to ast_get_encoded_char, ast_get_encoded_str, and ast_str_get_encoded_str.

Locking in Asterisk

Locking Fundamentals

Asterisk is a heavily multithreaded application. It makes extensive use of locking to ensure safe access to shared resources between different threads.

When more that one lock is involved in a given code path, there is the potential for deadlocks. A deadlock occurs when a thread is stuck waiting for a resource that it will never acquire. Here is a classic example of a deadlock:

```
Thread 1                        Thread 2
   -----------                     -----------
   Holds Lock A                    Holds Lock B
   Waiting for Lock B              Waiting for Lock A
```
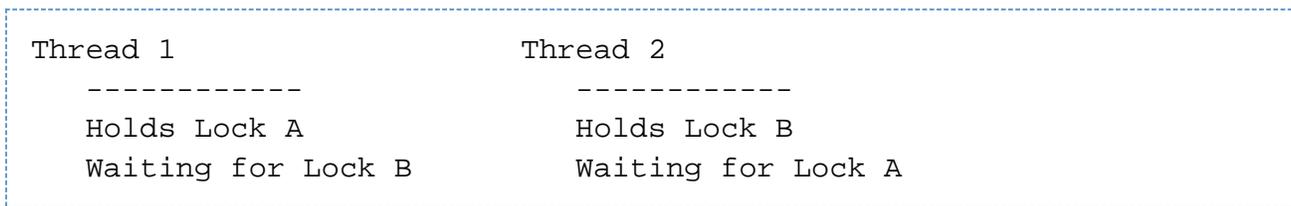
In this case, there is a deadlock between threads 1 and 2. This deadlock would have been avoided if both threads had agreed that one must acquire Lock A before Lock B.

In general, the fundamental rule for dealing with multiple locks is: **an order *must* be established to acquire locks, and then all threads must respect that order when acquiring locks**.

Establishing a locking order

Because any ordering for acquiring locks is ok, one could establish the rule arbitrarily, e.g. ordering by address, or by some other criterion.
The main issue, though, is defining an order that

1. is easy to check at runtime;
2. reflects the order in which the code executes.

As an example, if a data structure B is only accessible through a data structure A, and both require locking, then the natural order is locking first A and then B. As another example, if we have some unrelated data structures to be locked in pairs, then a possible order can be based on the address of the data structures themselves.

Minding the boundary between channel drivers and the Asterisk core

The #1 cause of deadlocks in Asterisk is by not properly following the locking rules that exist at the boundary between Channel Drivers and
the Asterisk core. The Asterisk core allocates an ast_channel, and Channel Drivers allocate "technology specific private data" (PVT) that is
associated with an ast_channel. Typically, both the ast_channel and PVT have their own lock. There are *many* code paths that require both objects to be locked.

The locking order in this situation is the following:

1. ast_channel
2. PVT

Channel Drivers implement the ast_channel_tech interface to provide a channel implementation for Asterisk. Most of the channel_tech
interface callbacks are called with the associated ast_channel locked. When accessing technology specific data, the PVT can be locked directly because the locking order is respected.

Preventing lock ordering reversals.

There are some code paths which make it extremely difficult to respect the locking order. Consider for example the following situation:

1. A message comes in over the "network"
2. The Channel Driver (CD) monitor thread receives the message
3. The CD associates the message with a PVT and locks the PVT
4. While processing the message, the CD must do something that requires locking the ast_channel associated to the PVT

This is the point that must be handled carefully.

The following psuedo-code

```
unlock(pvt);
        lock(ast_channel);
        lock(pvt);
```

is *not* correct for two reasons:

1. first and foremost, unlocking the PVT means that other threads can acquire the lock and believe it is safe to modify the associated data. When reacquiring the lock, the original thread might find unexpected changes in the protected data structures. This essentially means that the original thread must behave as if the lock on the pvt was not held, in which case it could have released it itself altogether;
2. Asterisk uses the so called "recursive" locks, which allow a thread to issue a lock() call multiple times on the same lock. Recursive locks count the number of calls, and they require an equivalent number of unlock() to be actually released.

For this reason, just calling unlock() once does not guarantee that the lock is actually released – it all depends on how many times lock() was called before.

An alternative, but still incorrect, construct is widely used in the asterisk code to try and improve the situation:

```
while (trylock(ast_channel) == FAILURE) {
        unlock(pvt);
        usleep(1); /* yield to other threads */
        lock(pvt);
    }
```

Here the trylock() is non blocking, so we do not deadlock if the ast_channel is already locked by someone else: in this case, we try to unlock the PVT (which happens only if the PVT lock counter is 1), yield the CPU to give other threads a chance to run, and then acquire the lock again.

This code is not correct for two reasons:

1. same as in the previous example, it releases the lock when the thread probably did not expect it;
2. if the PVT lock counter is greater than 1 we will not really release the lock on the PVT. We might be lucky and have the other contender actually release the lock itself, and so we will "win" the race, but if both contenders have their lock counts > 1 then they will loop forever (basically replacing deadlock with livelock).

Another variant of this code is the following:

```
if (trylock(ast_channel) == FAILURE) {
        unlock(pvt);
        lock(ast_channel);
        lock(pvt);
    }
```

which has the same issues as the while(trylock...) code, but just deadlocks instead of looping forever in case of lock counts > 1.

The deadlock/livelock could be in principle spared if one had an unlock_all() function that calls unlock as many times as needed to actually release the lock, and reports the count. Then we could do:

```
if (trylock(ast_channel) == FAILURE) {
        n = unlock_all(pvt);
        lock(ast_channel)
        while (n-- > 0) lock(pvt);
    }
```

The issue with unexpected unlocks remains, though.

Locking multiple channels.

The next situation to consider is what to do when you need a lock on multiple ast_channels (or multiple unrelated data structures).

If we are sure that we do not hold any of these locks, then the following construct is sufficient:

```
lock(MIN(chan1, chan2));
        lock(MAX(chan1, chan2));
```

That type of code would follow an established locking order of always locking the channel that has a lower address first. Also keep in mind
that to use this construct for channel locking, one would have to go through the entire codebase to ensure that when two channels are locked,
this locking order is used.

However, if we enter the above section of code with some lock held (which would be incorrect using non-recursive locks, but is completely legal using recursive mutexes) then the locking order is not guaranteed anymore because it depends on which locks we already hold. So we have to go through the same tricks used for the channel+PVT case.

Recommendations

As you can see from the above discussion, getting locking right is all but easy. So please follow these recommendations when using locks:

Use locks only when really necessary

Please try to use locks only when strictly necessary, and only for the minimum amount of time required to run critical sections of code. A common use of locks in the current code is to protect a data structure from being released while you use it. With the use of reference-counted objects (astobj2) this should not be necessary anymore.

There is only one time in the entire code base where deadlock avoidance is absolutely necessary and that is when multiple channels are being locked at the same time. This is because there is no locking order established for locking multiple channels.

With everything else, there is a locking order established. In this case the channels container must always be locked before an individual channel. Deadlock avoidance is not necessary here.

So, never use deadlock avoidance unless you have to grab two channel locks at the same time, otherwise it is not necessary. I know the code has misuses of deadlock avoidance all over the place, ignore them. They will go away in time.

> ⚠ As a side note, if you ever find yourself in the position where you are designing a program and think using deadlock avoidance is a quick solution for a concurrency problem you run into, it is not. Running into a deadlock almost certainly means your design is flawed. I'd go as far as to say if you ever find yourself designing a system with multiple locks held at the same time, your design is flawed. Seriously, anyone who reads this remember this, and say it to yourself every time you create a new mutex.
> – David Vossel

Do not sleep while holding a lock

If possible, do not run any blocking code while holding a lock, because you will also block other threads trying to access the same lock. In many cases, you can hold a reference to the object to avoid that it is deleted while you sleep, perhaps set a flag in the object itself to report other threads that you have some pending work to complete, then release and acquire the lock around the blocking path, checking the status of the object after you acquire the lock to make sure that you can still perform the operation you wanted to.

Try not to exploit the 'recursive' feature of locks.

Recursive locks are very convenient when coding, as you don't have to worry, when entering a section of code, whether or not you already hold the lock – you can just protect the section with a lock/unlock pair and let the lock counter track things for you.
But as you have seen, exploiting the features of recursive locks make it a lot harder to implement proper deadlock avoidance strategies. So please try to analyse your code and determine statically whether you already hold a lock when entering a section of code.
If you need to call some function foo() with and without a lock held, you could define two function as below:

```
foo_locked(...) {
        ... do something, assume lock held
      }

      foo(...) {
          lock(xyz)
          ret = foo_locked(...)
          unlock(xyz)
          return ret;
      }
```

and call them according to the needs.

Document locking rules.

Please document the locking order rules are documented for every lock introduced into Asterisk. This is done almost nowhere in the existing code. However, it will be expected to be there for newly introduced code. Over time, this information should be added for all of the existing lock usage.

Measuring SIP Channel Performance

Measuring the SIP channel driver's Performance

This file documents the methods I used to measure the performance of the SIP channel driver, in terms of maximum simultaneous calls and how quickly it could handle incoming calls.

Knowing these limitations can be valuable to those implementing PBX's in 'large' environments. Will your installation handle expected call volume?

Quoting these numbers can be totally useless for other installations. Minor changes like the amount of RAM in a system, the speed of the ethernet, the amount of cache in the CPU, the CPU clock speed, whether or not you log CDR's, etc. can affect the numbers greatly.

In my set up, I had a dedicated test machine running Asterisk, and another machine which ran sipp, connected together with
ethernet.

The version of sipp that I used was sipp-2.0.1; however, I have reason to believe that other versions would work just as well.

On the asterisk machine, I included the following in my extensions.ael file:

```
context test11
{
        s => {
                Answer();
                while (1) {
                        Background(demo-instruct);
                }
                Hangup();
        }
        _X. => {
                Answer();
                while (1) {
                        Background(demo-instruct);
                }
                Hangup();
        }
}
```

Basically, incoming SIP calls are answered, and the demo-instruct sound file is played endlessly to the caller. This test depends on the calling party to hang up, thus allowing sipp to determine the length of a call.

The sip.conf file has this entry:

```
[asterisk02]
type=friend
context=test11
host=192.168.134.240 ;; the address of the host you will be running
sipp on
user=sipp
directmedia=no
disallow=all
allow=ulaw
```

Note that it's pretty simplistic; no authentication beyond the host ip, and it uses ulaw, which is pretty efficient, low-cpu-intensive codec.

To measure the impact of incoming call traffic on the Asterisk machine, I run vmstat. It gives me an idea of the cpu usage by Asterisk. The most common failure mode of Asterisk at high call volumes, is that the CPU reaches 100% utilization, and then cannot keep up with the workload, resulting in timeouts and other failures, which swiftly compound and cascade, until gross failure ensues. Watch the CPU Idle % numbers.

I learned to split the testing into two modes: one for just call call processing power, in the which we had relatively few simultaneous calls in place, and another where we allow the the number of simultaneous calls to quickly reach a set maximum, and then rerun sipp, looking for the maximum.

Call processing power is measured with extremely short duration calls:

```
./sipp -sn uac 192.168.134.252 -s 12 -d 100 -l 256
```

The above tells sipp to call your asterisk test machine (192.168.134.252) at extension 12, each call lasts just .1 second, with a limit of 256 simultaneous calls. The simultaneous calls will be the rate/sec of incoming calls times the call length, so 1 simultaneous call at 10 calls/sec, and 45 at 450 calls/sec. Setting the limit to 256 implies you do not intend to test above 2560 calls/sec.

Sipp starts at 10 calls/sec, and you can slowly increase the speed by hitting '*' or '+'. Watch your cpu utilization on the asterisk server. When you approach 100%, you have found your limit.

Simultaneous calls can be measured with very long duration calls:

```
./sipp -sn uac 192.168.134.252 -s 12 -d 100000 -l 270
```

This will place 100 sec duration calls to Asterisk. The number of simultaneous calls will increase until the maximum of 270 is reached. If Asterisk survives this number and is not at 100% cpu utilization, you can stop sipp and run it again with a higher -l argument.

By changing one Asterisk parameter at a time, you can get a feel for how much that change will

affect performance.

Modules

All modules must have at least the following:

```
static int load_module():
```

Do what you need to do when you get started. This function can return AST_MODULE_LOAD_FAILURE if an action fails and the module is prevented from loading, AST_MODULE_LOAD_DECLINE if the module can not load because of a non-critical failure (the configuration file was not found), or AST_MODULE_LOAD_SUCCESS if the module loaded fine.

```
static int unload_module():
```

The module will soon be unloaded. If any channels are using your features, you should give them a softhangup in an effort to keep the program from crashing. Generally, unload_module is only called when the usecount is 0 or less, but the user can force unloading at their discretion, and thus a module should do its best to comply (although in some cases there may be no way to avoid a crash). This function should return 0 on success and non-zero on failure (i.e. it cannot yet be unloaded).

```
AST_MODULE_INFO_STANDARD(keystr, desc);
```

- keystr - Applicable license for module. In most cases this is ASTERISK_GPL_KEY.
- desc - Description of module.

Confluence Tips

1. Page Setup Tips

1.1. Numbered Headings

To have the section headings automatically numbered, wrap the entire page in the {numberedheadings} macro.

For more information, see the home page for the Numbered Headings plugin.

The numbered headings plugins expects that the headings are used in order (h1, h2, h3, etc.). If you skip levels, it will not number the headings.

1.2. Table of Contents

Adding a table of contents to your page is easy. To do so, add the `{toc}` macro where you would like the table of contents to be inserted. This is useful in combination with numbered headings. Both are used on this page.

2. Formatting Tips

2.1. Configuration Examples

The `{code}` macro does a nice job with formatting a block for an Asterisk configuration example. Be sure to add a title, too by doing `{code:title=asterisk.conf}`. For example:

<div>

asterisk.conf

```
[section]

var=value
foo=bar
```

</div>

2.2. Code Examples

The `{code}` macro supports a bunch of different programming languages. See the notation guide for details. Tell the macro which programming language you're using (`{code:xml|title=example.xml}`) and it will do syntax highlighting for you. For a list of programming languages supported by this macro, see this page on the newcode macro.

<div>

example.xml

```
<?xml version="1.0"?>

<menu name="Asterisk Module and Build Option Selection">
    <category name="MENUSELECT_ADDONS" displayname="Add-ons (See
README-addons.txt)" remove_on_change="addons/modules.link">
        <member name="app_mysql" displayname="Simple Mysql
Interface" remove_on_change="addons/app_mysql.o
addons/app_mysql.so">
            <depend>mysqlclient</depend>
            <defaultenabled>no</defaultenabled>
        </member>
    </category>
</menu>
```

</div>

example.java

```java
class Example {

public static void main(String args[])
{
    System.out.println("Hello World");
}

}
```

example.cpp

```cpp
class Example {
public:
    Example();
    ~Example();

    void setFoo(int f);
    int getFoo(void) const;

private:
    int foo;
};
```

Roadmap

⊖   The content of this page is a draft and is subject to change at any time.

1. Related Links

- The Roadmap Page on the issue tracker has information about various issues and patches that have been targeted for a specific Asterisk release.

- The CHANGES file in Asterisk lists new features that have been added to each version.

2. Asterisk 1.10

Committed Features - developed by Digium

- Media Overhaul
- SIP Security Events
- T.38 Gateway
- Documentation Improvements

3. Future Development Wishlist

The items in this list primarily came from AstriDevCon 2010.

3.1. (P0)

P0 are committed projects that are going to get done.

- T.38 Gateway (Digium)
- Performance of State Change Processing (Stefan Schmidt)
    - (work is already being done on this front)
- SIP path support (Olle)
    - (first generation of code exists, needs more work, simple patch, going to get it done, needs an extra field in astdb; helps when there are 2 or more load balancing proxies in front of asterisk, when you'd like the call to be able to get back to Asterisk; see https://reviewboard.asterisk.org/r/991/)
    - https://issues.asterisk.org/view.php?id=18223
- Group variables (Kobaz)
    - (on review board, in progress)
- Pre-Dial (Kobaz)
    - (practically done, or something)
- Distributed extension state using SIP (Olle)
    - (resources in place, doing it, 1.4 done before Christmas, project pinana)
- Manager event docs (Paul Belanger)
- Cross-platform documentation (Ben Klang)
    - (caveats for using Asterisk on operating system xyz; pull a PDF of the Wiki documentation into the source, don't forget to include basic installation information, and do it all in .txt - Ben)
- Fix libs to optionally init OpenSSL (Digium)
    - (or use existing tools; sort of a bug)
- Make ast_channel an opaque type (Digium)

3.2. (P1)

P1 is the highest priority.

- Codecs (SILK, Opal), Media Negotiation (Digium)
- RTCP (Olle)
    - Pinefrog; Work to be done - Ported to trunk, added to CEL
- Conferencing that supports a new magic media (Digium)
    - higher sampling rates

3.3. (P2)

- Async DNS (TCP DNS and use a good resolver)
- Named ACLs (deluxepine)
- SIP Security Events
- Light weight means of holding NAT open in SIP (less complex than current qualify, Consider it done)
- IPv6 for the restivus (IAX, Jabber/XMPP/Gtalk, Manager, etc.)
- ConfBridge feature complete with MeetMe
- Support sound file containers (matroska)

3.4. (P3)

- Who hung up? (there's a branch, shouldn't take too much time - Olle)

- Unique identifier for filtering log data to a call
  - (finishing what was already begun w/ Clod's project, CLI filtering; should take a look at what Stephan from Unlimitel.ca's created)

3.5. (P4)

- Multiple SIP Sockets
  - (Listen on multiple ports or on multiple interfaces, but not all; also set binding for RTP)...alternate idea / solution would be to make Asterisk capable of loading multiple SIP profiles, it might be easier
- Multiple DNS results
  - (need to be able to traverse a list of DNS results, rather than just getting back one result)
- ICE-lite
  - (no code, responding correctly to ICE connectivity checks (STUN multiplexed on the RTP port) and understanding the SDP); it makes NAT traversal work for clients that do ICE; also addressed lightweight NAT refresh)

3.6. (P5)

- AstDB replacement
  - (realtime, there's code, nearly ready)
- SIP identity
  - (on reviewboard; needs to be forward ported; important for organizations w/ federated identities; a requirement for DTLS SRTP; not widely deployed)
- RTMP client channel driver

3.7. (P6)

- Structured identifiers for errors
  - (tag an error message with a unique string, specific to the error message and where it came from; should be alphanumeric to keep them short)
- AMI SetVar, Context limits
  - (there's code already...Olle has it)
- AMI filters on demand
- DTLS SRTP
  - (not likely to be widely deployed in the next 12 months)

3.8. (P7

- Asterisk register for XMPP account (Leif)
- Write a Specification for AMI (not kobaz)
- Multiple TLS server certs
  - (1 socket, requires support by OpenSSL; simpler to implement than multiple SIP profiles; don't know if any clients use it yet; needs more research)

3.9. (P8)

- Make resource modules that talk to DBs attempt reconnects
- Apple's new file streaming format, derived from .m3u
- Make MixMonitor and Monitor feature compatible

3.10. (P?, Research Required)

- New app_queue (as if? no, seriously? talking about this scares Russell)
- Identify and fix all bugs in AMI
- Broadsoft or Dialog Info shared line appearance (SLA) support
  - (Tabled for later discussion)
- LDAP from within the dialplan
  - (we may already have it, needs research to see if the realtime driver does what's desired - Leif)
- Device state normalization
- Anything DB over HTTP(s) with failover handling
- Use a channel as a MoH Source
- Kill Masquerades
- Bridging thread pool
- Threadify chan_sip
- Export ISDN ROSE information up to Asterisk channels

AstriDevCon 2010

1. Introduction

AstriDevCon 2010 was held on Friday, October 29th. It was held on the day following AstriCon at the same location. A group of active development community members met and discussed a number of topics, including:

- Asterisk SCF Ideas
- Asterisk 1.10 Ideas
- Asterisk Test Framework Ideas
- Asterisk Release Policy

2. Asterisk SCF Ideas

The Asterisk SCF discussion kicked off the morning. The topics that were covered included:

- VMWare/Etc. Image
- Channel driver
- ICE management tools.
- Queues
- Security, security, security
- Do we build the security functionality into the SCF code, or do we simply define best practices for securing an SCF system?

3. Asterisk 1.10 Ideas

We spent probably half of the day discussing Asterisk 1.10. We came up with a list of development projects we'd like to get done and then prioritized them.

3.1. (P0)

P0 are committed projects that are going to get done.

- T.38 Gateway (Digium)
- ~~Performance of State Change Processing (Stefan Schmidt)~~
    - ~~(work is already being done on this front)~~
- SIP path support (Olle)
    - (first generation of code exists, needs more work, simple patch, going to get it done, needs an extra field in astdb; helps when there are 2 or more load balancing proxies in front of asterisk, when you'd like the call to be able to get back to Asterisk; see https://reviewboard.asterisk.org/r/991/)
    - https://issues.asterisk.org/view.php?id=18223
- Group variables (Kobaz)
    - (on review board, in progress)
- Pre-Dial (Kobaz)
    - (practically done, or something)
- Distributed extension state using SIP (Olle)

- (resources in place, doing it, 1.4 done before Christmas, project pinana)
- Manager event docs (Paul Belanger)
- Cross-platform documentation (Ben Klang)
  - (caveats for using Asterisk on operating system xyz; pull a PDF of the Wiki documentation into the source, don't forget to include basic installation information, and do it all in .txt - Ben)
- Fix libs to optionally init OpenSSL (Digium)
  - (or use existing tools; sort of a bug)
- Make ast_channel an opaque type (Digium)

3.2. (P1)

P1 is the highest priority.

- ~~Codecs (SILK, OPUS), Media Negotiation~~ (Digium)
- RTCP (Olle)
  - Pinefrog; Work to be done - Ported to trunk, added to CEL
- ~~Conferencing that supports a new magic media~~ (Digium)
  - ~~higher sampling rates~~

3.3. (P2)

- Async DNS (TCP DNS and use a good resolver)
- Named ACLs (deluxepine)
- SIP Security Events
- Light weight means of holding NAT open in SIP (less complex than current qualify, Consider it done)
- IPv6 for the restivus (IAX, Jabber/XMPP/Gtalk, Manager, etc.)
- ~~ConfBridge feature complete with MeetMe~~
- Support sound file containers (matroska)

3.4. (P3)

- Who hung up? (there's a branch, shouldn't take too much time - Olle)
- Unique identifier for filtering log data to a call
  - (finishing what was already begun w/ Clod's project, CLI filtering; should take a look at what Stephan from Unlimitel.ca's created)

3.5. (P4, Simon's features)

- Multiple SIP Sockets
  - (Listen on multiple ports or on multiple interfaces, but not all; also set binding for RTP)...alternate idea / solution would be to make Asterisk capable of loading multiple SIP profiles, it might be easier
- Multiple DNS results
  - (need to be able to traverse a list of DNS results, rather than just getting back one result)
- ICE-lite
  - (no code, responding correctly to ICE connectivity checks (STUN multiplexed on the RTP port) and understanding the SDP); it makes NAT traversal work for clients that do ICE; also addressed lightweight NAT refresh)

3.6. (P5)

- AstDB replacement
  - (realtime, there's code, nearly ready)
- SIP identity
  - (on reviewboard; needs to be forward ported; important for organizations w/ federated identities; a requirement for DTLS SRTP; not widely deployed)
- RTMP client channel driver

3.7. (P6)

- Structured identifiers for errors
  - (tag an error message with a unique string, specific to the error message and where it came from; should be alphanumeric to keep them short)
- AMI SetVar, Context limits
  - (there's code already...Olle has it)
- AMI filters on demand
- DTLS SRTP
  - (not likely to be widely deployed in the next 12 months)

3.8. (P7, not kobaz)

- Asterisk register for XMPP account (Leif)
- Write a Specification for AMI (not kobaz)
- Multiple TLS server certs
  - (1 socket, requires support by OpenSSL; simpler to implement than multiple SIP profiles; don't know if any clients use it yet; needs more research)

3.9. (P8, nice to have)

- Make resource modules that talk to DBs attempt reconnects
- Apple's new file streaming format, derived from .m3u
- Make MixMonitor and Monitor feature compatible

3.10. (P?, Research Required)

- New app_queue (as if? no, seriously? talking about this scares Russell)
- Identify and fix all bugs in AMI
- Broadsoft or Dialog Info shared line appearance (SLA) support
  - (Tabled for later discussion)
- LDAP from within the dialplan
  - (we may already have it, needs research to see if the realtime driver does what's desired - Leif)
- Device state normalization
- Anything DB over HTTP(s) with failover handling
- Use a channel as a MoH Source
- Kill Masquerades
- Bridging thread pool
- Threadify chan_sip
- Export ISDN ROSE information up to Asterisk channels

4. Testing Framework Ideas

We discussed the automated testing that has been built for Asterisk and discussed ideas for future improvements.

- Add Adhearsion
- Media Analysis (Quality)
- TestServer / TestClient
- Emulate phone-specific behavior
- Test ${RESULT} vars
- Check memory usage (for leaks)
- Custom statistics over time
- Automated load tests
- Valgrind integration
- Tone generation and analysis
- Broken dial strings
- Bad options to applications
- Make tests more generic so they can be toggled to run across multiple channels, or channel types, more easily
- Try to access features that are disallowed
- Test calls between versions of Asterisk
- Randomize call length and application during load testing - "Fuzzing"
- Automated crash analysis (generate backtraces and logs, etc)
- Testing against a SIP Proxy
- Non-root isolated test suite
- RFC4475 and RFC5118 tests (SIP Torture tests)
- More chan_sip parsing unit tests
- SDP testing multiple media streaming
- protos SIP tester
- TAHI SIP tester for IPv6
- test SSL / TLS SIP connectivity
- Sound file I/O
- Manager events for scenarios
- Basic calls with all channel types
- Connected party ID
- Tonezone tests
- Language tests

5. Release Policy Discussion

We discussed Asterisk release policy. Specifically, we were considering the current policy that excludes features from a release branch. After a bit of discussion, it was decided that no changes to policy would be made. We did agree that a new self contained module that was not compiled by default would be fine, but that it would be rare that it would provide benefit, since most projects are modifications of existing code.

6. Photos of attendees

We rock.

Asterisk 1.8 Projects

This wiki was launched after the development of Asterisk 1.8. However, this page is here as a parent for any notes regarding Asterisk 1.8 that we find lying around.

CCSS Architecture

Call Completion Supplementary Services

See attached PDF for architecture discussion.

CODEC Bit Expansion

CODEC Bit Expansion

The code base up to and including Asterisk 1.6.2 has a basic limit of 32 codecs recognizable, due to the use of a 32-bit integer for the codec bitmask. We have expanded the number of available codecs from 32 to 64, through the use of an immutable type, called format_t. This should make future expansion to even more bits more easily done.

The design of this expansion has made some changes to the architecture of codecs in order to accomplish this task. I will attempt to enumerate them here.

The initial set of 32-bits were allocated as the first 16 to audio codecs, the next 8 to video codecs, and the remaining to text codecs (which are used for fax capabilities). Initially, there is an assumption in the code that all audio codecs are contiguous, followed by a contiguous set of video codecs. After the conversion, this assumption will no longer be true. The codec bits for the existing codecs will continue to be allocated as-is, and the additional codec bits should be allocated on an as-needed basis, with audio codecs occupying slots 32-47 and video codecs occupying slots 48-62 (with a 0-based offset). Slot 63 is reserved and should not be allocated; it is used in code as an end condition for iterating through the entire set of codecs.

The frame structure has been altered. Initially, the subclass held an integer whose meaning was specified by the frametype. If the frametype was AST_FRAME_VOICE, the subclass specified the audio codec. If the frametype was AST_FRAME_VIDEO, the subclass specified the video codec, with the 0-bit set to specify a key frame. This was done with a union on the subclass, where the "integer" union member specifies the traditional 32-bit subclass and the "codec" union member specifies the new 64-bit codec bitmask. This additionally guarantees that code compiled under the old scheme will need to be altered to compile under the new scheme, which helps avoid incorrect assumptions about the state of code which might otherwise compile without errors.

The IAX2 code initially used a 32-bit integer IE to specify both the codec as well as the preferred format. An additional IE has been added, which specifies a single byte version number as the initial part of the data. This version number is initially specified as 00 and requires 8 bytes to follow, specifying the 64-bit codec bitmask, in network-byte order. This schema should allow further codec expansion in the future without allocation of any additional IEs.

Little changes are required to support further codec expansion in the future, though the majority of the work has already been accomplished. Specifically, the bitwise operations that are immutable operations in the gcc compiler will need to be altered to handle larger bitmasks. Additionally, the constants that define specific codecs will need to be changed from integers to structures.

Asterisk 1.10 Projects

chan_sip Transaction Support Proprosal

1. Introduction

SIP is a transactional protocol. Asterisk's chan_sip has no transaction concept of a transaction layer. Developers have spent countless hours providing workarounds for bugs caused by this situation. In fact, more hours have been spent trying to work around the issue than it would take to actually implement a transaction layer for chan_sip. This document outlines how a transaction layer can be added to chan_sip with a relatively minimal amount of change to existing code.

2. Table of Contents

3. Project Requirements

- RFC 3261-compliant transactions
    - Client/Server INVITE/Non-INVITE transactions
    - Connection-oriented protocols (TCP/TLS)
    - Connectionless protocols (UDP)
    - Handle transport layer errors
- Code should be as self-contained as possible to minimize bug introduction
- Possibly should be optional for backward compatibility

4. What is a transaction layer?

SIP is a transactional protocol. A transaction is basically a single request and all of the responses to that request. Essentially, the SIP transaction layer sits between the User Agent core code and the transmission layer. The purpose of the transaction layer is to handle sending retransmissions of messages that have not received a timely response (with some exceptions) and to filter out (most) retransmissions and invalid messages instead of having the UA core code handle them.

5. What does chan_sip currently do instead of using a transaction layer?

chan_sip, instead of maintaining a full transaction layer, tries to identify incoming retransmissions and marks them with an "ignore" flag. Then, in every request/response handling function where we think it might be important, we try to handle falling through the code to "do the right thing." This has led to many, many bugs. Many times we end up trying to reconstruct the same response to a request based on some state that isn't guaranteed not to have changed. The core code, in most circumstances should not even be aware that a retransmission has been received, let alone make a response to it one.

For retransmitting messages that chan_sip's UA code sends, chan_sip relies on the retrans_pkt function which is scheduled to run from sip_reliable_xmit.

6. Proposed solution

If transaction support is enabled (either by having it be the only option, or by a config option), the "ignore" flag should never be set. The transaction layer code itself can absorb the

retransmissions and the legacy code will only receive the "non-ignored" requests and responses. The existing code checking for "ignore" can be cleaned up at leisure without worrying about it causing a problem.

For handling retransmissions for messages that we send, we should always have a copy of the packet that we are sending and re-send that.

The transaction handling code should be self-contained and should rely on copies of data where possible, since retransmissions will be happening from a separate thread.

Documentation Improvements

Improvements Overview

To-Do

- Create a place for AMI event documentation, likely in the Asterisk XML docs. Make them available via the Asterisk CLI and Wiki.
- Create a simplified set of sample configuration files and PBX starter configuration files

In Progress

- Create a script that can export the wiki into PDF and plain text for inclusion with release tarballs.
    - The Confluence API includes an exportSpace() API call that should do the trick.

Done

- Migrate documentation from the doc/ directory to this wiki.
- Create a tool that syncs the built-in XML based documentation for applications, functions, etc. to this wiki

XML to Wiki

To get the reference information for applications, functions, etc. into the wiki, we need a tool that will keep what is in the source tree in sync with the wiki. The XML documentation in Asterisk should be the master, meaning that the content should never be modified in the wiki directly. The documentation should be imported into the following pages, which are all child pages of Asterisk Command Reference:

For example, the Dial() application should get its own page that is a child of Dialplan Applications .

Development

The development of this tool should be done against a development instance of Confluence. An evaluation license can be used for that purpose. The APIs for Confluence are documented here. Confluence has a set of REST APIs that are the newest development interface and will eventually be the preferred interface for development. However, the REST APIs do not yet support creating and updating pages. It is a read-only API. This tool will need to use the Confluence remote API, instead.

The programming language and supporting libraries are left as an option to the implementor.

Usage

The "astxmltowiki" tool should take the following arguments:

- path/to/core-en_US.xml
  - Asterisk XML documentation file

The tool should also take the following options:

- *username* / *password*
  - for confluence authentication
  - if not specified, request it interactively at the command line
- *server*
  - for the confluence URL
  - default to https://myth.asterisk.org

High level operation

This is the general operation of the tool, expressed in psuedo code.

```
for each application/function/AGI command/AMI action {
    if page does not exist {
        create page
    }

    translate XML documentation into Confluence wiki syntax

    update page with new content
}
```

Wiki Syntax

Use the following pages for development of the wiki syntax template to use for each command reference type. While there is already a first pass on the wiki templates to be used on these pages, feel free to tweak it further.

- Dialplan Application Template Page
- Dialplan Function Template Page
- AMI Action Template Page
- AGI Command Template Page

Other Formatting Notes

In addition to the structural elements of the XML documentation which can be mapped into the templates, there are some other formatting elements used in the Asterisk XML documentation that need to be mapped to wiki syntax.

- <para>
  - A paragraph.
- <literal>
  - Literal (fixed-width). Wrap in

    ```
    {{curly braces}}
    ```

and it will `look like this.`
- <emphasis>
  - Make these italics by wrapping the next with underscores. It will *look like this.*
- <replaceable>
  - ...
- <directory>
  - Do nothing.
- <astcli>
  - ...

Asterisk Versioning

Every page should include a note that identifies which version of Asterisk the documentation was imported from. There should *not* be more than one instance of a page for a particular application by version. Instead, we need to improve the documentation XML format to include the ability to tag elements with a version when they were first introduced. Once the mechanism for doing this has been decided, the base version for all documentation in this wiki will need to be Asterisk 1.8. From there, we can start tagging new additions with a version number of 1.10.

Media Architecture Proposal

1. Introduction

Asterisk was written from the ground up with a set of assumptions about how media is represented and negotiated. These assumptions have worked to get Asterisk where it is today, but unfortunately they have also put in place a set of limitations that must be overcome before Asterisk can meet the demands of the future. While these limitations are built into the foundation of Asterisk's design, the necessary changes required to lift these constraints can be made. This document outlines the required changes and breaks them up into a layered approach. Each section addresses a specific problem with Asterisk's current media architecture and proposes a solution. As the sections progress each new section uses the foundation outlined in the previous sections to address an increasingly complex set of problems. By attacking this issue from the foundation up it is possible to generate a complete solution that exceeds the current development constraints opening Asterisk up to an entire new set of possibilities.

2. Table of Contents

3. Project Requirements

- Lift the limit placed on the number of media formats Asterisk can support.
- Add the ability for Asterisk to represent media formats with attributes.
    - Support for SILK with attributes
    - Support for H.264 with attributes
- Add the ability for Asterisk to negotiate media formats with attributes.
- Allow translation paths to be built between all media types, not just audio.
- Allow translation paths to be built in a way that takes into account both media quality and translation cost for all media formats.
- Allow a channel to process multiple media streams, even of the same media type, with translation.
- Support the ability to renegotiate media formats after call setup is complete.
- Support the ability to pass-through media Asterisk does not yet understand.
- Support the ability to for users to specify media formats with attributes in .conf files.

4. Representation of Media Formats

4.1. Problem Overview

One of the key problems the new media architecture must address is how to represent a media format that does not have statically defined parameters. In the past, simply defining a media format type as uLaw or g722 posed no problem as these formats have a very specific set of parameters associated with them. For example uLaw is always 8khz, each sample is always the exact same size, and there is really nothing more required to describe a uLaw payload other than now large it is. Everything else can be calculated because parameters for uLaw payloads never change. Unfortunately the assumption that media formats do not need to be defined beyond their format type has proven to be a limitation in the ability to adopt modern media formats. The problems prohibiting integration of feature complete SILK codec support into Asterisk offers a prime example of how this limitation is hindering development. SILK is an audio codec that may adjust the sample rate used in a stream based upon the capabilities of the network. Right now Asterisk assumes every media format will always contain the same sample rate. Without the ability to define a format's sample rate outside of the hard coded rate defined at compile time, implementing SILK into Asterisk without limiting the codec's functionality is not currently possible.

In order to address this limitation, media formats will have the ability to be further defined using

format specific attribute structures. These structures along with usage examples are outlined in below.

4.2. Introducing ast_format, The New and Improved format_t

The ast_format structure completely replaces format_t everywhere in the code. This new structure allows for a format to be represented not only by a unique ID, but with an attribute structure as well. This means if a channel's read format is SILK and it understands 8khz->16khz without the need of translation, this can be now represented using only a single format identifier. In this case the ast_format's uid would be AST_FORMAT_SILK, and the attribute structure would be configured to further define this format as having a possible dynamic sample rate between 8khz and 16khz.

The ast_format structure on an ast_frame has a slightly different behavior than representing a read and write format though. When on a frame the attributes structure must be used only to further define the frame's payload. In the SILK read format example discussed above, the attribute structure is used to represent a sample rate range the channel's read format is capable of understanding without translation, but when the attribute structure is used on a frame it must represent a very precise set of parameters directly related to the media payload being transported. In the case of SILK, the attribute structure on a frame would represent precisely what sample rate the payload contains.

4.3. The Ast Format API

<table>
<tr><td align="center">format.h</td></tr>
</table>

```
/*! \brief This structure contains the buffer used for format
attributes */
struct ast_format_attr {
    uint8_t format_attr[AST_FORMATNEW_ATTR_SIZE];
};

/*! \brief Represents a media format within Asterisk. */
struct ast_format {
    /*! The unique id representing this format from all the other
formats. */
    enum ast_format_id id;
    /*!  Attribute structure used to associate attributes with a
format. */
    struct ast_format_attr fattr;
};

enum ast_format_cmp_res {
 /*! structure 1 is identical to structure 2. */
 AST_FORMAT_CMP_EQUAL = 0,
 /*! structure 1 contains elements not in structure 2. */
 AST_FORMAT_CMP_NOT_EQUAL,
```

```
  /*! structure 1 is a proper subset of the elements in structure
2.*/
 AST_FORMAT_CMP_SUBSET,
};

/*!
 * \brief This function is used to set an ast_format object to
represent a media format
 * with optional format attributes represented by format specific
key value pairs.
 *
 * \param format to set
 * \param id, format id to set on format
 * \param set_attributes, are there attributes to set on this
format. 0 == false, 1 == True.
 * \param var list of attribute key value pairs, must end with
AST_FORMATNEW_ATTR_END;
 *
 * \details Example usage.
 * ast_format_set(format, AST_FORMATNEW_ULAW, 0); // no capability
attributes are needed for ULAW
 *
 * ast_format_set(format, AST_FORMATNEW_SILK, 1, // SILK has
capability attributes.
 *      AST_FORMATNEW_SILK_ATTR_RATE, 24000,
 *      AST_FORMATNEW_SILK_ATTR_RATE, 16000,
 *      AST_FORMATNEW_SILK_ATTR_RATE, 12000,
 *      AST_FORMATNEW_SILK_ATTR_RATE, 8000,
 *      AST_FORMATNEW_ATTR_END);
 *
 * \return Pointer to ast_format object, same pointer that is passed
in
 * by the first argument.
 */
struct ast_format *ast_format_set(struct ast_format *format, enum
ast_format_id id, int set_attributes, ... );

/*!
 * \brief This function is used to set an ast_format object to
represent a media format
 * with optional capability attributes represented by format
specific key value pairs.
 *
 * \details Example usage. Is this SILK format capable of 8khz
 * is_8khz = ast_format_isset(format, AST_FORMATNEW_SILK_CAP_RATE,
8000);
```

```
 *
 * \return 0, The format key value pairs are within the capabilities
defined in this structure.
 * \return -1, The format key value pairs are _NOT_ within the
capabilities of this structure.
 */
int ast_format_isset(struct ast_format *format, ... );

/*!
 * \brief Compare ast_formats structures
 *
 * \retval ast_format_cmp_res representing the result of comparing
format1 and format2.
 */
enum ast_format_cmp_res ast_format_cmp(struct ast_format *format1,
struct ast_format *format2);

/*!
 * \brief Find joint format attributes of two ast_format
 * structures containing the same uid and return the intersection in
the
 * result structure.
 *
 * retval 0, joint attribute capabilities exist.
 * retval -1, no joint attribute capabilities exist.
 */
int ast_format_joint(struct ast_format *format1, struct ast_format
*format2, struct ast_format *result);

/*!
 * \brief copy format src into format dst.
 */
void ast_format_copy(struct ast_format *src, struct ast_format
*dst);

/*!
 * \brief ast_format to iax2 bitfield format represenatation
 *
 * \note This is only to be used for IAX2 compatibility
 *
 * \retval iax2 representation of ast_format
 * \retval 0, if no representation existis for iax2
 */
uint64_t ast_format_to_iax2(struct ast_format *format);

/*!
```

```
 * \brief convert iax2 bitfield format to ast_format represenatation
 * \note This is only to be used for IAX2 compatibility
 *
 * \retval on success, pointer to the dst format in the input
parameters
 * \retval on failure, NULL
```

```
 */
struct ast_format *ast_format_from_iax2(uint64_t src, struct
ast_format *dst);
```

4.4. Introducing the Format Attribute Structure

The attribute structure is present on every ast_format object. This attribute structure is an opaque
buffer that can be used in anyway necessary by the format it represents. Since it will be
necessary for Asterisk to perform a few generic operations on these attribute structures, every
format requiring the use of the attribute structure must implement and register a format attribute
interface with Asterisk. These registered interfaces are used by the Ast Format API allowing for
attributes on an ast_format structure to be set, removed, and compared using a single set of API
functions for all format types. The Ast Format API does all the work of finding the correct
interface to use and calling the correct interface functions.

The size of the buffer in the attribute structure was determined by researching the media format
with the largest number of attributes expected to be present in Asterisk 1.10. In this case the
H.264 SVC draft was used, which is an expanded form of RFC 3984 allowing for some additional
functionality. The attributes required by H.264 SVC are determined based upon the SDP
parameters defined in the draft. The SDP parameters used by the draft do not all have fixed
sizes, but it was determined that an attribute buffer of ~70 bytes will easily suffice for
representing the most common use cases. In order to account for undefined future development,
this buffer is initially set at 128 bytes which satisfies the current estimated attribute size
requirements.

4.5. The Ast Format Attribute API

format_attribute.h

```
#define AST_FORMAT_ATTR_SIZE 128

struct ast_format_attr {
 uint8_t format_attr[AST_FORMAT_FORMAT_ATTR_SIZE];
}

/*! \brief A format must register an attribute interface if it
requires the use of the format attributes void pointer */
struct ast_format_attr_interface {
    /*! format type */
    enum ast_format_id id;

    /*! \brief Determine if format_attr 1 is a subset of format_attr
2.
      *
      * \retval ast_format_cmp_res representing the result of
comparing fattr1 and fattr2.
```

```
    */
    enum ast_format_cmp_res (* const format_attr_cmp)(struct
ast_format_attr *fattr1, struct ast_format_attr *fattr2);

    /*! \brief Get joint attributes of same format type if they
exist.
     *
     * \retval 0 if joint attributes exist
     * \retval -1 if no joint attributes are present
     */
    int (* const format_attr_get_joint)(struct ast_format_attr
*fattr1, struct ast_format_attr *fattr2, struct ast_format_attr
*result);

    /*! \brief Set format capabilities from a list of key value
pairs ending with AST_FORMAT_ATTR_END.
   * \note This function is expected to call va_end(ap) after
processing the va_list. */
    void (* const format_attr_set)(struct ast_format_attr
*format_attr, va_list ap);
};

/*! \brief register ast_format_attr_interface with core.
 *
 * \retval 0 success
 * \retval -1 failure
 */
int ast_format_attr_reg_interface(struct ast_format_attr_interface
*interface);

/*!
 * \brief unregister format_attr interface with core.
 *
 * \retval 0 success
 * \retval -1 failure
```

```
   */
int ast_format_attr_unreg_interface(struct ast_format_attr_interface
*interface);
```

4.6. The New Format Unique Identifier

Media formats in Asterisk are currently defined using a bit field, format_t, where every format is uniquely identified by a single bit. While this makes comparing media format capabilities extremely simple using bitwise operations, this representation limits the number of media formats that can be represented due to the limited size of the bit field in use. Even if a bit field could represent an infinite number of bits, this representation has no concept of how to compare format capability attributes.

In order to remove the limitation of the number of unique formats that can be represented the identifier will change from a single bit representation to a numeric representation. This means that #define AST_FORMAT_ULAW (1 << 0) now becomes #define AST_FORMAT_ULAW 1. By changing the way media formats are identified from a bit in a bit field to a numeric value, the limit on the number of formats that can be represented goes from 64 to 4,294,967,296. Altering this representation completely removes the ability to use bitwise operations on a bit field containing multiple media format capabilities, but since these bitwise operations lack the ability to process format attributes, they must be replaced by a more robust system anyway. The new system for computing joint media capabilities between peers hinted at here is discussed in detail in the Representation of Format Capabilities section.

4.7. Format Unique Identifier Organization

The old system of using a single bit in a bit field to represent a single format also allows for bitmasks to be used to determine what type of media a format is categorized as. For example, there is a bitmask for determining if a format is an audio format, video format, or text format. By changing the unique id to a number the ability to use bitmasks to determine the category is no longer possible. Instead, a new convention of organizing these formats into media categories must be set in place.

Since the number of formats that can be represented will likely never be exhausted using the new system, formats can be uniquely identified and categorized using a system that sections off each category into a range of numbers. Since it is unlikely any category will ever have even close to 100,000 unique formats associated with it, each category will be sectioned off by increments of 100,000. For example, all audio formats will be uniquely identified in a category between 100,000-199,999, all video formats will be uniquely identified in a category between 200,000-299,999, and so on for every category. This new system allows for each format's unique id to be overloaded with a category as well just like the previous system did. Instead of using a bitmask to determine if a format is video or audio, a function or macro can be used to do the comparison consistently across the code base.

4.8. New Format Unique Id Changes to frame.h

<table>
<tr><td colspan="1" align="center">**frame.h Old**</td></tr>
</table>

```
/*OLD*/
#define AST_FORMAT_AUDIO_MASK 0xFFFF0000FFFFULL
#define AST_FORMAT_G723_1     (1ULL << 0)
#define AST_FORMAT_GSM        (1ULL << 1)
#define AST_FORMAT_ULAW       (1ULL << 2)
#define AST_FORMAT_ALAW       (1ULL << 3)


#define AST_FORMAT_VIDEO_MASK ((((1ULL << 25)-1) &
~(AST_FORMAT_AUDIO_MASK)) | 0x7FFF000000000000ULL)
#define AST_FORMAT_H263_PLUS  (1ULL << 20)
#define AST_FORMAT_MP4_VIDEO  (1ULL << 22)
```

<table>
<tr><td colspan="1" align="center">**frame.h New Changes**</td></tr>
</table>

```
/*NEW*/
#define AST_FORMAT_INC 100000

/* ALL FORMAT CATEGORIES */
enum ast_format_type {
    AST_FORMAT_TYPE_AUDIO = 1 * FORMAT_INC,
    AST_FORMAT_TYPE_VIDEO = 2 * FORMAT_INC,
    AST_FORMAT_TYPE_IMAGE = 3 * FORMAT_INC,
};

enum ast_format_id {
    /* ALL AUDIO FORMATS */
    AST_FORMAT_G723_1     = 1 + AST_FORMAT_TYPE_AUDIO,
    AST_FORMAT_GSM        = 2 + AST_FORMAT_TYPE_AUDIO,
    AST_FORMAT_ULAW       = 3 + AST_FORMAT_TYPE_AUDIO,
    AST_FORMAT_ALAW       = 4 + AST_FORMAT_TYPE_AUDIO,

    /* ALL VIDEO FORMATS */
    AST_FORMAT_H263_PLUS  = 1 + AST_FORMAT_TYPE_VIDEO,
    AST_FORMAT_MP4_VIDEO  = 2 + AST_FORMAT_TYPE_VIDEO,
};

/* Determine what category a format type is in */
#define AST_FORMAT_GET_TYPE(format) (((unsigned int) (format->uid /
AST_FORMAT_INC)) * AST_FORMAT_INC)
```

4.9. New Format Representation Code Examples and Use cases.

This section shows example usage of the ast_format structure and how it replaces existing

functionality in Asterisk. It also outlines other highlevel use cases that can not easilly be represented by a code example.

Example 1: One to one mapping of old format_t usage with ast_format structure and its API.

Example 1 - Old

```
/* OLD: Media formats are represented by a bit in the format_t bit
field. */
 format_t read_format;
 read_format = AST_FORMAT_ULAW;
```

Example 1 - New

```
/* NEW: Media formats are represented using the ast_format struct
and are stored in an ast_cap object. */
 struct ast_format read_format;
 ast_format_set(&read, AST_FORMAT_ULAW);
```

Example 2: Set an optional format attribute structure for a SILK ast_format structure capable of a dynamic sample rate.

Example 2

```
struct ast_format read_format;
ast_format_set(&read, AST_FORMAT_SILK,
 AST_FORMAT_SILK_RATE, 24000,
 AST_FORMAT_SILK_RATE, 16000,
 AST_FORMAT_SILK_RATE, 12000,
 AST_FORMAT_SILK_RATE, 8000,
 AST_FORMAT_END);
```

Example 3: Set the sample rate of SILK ast_frame representing the sample rate of the frame's payload. Then compare the format of the ast_frame with a read format determine if translation is required.

Example 3

```
struct ast_format read_format;
/* The read format is of format type SILK and can be of sample rates
8khz and 12khz */
ast_format_set(&read, AST_FORMAT_SILK,
 AST_FORMAT_SILK_RATE, 12000,
 AST_FORMAT_SILK_RATE, 8000,
 AST_FORMAT_END);

/* The frame's format type is SILK and the payload is 24khz audio.
*/
ast_format_set(frame->subclass.format, AST_FORMAT_SILK,
 AST_FORMAT_SILK_RATE, 24000,
 AST_FORMAT_END);

/* Comparing the frame with the read format shows that while the
formats are identical
 * their attributes make them incompatible requiring a translation
path to be built. */
if ((ast_format_cmp(&read_format, frame->subclass.format) < 0)) {
 /* Build Translation Path.
  * This will be the outcome of this example. */
} else {
 /* Frame's format is either identical or a subset of the
read_format
  * requiring no translation path. */
}
```

Example 4. Determine if a format is of type audio.

Example 4 Old

```
/*OLD*/
format_t format = AST_FORMAT_ULAW;
if (format & AST_FORMAT_AUDO_MASK) {
 /* this is of type audio */
}
```

<table>
<tr><td align="center">**Example 4 New**</td></tr>
</table>

```
/*NEW*/
struct ast_format format;
ast_format_set(&format, AST_FORMAT_ULAW);
if (AST_FORMAT_GET_TYPE(&format) == AST_FORMAT_TYPE_AUDIO) {
 /* this is of type audio */
}
```

Example 5: Media format seamlessly changes parameters midstream.

1. A channel is defined to have a write format of SILK with the capabilities of understanding 8khz and 16khz without translation.
2. A stream of SILK audio ast_frames containing 16khz frame attributes begin to be written to the channel.
3. During the call the audio stream's SILK frame attributes change to 8khz.
4. ast_write() determines this change is still within the channel's write format capabilities and continues without translation.

Example 6: Media format changes parameters requiring translation midstream.

1. A channel is defined to have a write format of SILK with the capabilities of understanding 8khz and 16khz without translation.
2. A stream of SILK audio ast_frames containing 16khz frame attributes begin to be written to the channel.
3. During the call the audio stream's SILK frame attributes change to 24khz.
4. ast_write() determines this change is not within the bounds of the channel's write format capabilities and builds a translation path from 24khz SILK to 16khz SILK.

5. Representation of Format Capabilities

5.1. Problem Overview

The new way of handling format capabilities must address two issues. First, formats are no longer represented by the format_t bit field and are replaced by the ast_format structure. This means that the old system of representing format capability sets with a bit field must be replaced as well. Second, even if we could use a bit field to represent format capability sets, the bitwise operators used to compare capabilities and calculate joint capabilities are incapable of processing the new format attribute structures. In order to handle both of these changes, an opaque capabilities container must be created to manipulate sets of ast_format structures. This container must also be coupled with an API that abstracts all the work required to compare sets of ast_formats and their internal format attributes.

5.2. Introducing ast_cap, The Format Capability Container.

The Format Capability API introduces a new container type, struct ast_cap, which acts as the opaque capabilities container discussed in the overview. Like an ao2_container holds astobj2 objects, the ast_cap container holds ast_format objects. The thing that sets the ast_cap

container apart from other generic containers in Asterisk is that it is designed specifically for the purpose of comparing and manipulating sets of ast_format structures. API functions for adding/removing ast_formats, computing joint capabilities, and retrieving all capabilities for a specific media type are present. The best way to communicate the big picture for how this new container and API replaces the current architecture is by providing some examples. These examples will walk through the sections discussed so far and provide a better understanding for how the ast_format and ast_cap containers interact with each other using the new API. All the examples below take code from the existing media architecture in Asterisk and show how the new architecture replaces it.

Example 1: Add format capabilities to a peer.

Example 1 - Old

```
/* ---OLD: Media formats are represented by a bit in a bit field. */
format_t capabilities = 0;
capabilities |= AST_FORMAT_ULAW;
capabilities |= AST_FORMAT_GSM;
/* XXX SILK can not be set using a bit since it requires a
capability
 * attribute to be associated with it.
 * capabilities |= AST_FORMAT_SILK;
 */
```

Example 1 - New

```
/* ---NEW: Media formats are represented using the ast_format struct
and are stored in an ast_cap object.*/
struct ast_format tmp = { 0, };

ast_cap_add(capabilties, ast_format_set(&tmp, AST_FORMAT_ULAW));
ast_cap_add(capabilties, ast_format_set(&tmp, AST_FORMAT_GSM));

/* SILK media format requires the format capability attribute to be
set.*/
ast_format_set(&tmp, AST_FORMAT_SILK,
 AST_FORMAT_SILK_CAP_RATE, 24000,
 AST_FORMAT_SILK_CAP_RATE, 16000,
 AST_FORMAT_SILK_CAP_RATE, 12000,
 AST_FORMAT_SILK_CAP_RATE, 8000,
 AST_FORMAT_ATTR_END);
ast_cap_add(capabilties, &tmp);
```

Example 2: Find joint capabilities between a peer and remote endpoint.

Example 2 - Old

```
/*---OLD: Peer and remote capabilities are bit fields, no capability
attributes can be used.*/
format_t jointcapabilties = 0;

peer->capability |= (AST_FORMAT_ULAW | AST_FORMAT_GSM);

/*
 * peer->capability = ULAW and GSM
 *
 * remote_capabilities structure is already built to contain uLaw
 * remote_capability = ULAW
 *
 * jointcapabilities will be ULAW
 */
jointcapabilties = peer->capability & remote_capability
```

Example 2 - New

```
/*---NEW: Peer and remote capabilities are ast_cap objects.*/
struct ast_cap *jointcapabilities;

ast_cap_add(peer->capability, ast_format_set(&tmp,
AST_FORMAT_ULAW));
ast_cap_add(peer->capability, ast_format_set(&tmp, AST_FORMAT_GSM));

ast_format_set(&tmp, AST_FORMAT_SILK,
 AST_FORMAT_SILK_CAP_RATE, 24000,
 AST_FORMAT_SILK_CAP_RATE, 16000,
 AST_FORMAT_SILK_CAP_RATE, 12000,
 AST_FORMAT_SILK_CAP_RATE, 8000,
 AST_FORMAT_ATTR_END);
ast_cap_add(peer->capabilties, &tmp);

ast_format_set(&tmp, AST_FORMAT_H264,
 AST_FORMAT_H264_CAP_PACKETIZATION, 0,
 AST_FORMAT_H264_CAP_PACKETIZATION, 1,
 AST_FORMAT_H264_CAP_RES, "CIF",
 AST_FORMAT_H264_CAP_RES, "VGA",
 AST_FORMAT_ATTR_END);
ast_cap_add(peer->capabilties, &tmp);

/*
 * peer->capabilities structure was just built to contain.
 * silk (rate = 24000, rate = 16000, rate = 12000, rate = 8000)
 * h.264 (packetization = 0, packetization = 1, res = vga, res =
cif)
 *
 * remote_capabilities structure is already built to contain
 * silk (rate = 16000)
 * h.264 (packetization = 0, res = vga, res = svga)
 *
 * The resulting jointcapabilities object contains
 * SILK (Rate = 16000khz)
 * H.264 (packetization = 0, Res = VGA)
 *
 * Computing of joint capabilities of formats with capability
attributes is
 * possible because of the format attribute interface each format
requiring
 * attributes must implement and register with the core.
 */
jointcapabilities = ast_cap_joint(peer->capability,
remote_capability);
```

Example 3: Separate audio, video, and text capabilities.

Example 3 - Old

```
/*---OLD: Separate media types are separated by a bit mask.*/
format_t video_capabilities = capabilities & AST_FORMAT_VIDEO_MASK;
format_t audio_capabilities = capabilities & AST_FORMAT_AUDIO_MASK;
format_t text_capabilities = capabilities & AST_FORMAT_TEXT_MASK;
```

Example 3 - New

```
/*---NEW: Separate media types are returned on a new capabilities
structure using ast_cap_get_type()*/
struct ast_cap *video = ast_cap_get_type(capabilities,
AST_FORMAT_TYPE_AUDIO);
struct ast_cap *voice = ast_cap_get_type(capabilities,
AST_FORMAT_TYPE_VIDEO);
struct ast_cap *text = ast_cap_get_type(capabilities,
AST_FORMAT_TYPE_TEXT);
```

5.3. Ast Format Capability API Defined

format_capability.h

```
/*! Capabilities are represented by an opaque structure statically
defined in format_capability.c */
struct ast_cap;

/*! \brief Allocate a new ast_cap structure.
 *
 * \retval ast_cap object on success.
 * \retval NULL on failure.
 */
struct ast_cap *ast_cap_alloc(void);

/*! \brief Destroy an ast_cap structure.
 *
 * \return NULL
 */
void *ast_cap_destroy(struct ast_cap *cap);

/*! \brief Add format capability to capabilities structure.
 *
 * \note A copy of the input format is made and that copy is
 * what is placed in the ast_cap structure.  The actual
```

```
 * input format ptr is not stored.
 */
void ast_cap_add(struct ast_cap *cap, struct ast_format *format);

/*! \brief Remove format capability from capability structure.
 *
 * \Note format must match Exactly to format in ast_cap object in
order
 * to be removed.
 *
 * \retval 0, remove was successful
 * \retval -1, remove failed. Could not find format to remove
 */
int ast_cap_remove(struct ast_cap *cap, struct ast_format *format);

/*! \brief Remove all format capabilities from capability
 * structure for a specific format id.
 *
 * \Note This will remove _ALL_ formats matching the format id from
the
 * capabilities structure.
 *
 * \retval 0, remove was successful
 * \retval -1, remove failed. Could not find formats to remove
 */
int ast_cap_remove_byid(struct ast_cap *cap, enum ast_format_id id);

/*! \brief Find if ast_format is within the capabilities of the
ast_cap object.
 *
 * retval 1 format is compatible with formats held in ast_cap
object.
 * retval 0 format is not compatible with any formats in ast_cap
object.
 */
int ast_cap_iscompatible(struct ast_cap *cap, struct ast_format
*format);

/*! \brief Get joint capability structure.
 *
 * \retval !NULL success
 * \retval NULL failure
 */
struct ast_cap *ast_cap_joint(struct ast_cap *cap1, struct ast_cap
*cap2);
```

```
/*! \brief Get all capabilities for a specific media type
 *
 * \retval !NULL success
 * \retval NULL failure
 */
struct ast_cap *ast_cap_get_type(struct ast_cap *cap, enum
ast_format_type ftype);


/*! \brief Start iterating formats */
void ast_cap_iter_start(struct ast_cap *cap);

/*! \brief Next format in interation
 *
 * \details
 * Here is how to use the ast_cap iterator.
 *
 * 1. call ast_cap_iter_start
 * 2. call ast_cap_iter_next in a loop until it returns -1
 * 3. call ast_cap_iter_end to terminate the iterator.
 *
 * example:
 *
 * ast_cap_iter_start(cap);
 * while (!ast_cap_iter_next(cap, &format)) {
 *
 * }
 * ast_cap_iter_end(Cap);
 *
 * \retval 0 on success, new format is copied into input format
struct
 * \retval -1, no more formats are present.
 */
int ast_cap_iter_next(struct ast_cap *cap, struct ast_format
*format);

/*! \brief Ends ast_cap iteration.
 * \note this must be call after every ast_cap_iter_start
 */
void ast_cap_iter_end(struct ast_cap *cap);

/*!
 * \brief ast_cap to iax2 bitfield format represenatation
 *
 * \note This is only to be used for IAX2 compatibility
 *
```

```
 * \retval iax2 representation of ast_cap
 * \retval 0, if no iax2 capabilities are present in ast_cap
 */
uint64_t ast_cap_to_iax2(struct ast_cap *cap);

/*!
 * \brief convert iax2 bitfield format to ast_cap represenatation
```

```
 * \note This is only to be used for IAX2 compatibility
 */
void ast_cap_from_iax2(uint64_t src, struct ast_cap *dst);
```

6. IAX2 Ast Format API Compatibility

IAX2 represents media formats the same way Asterisk currently does using a bit field. This allows Asterisk to communicate format capabilities over IAX2 using the exact same representation Asterisk uses internally. This relationship between Asterisk and IAX2 breaks with the introduction of the ast_format and ast_cap structures though. In order for Asterisk to maintain compatiblity with IAX2 a conversion layer must exist between the previous format representation and the new format representation. This conversion layer will be limited to the formats defined at the moment the media format representation in Asterisk changes to use the ast_format structure. As new media formats are introduced, they must be added to this conversion layer in order to be transported over IAX2. Any media formats requiring the use of media attributes may have to be excluded from this conversion depending on their complexity. Eventually the number of media formats that can be represented in IAX2 will be exhasted. At that point it must be decided to either accept that limitation or alter the protocol in a way that will expand it to take advantage of Asterisk's new format capabilities. This proposal is not defining a change any changes to the IAX2 protocol.

7. Revised Format Translation

7.1. Problem Overview

There are two sets of problems that must be addressed in regards to media translation in Asterisk. The first set of problems is a ripple effect caused by the changes surrounding the new representation of media formats with attributes. Translators must gain the ability to process these attributes and build translation paths between formats requiring the use of them. The other set of problems involves the ability to translate between media types other than just audio. The current translation architecture is very audio specific. It assumes that all translators are audio format translators of some kind, and that no other media type will ever be translated. This assumption is not only within the translation code, it is also deeply rooted throughout the code base. The ability to translate between media other than audio is a concept Asterisk completely lacks at the moment.

This section builds upon the foundation established by the new ast_format media format representation and uses it to redefine what translators look like and how translation paths are built. After these changes are made Asterisk will still not be able to translate video or other types of media even if translation paths actually exist between them. This problem is a result of limitations set in place by the Ast Channel API. The changes required to lift that limitation are discussed in the "Handling Multiple Media Streams" section.

7.2. Building Translation Paths

The current method of calculating translation cost by using the computational time required to translate between formats is no longer effective. When all the formats in Asterisk were 8khz audio, picking the best translation path based upon computational cost made sense. The problem with this system now is that it does not take into account the quality of the translation. It

may be computationally quicker to translated from one 16khz audio format to another 16khz audio format using 8khz signed linear audio even when 16khz signed linear is an option. Regardless of the computational costs, down sampling an audio stream unless it is absolutely necessary is a bad idea from a quality perspective. Provisions were made in the current code base to account for the down sampling issue just described, but the introduction of that fix was merely a hack to sustain the current system until a more robust architecture could be set in place. The new system must be aware of quality changes between all forms of media, not just sample rate.

Instead of building a translation cost table out based on computational complexity, the table should be built based on what kind of translation is taking place. For example categorizing a translator as a "lossless to lossy translation with a down sampling of quality" gives quite a bit more information about what kind of translation is actually taking place than simply knowing the translation is between two formats and it takes x amount of time to compute 1 second of sample data. As new formats are introduced, knowing how all the different translators affect media during translation allows the path builder algorithm to consistently produce the best quality available.

7.2.1. Computing Translation Costs

The new translation cost table is built on a scale between 400 and 9999. Notice that the lowest cost is 400 and the next cost after that is 600. These two numbers add up to 1000, which guarantees that a direct translation path will always take precedence over any path containing multiple translation steps. The only exception to this rule is a multiple step translation path between lossless formats of the same quality, which does not exist in Asterisk yet but may in the future.

Every one of these cost categories can be thought of as a range starting at the number listed and ranging all the way up to the next category. If a format is capable of multiple translators for any single category listed below, the cost associated with those translators should not fall onto the same cost number as each other. Instead each translator for a single format calling into the same cost table category should be given a weighted cost within the category's range. For example, siren17 is a 32khz audio codec with translators capable of down sampling to both 16khz signed linear and 8khz signed linear. Both of these translators fall under the [~dvossel@digium.com:lossy -> lossless] downsample" category which starts at cost 960. In order to make this work the 16khz conversion would be 960 and the 8khz conversion would be 961. This gives the translator that loses the least amount of information priority over the the one that loses more if a tie occurs.

This cost table is weighted in a way that assigns lower cost to translators with the most ideal outcome. For example, translating between a lossless format to a lossy format is aways more ideal that converting a lossy format to a lossless format, translating between two lossy formats of the same quality is always more ideal than translating to a lossy format of lesser quality, and translating to a format of equivalent in quality to the original format is more ideal than any translation that requires some sort of re-sampling. The costs are computed based on these principles and more.

7.2.2. Translation Cost Table

Table Terms
Up Sample: The original format is translated to a format capable of representing more detailed

information than the original one. Examples of this term would be audio codec being translated to a higher sample rate, a video codec being translated to a higher resolution/frame rate, or an image being translated to a higher resolution.

Down Sample: The original format is translated to a format of lesser quality. Examples of this term would be audio codec being translated to a lower sample rate, a video codec being translated to a lower resolution/frame rate, or an image being translated to a lower resolution.

Original Sampling: The original format is translated to a format of similar quality with little to no loss of information. Examples of this term would be an audio codec being translated to a format equivalent in quality of the original one, a video codec being translated to a format which preserves all the original information present, and an image being translated to another format preserving the same resolution and color depth.

Translation Cost Table
| ```
--- Lossless Source Translation Costs
400 [lossless -> lossless] original sampling
600 [lossless -> lossy]    original sampling
800 [lossless -> lossless] up sample
825 [lossless -> lossy]    up sample
850 [lossless -> lossless] down sample
875 [lossless -> lossy]    down sample
885 [lossless -> Unknown]  Unknown sample

--- Lossy Source Translation Costs
900 [lossy -> lossless]    original sampling
915 [lossy -> lossy]       original sampling
930 [lossy -> lossless]    up sample
945 [lossy -> lossy]       up sample
960 [lossy -> lossless]    down sample
975 [lossy -> lossy]       down sample
985 [lossy -> Unknown]     Unknown sample
``` |

7.2.3. Translation Path Examples

Example 1: Downsampling g722 to ulaw using signed linear as an intermediary step. Notice that using two lossless conversions is more expensive than downsampling g722 directly to 8khz slin.

Example 1 - g722 to ulaw
| ```
[g722->slin16->slin->ulaw] 900+850+600 = 2350
[g722->slin->ulaw]         960+600     = 1560 wins
``` |

Example 2: Direct lossy to loss translation using ulaw to alaw. Notice how the direct path between uLaw and aLaw beats using the intermediary slin step.

<div align="center">

Example 2 - uLaw to aLaw

</div>

```
[ulaw->slin->alaw] 900+600 = 1500
[ulaw->alaw]       945      = 945 wins
```

Example 3: Complex resampling of siren14 to siren7 using g722 as an intermediary step. Notice how downsamping all the way to 8khz signed linear loses to the path that only requires downsampling to 16khz signed linear.

<div align="center">

Example 3 - siren14 to siren7

</div>

```
[siren14->slin->g722->slin16->siren7]   960+825+900+600 = 3285
[siren14->slin16->g722->slin16->siren7] 960+600+900+600 = 3060 wins
```

Example 4: Complex resampling using siren14 to a fake 32khz lossy codec. Notice how siren14->slin16 has a 830 cost while siren14-slin8 has 831. This allows translations within the same category to be weighted against each other to produce the best quality.

<div align="center">

Example 4 - siren14 to fake 32khz codec

</div>

```
[siren14->slin->Fake 32khz lossy Codec]   961+825 = 1786
[siren14->slin16->Fake 32khz lossy Codec] 960+825 = 1785 wins
```

7.2.4. Translator Costs Defined.

Note that the table costs actually defined in the code are larger than the ones discussed so far by a factor of 1000.

<div align="center">

translator.h Defining Cost Table

</div>

```
/*!
 * \brief Translator Cost Table definition.
 *
 * \note The defined values in this table must be used to set
 * the translator's table_cost value.
 *
 * \note The cost value of the first two values must always add
 * up to be greater than the largest value defined in this table.
 * This is done to guarantee a direct translation will always
 * have precedence over a multi step translation.
 *
 * \details This table is built in a way that allows translation
 * paths to be built that guarantee the best possible balance
 * between performance and quality.  With this table direct
```

```
     * translation paths between two formats always takes precedence
     * over multi step paths, lossless intermediate steps are always
     * chosen over lossy intermediate steps, and preservation of
     * sample rate across the translation will always have precedence
     * over a path that involves any re-sampling.
     */
    enum ast_trans_cost_table {

     /* Lossless Source Translation Costs */

     /*! [lossless -> lossless] original sampling */
     AST_TRANS_COST_LL_LL_ORIGSAMP = 400000,
     /*! [lossless -> lossy]    original sampling */
     AST_TRANS_COST_LL_LY_ORIGSAMP = 600000,

     /*! [lossless -> lossless] up sample */
     AST_TRANS_COST_LL_LL_UPSAMP   = 800000,
     /*! [lossless -> lossy]    up sample */
     AST_TRANS_COST_LL_LY_UPSAMP   = 825000,

     /*! [lossless -> lossless] down sample */
     AST_TRANS_COST_LL_LL_DOWNSAMP = 850000,
     /*! [lossless -> lossy]    down sample */
     AST_TRANS_COST_LL_LY_DOWNSAMP = 875000,

     /*! [lossless -> unknown]     unknown.
      * This value is for a lossless source translation
      * with an unknown destination and or sample rate conversion. */
     AST_TRANS_COST_LL_UNKNOWN     = 885000,

     /* Lossy Source Translation Costs */

     /*! [lossy -> lossless]    original sampling */
     AST_TRANS_COST_LY_LL_ORIGSAMP = 900000,
     /*! [lossy -> lossy]       original sampling */
     AST_TRANS_COST_LY_LY_ORIGSAMP = 915000,

     /*! [lossy -> lossless]    up sample */
     AST_TRANS_COST_LY_LL_UPSAMP   = 930000,
     /*! [lossy -> lossy]       up sample */
     AST_TRANS_COST_LY_LY_UPSAMP   = 945000,

     /*! [lossy -> lossless]    down sample */
     AST_TRANS_COST_LY_LL_DOWNSAMP = 960000,
     /*! [lossy -> lossy]       down sample */
     AST_TRANS_COST_LY_LY_DOWNSAMP = 975000,
```

```
/*! [lossy -> unknown]    unknown.
 * This value is for a lossy source translation
 * with an unknown destination and or sample rate conversion. */
```

```
    AST_TRANS_COST_LY_UNKNOWN        = 985000,

};
```

7.2.5. Creation of Translation Path Matrix

Most least cost algorithms take a matrix as input. The current code's translation path matrix is represented by a 2 dimensional array of translation path structures. The current matrix will not change structurally, but there are some complications involved. The current code accesses translation paths from the matrix using index values which represent individual formats. The index values are computed by converting the format's bit representation to a numeric value. Since the numeric representation of a format bit has to be between 1 and 64, the maximum size of the bit field in use, the numeric representation works as an index for the current two dimensional matrix. With the introduction of the ast_format structure, this conversion between a format's unique id and the a matrix index value is not clean. To account for this complication a hash table mapping every format id to a matrix index value will be used.

7.2.6. Computing Least Cost Translation Paths

The Floyd-Warshall algorithm will be the least cost algorithm in use. At its core, the current translation path building code uses this algorithm but has a few layers of complexity added on top of the base algorithm to deal with translation paths between audio codecs of differing sample rates. With the introduction of the new translation cost table, this additional complexity is completely stripped away from the algorithm. Now the translation costs are computed with translation quality and efficiency in mind, which abstracts these concepts away from least cost algorithm in use.

Floyd Warshall Algorithm
| ```
FloydWarshall ()
    for k := 1 to n
        for i := 1 to n
            for j := 1 to n
                path[i][j] = min (path[i][j], path[i][k]+path[k][j]);
``` |

7.3. Translator Redundancy and Failover

It is possible that multiple redundant translators may exist for a single translation path. A common example of this would be a hardware translator with limited capacity coupled with a software translator. Both of these translators perform the exact same task, but the hardware translator is much faster. In this case the hardware translator would be used until it reached capacity and then it would failover to the software translator. There is however a complication involved with this. Only one of these translators can exist in the translation path matrix at a time. This means that when multiple translators with the same source and destination formats are present, some sort of priority must be used to pick which one is used. If the translator in use reaches capacity it then must deactivate itself allowing the matrix to be rebuilt in order to take advantage of the redundant translator.

In order to prioritize redundant translators, computational cost will be used. Formats requiring the use of redundant translators must supply a set of sample data to translate. This data is already present for most audio formats because it is required by the current architecture to compute translation cost. Translation cost in the new architecture is replaced by the translation cost table, but computational cost is still important when choosing between redundant translators.

7.4. Redefining The Translator Interface

Translators are currently defined by a simple set of functions (constructor, destructor, framein, frameout) coupled with a source and destination media format to translate between. There is not much that needs to be changed about this interface except that the source and destination formats must be converted to be ast_format structures in all the existing code, and each translator must provide a cost value. There will be a table available to guide exactly what cost value to use. In order to make any future changes to the cost table effortless, defined values will be used when assigning cost to a translator. Otherwise this interface is in great shape for the changes ahead.

<table>
<tr><td colspan="1" align="center">**Registering Translator Interface**</td></tr>
</table>

```c
/* each format must be declared statically now */
static struct ast_format slin16;
static struct ast_format g722;

/* each interface holds a pointer to the static formats. */
static struct ast_translator lin16tog722 = {
 .name = "lin16tog722",
 .cost = AST_TRANS_COST_LL_LY_ORIGSAMP,
 .srcfmt = &slin16,
 .dstfmt = &g722,
 .newpvt = lin16tog722_new, /* same for both directions */
 .framein = lintog722_framein,
 .sample = slin16_sample,
 .desc_size = sizeof(struct g722_encoder_pvt),
 .buffer_samples = BUFFER_SAMPLES * 2,
 .buf_size = BUFFER_SAMPLES,
};

/* Notice the static formats are initialized before registering the
translator */
static int load_module(void)
{
 int res = 0;

 ast_format_set(&slin16, AST_FORMAT_SLIN16);
 ast_format_set(&g722, AST_FORMAT_G722);

 res |= ast_register_translator(&lin16tog722);

 if (res) {
  unload_module();
  return AST_MODULE_LOAD_FAILURE;
 }

 return AST_MODULE_LOAD_SUCCESS;
}
```

8. Handling Multiple Media Streams

8.1. Problem Overview

Asterisk was designed from the ground up with the idea of only one audio media path being passed between channels. The code that handles this media path is done in such a way that makes expanding it to multiple media paths very difficult, especially media that is not audio.

Asterisk has gotten away with being able to support very limited video functionality by not treating it as a media path at all. Instead of putting all media in the same media path as audio, video and other forms of media are just passed through similar to the way signalling is done. In order to bring all media into the same code path as audio, several fundamental design changes must be made to the way channels represent media streams. This section discusses those changes and how they affect channel drivers and other applications requiring access to media streams.

8.2. Defining a Media Stream in Asterisk

The first step in improving Asterisk's ability to represent multiple media streams is to actually define what a media stream is. At the moment, a stream in Asterisk is a very abstract idea. There is no tangible representation of a stream, no stream object or structure. The best representation of a stream Asterisk has now is the ast_channel structure which is capable of representing a single set of audio tx/rx streams through the use of a bunch disjoint elements. Lets start this discussion by breaking out the elements of the ast_channel structure that allow it to represent these streams.

In order for the ast_channel structure to represent a single set of audio tx/rx streams it needs the following things.

1. **Read translator** - Translates stream on the read path going into the Asterisk Core.
2. **Write translator** - Translates stream on the write path going out to the channel driver.
3. **Native Format Capabilities** - Native capabilities the channel driver is capable of understanding without translation for this stream.
4. **Read Format** - Requested Read format after translation on the read path.
5. **Raw Read Format** - Expected read format before translation.
6. **Write Format** - Requested write format after translation on the write path.
7. **Raw Read Format** - Expected write format before translation.

The combination of all these items represent everything Asterisk needs to make channels compatible with one another and build translation paths between one another for a single set of corresponding tx/rx streams. The problem with this architecture is that all these disjoint elements make it impossible to replicate this functionality allowing for multiple tx/rx streams to exist on a single channel. In order for channels in Asterisk to gain the ability to process multiple tx/rx stream sets on a single channel all of theses stream elements must be organized into an isolated structure that can be easily replicated and manipulated. This new structure is called the **ast_channel_stream** structure and is discussed in detail in the next section.

8.3. Introducing ast_channel_stream, Making Sense out of Madness

The ast_channel_stream structure is made up of all the individual elements required to represent single set of tx/rx streams on an ast_channel structure. This allows all the disjoint translators and formats on the ast_channel structure associated with the audio streams go away and be replaced by a single ast_channel_stream structure. Everyplace in the current code base that directly accesses any of the stream elements on a channel such as nativeformats, readformat, and writeformat will be replaced by a set of API functions provided by the new Ast Channel Stream API. This API contains all the common operations channel drivers and applications need to perform on a stream, such as setting the native format capabilities, initializing the read/write formats, retrieving the current read/write formats, and setting the read/write formats. By using this API, channels also gain the ability to contain more than one media stream set. This is done through the concept of stream identifiers which is further discussed in the next section.

8.4. Stream Identifiers

The ast_channel_stream structure isolates the concept of tx/rx streams to a single entity allowing channels to represent multiple streams through the use of multiple ast_channel_stream structures. Since it is prohibited for any part of Asterisk except channel.c to directly access the ast_channel_stream structures on a channel, the rest of Asterisk needs a way access these individual streams through the use of the Ast Channel Stream API. This introduces the concept of **stream identifiers**. Stream identifiers completely abstract away the concept of the ast_channel_stream structure from the rest of Asterisk. Every ast_channel_stream structure on a channel will have a unique stream id assigned to it. This stream id is required by every function in the Ast Channel Stream API to access and manipulate the individual streams on a channel.

In order to separate ast_frames belonging to separate streams, a stream id will also be present on each frame. This will involve placing a new value on the ast_frame structure to represent what stream the frame belongs to. By default the current code will not use the stream id on the ast_frame even though it will be present. This concept is discussed in more detail in the "Default Streams" section.

Steam identifiers are organized into three categories. For the sake of organization and ABI compatibility each of these categories are given a range of unique stream identifiers available to them. Separating the default streams from the auxiliary and dynamic streams also makes it much easier to filter out auxiliary and dynamic streams for applications and modules that do not support them. Once a new stream identifier is defined, its unique id must remain consistent.

1. **default streams**: Unique id defined between 1 - 99999
2. **auxiliary streams**: Unique id defined between 100000 - 199999
3. **dynamic streams**: Unique id defined between 200000 - 299999

8.4.1. Default Streams

Since Asterisk was designed with the concept of a single audio tx/rx stream set existing on a channel, some provisions must be made to allow for a smooth transition into the concept of multiple stream sets. This is where default streams come into play. Every ast_channel structure will contain a set of default streams associated with it, each with a predefined consistent stream id.

Default Audio Streams - The first default tx/rx stream set present on every channel is the default audio streams. This is the stream set all of Asterisk already knows about. It is the one that used to be made of individual elements in the ast_channel structure but was stripped out after defining the ast_channel_stream structure. Every channel driver built so far already knows how to manipulate these streams and many applications require access to them as well. All ast_frames of type AST_FRAME_VOICE with a stream id of 0 will automatically match this default stream set on a channel. Since 0 is the default initialization value for the stream id on a frame, all the channel drivers and applications already making use of these streams do not have to be modified.

It should be noted that while additional audio streams will be possible in the future, it is likely the default audio stream will be the only one that any kind of tone detection is performed on for DTMF, FAX, etc. This document does not attempt to alter this limitation in any way.

Default Video Streams - It is currently impossible to do translation between two channels transmitting different video formats because the channel has no way of representing video translators. This changes with the introduction of the default video rx/tx stream set. Similar to the

default audio streams, any video frames containing a stream Id of 0 is automatically matched to the default video stream set on a channel.

As more media types are introduced, it may be beneficial to define additional default stream sets. Initially only audio and video will present.

8.4.2. Auxiliary Streams

If a channel driver is capable of negotiating more streams than can be represented by the default rx/tx stream sets on a channel, the auxiliary media stream sets can be used. These stream sets work the exact same way as the default stream sets except they require the use of the media stream id on frames. With auxiliary streams the stream id must be present on every ast_frame created for the stream. This allows channels and applications not capable of processing auxiliary streams to filter out the frames they don't understand.

Since Asterisk supports multiple protocols with various capabilities, all the auxiliary streams that can be used anywhere in Asterisk must be defined at compile time. This means when a channel driver is extended to make use of a new type of auxiliary stream, that stream must be defined with a stream id that uniquely represents it across the entire code base. This is the only way to keep the different types of auxiliary streams and what they are used for consistent across all modules.

<div style="border: 1px solid #ccc;">

Auxiliary Stream Usage Example

```
Example 1: Chan_sip is extended to make use of up to four video and
audio streams per call.  This sort of functionality has never been
done before so six new auxiliary streams must be defined for the
three new video and three new audio streams.

enum ast_channel_stream_id {
    /*! Define Default Streams below */
    AST_STREAM_DEFAULT_AUDIO = 1,
    AST_STREAM_DEFAULT_VIDEO = 2,

    /*! Define Auxiliary Streams Below */
    AST_STREAM_VIDEO_AUX1     = 100000,
    AST_STREAM_VIDEO_AUX2,    = 100001,
    AST_STREAM_VIDEO_AUX3,    = 100002,
    AST_STREAM_AUDIO_AUX1,    = 100003,
    AST_STREAM_AUDIO_AUX2,    = 100004,
    AST_STREAM_AUDIO_AUX3,    = 100005,
}

As chan_sip receives individual stream payloads and creates
ast_frames to pass into the core, each frame's stream id is marked
with the ast_channel_stream_id it belongs to.  Any channel driver or
applications that gets passed an audio or video frame belonging to
one of these newly defined auxiliary streams that does not support
it will ignore it.
```

</div>

8.4.3. Dynamic Streams

It is possible that Asterisk will need the ability to pass through streams containing media it does not understand. This can only be accomplished if both the channel negotiating the unknown media type and whatever that channel is bridged too can agree that they both understand the unknown media type and are assigning it a dynamic stream id that they both agree upon. This document does not define the negotiation of dynamic streams in Asterisk.

8.5. Ast Channel Stream API Defined

channel.c additions

```
/*! \brief Definition of opaque channel stream structure */
struct ast_channel_stream {
 /*! represents the stream type */
 enum ast_channel_stream_id id;

 struct ast_trans_pvt *writetrans;
 struct ast_trans_pvt *readtrans;

 struct ast_cap nativeformats;

 struct ast_format readformat;
 struct ast_format writeformat;
 struct ast_format rawreadformat;
 struct ast_format rawwriteformat;
};
```

channel.h additions and changes

```c
/*! \brief stream identifier structure.  Present on both ast_frame
 * and ast_channel_stream structure.
 */
enum ast_channel_stream_id {
    /*! Define all Default Streams below. */
    AST_STREAM_DEFAULT_AUDIO = 1,
    AST_STREAM_DEFAULT_VIDEO = 2,

    /*! Define Auxiliary Streams Below starting at 100000
     * Example:
     * AST_STREAM_VIDEO_AUX1     = 100000,
     */
}

void ast_channel_init_write_format(struct ast_channel *chan, enum
ast_channel_stream_id id, struct ast_format *format)

void ast_channel_init_read_format(struct ast_channel *chan, enum
ast_channel_stream_id id, struct ast_format *format)

void ast_channel_set_native_cap(struct ast_channel *chan, enum
ast_channel_stream_id id, struct ast_cap *cap)

int ast_channel_copy_readwrite_format(struct ast_channel *chan1,
struct ast_channel *chan2, enum ast_channel_stream_id id)

void ast_channel_set_read_format(struct ast_channel *chan, enum
ast_channel_stream_id id, struct ast_format *format)

void ast_channel_set_write_format(struct ast_channel *chan, enum
ast_channel_stream_id id, struct ast_format *format)

int ast_channel_get_native_cap(struct ast_channel *chan, enum
ast_channel_stream_id id, struct ast_cap *result)

int ast_channel_get_write_format(struct ast_channel *chan, enum
ast_channel_stream_id id, struct ast_format *result)

int ast_channel_get_read_format(struct ast_channel *chan, enum
ast_channel_stream_id id, struct ast_format *result)
```

8.6. Code Change Examples

This sections shows how the Ast Channel Stream API replaces existing usage in Asterisk.

Example 1: A channel driver creating a new channel and initializing the default audio stream's formats and capabilities.

Example 1 - Old
| ```
chan->nativeformats = capabilty;
chan->readformat = best_format;
chan->rawreadformat = best_format;
chan->writeformat = best_format;
chan->rawwriteformat = best_format;
``` |

Example 1 - New
| ```
ast_channel_set_native_cap(chan, AST_STREAM_DEFAULT_AUDIO,
capability);
ast_channel_init_write_format(chan, AST_STREAM_DEFAULT_AUDIO,
best_format);
ast_channel_init_read_format(chan, AST_STREAM_DEFAULT_AUDIO,
best_format);
``` |

Example 2: Setting the read format on a channel.

Example 2 - Old
| ```
ast_set_read_format(chan, format);
``` |

Example 2 - New
| ```
ast_set_read_format(chan, AST_STREAM_DEFAULT_AUDIO, format);
``` |

9. Media Format with Attributes User Configuration

With the addtion of media formats with attributes, users will need a way to represent these new formats and their attributes in their config files. This will be accomplished by the ability to define custom media types that can be used in the format allow/disallow fields. These custom media type definitions will live in codecs.conf. For people familiar with Asterisk's config structure, the best way to present this concept is with some examples.

Example 1. SILK is capable of several different sample rates. If a peer wants to negotiate only using SILK in a narrow band format, a custom format must be created to represent this.

Example 1 - codecs.conf

```
/* Limit negotiation of SILK to only 8khz and 12khz. */
[silk_nb]
type=silk
samplerates=8000,12000

/* Limit negotiation of SILK to only 16khz and 24khz */
[silk_wb]
type=silk
samplerates=16000,24000

/* Allow any SILK sample rate a device is capable of to be
negotiated */
[silk_all]
type=silk
samplerates=8000,12000,16000,24000
```

Example 1 - sip.conf

```
/* Define a peer using only the narrow band custom SILK format
definition */
[sip_peer]
type=friend
host=dynamic
disallow=all
allow=silk_nb
```

Example 2. H.264 is capable of negotiating a wide range of attributes. If specific attributes are to be negotiated, a custom format must be created to represent this.

Example 2 - codecs.conf

```
/* H.264 at vga or svga resolutions, 30 frames per second. */
[h264_custom1]
type=h264
res=vga,svga
framerate=30
```

<table>
<tr><td colspan="1">Example 2 - sip.conf</td></tr>
</table>

```
/* Define a peer using the new h264_custom1 custom format type. */
[sip_peer]
type=friend
host=dynamic
disallow=all
allow_ulaw
allow=h264_custom1
```

Notice from these examples that both the SILK and H264 custom formats are defined using fields specific to their format. Each format will define what fields are applicable to them. If there are common fields used for several different media formats, those fields should be named in a consistent way across all the media formats that use them. Every format allowing custom media formats to be defined must be documented in codecs.conf along with all the available fields.

10. Enhancing Format Negotiation During Call Setup

> This is an area of focus for our initial media overhaul efforts, but research into this area is still incomplete. Because of this, the design surrounding the ability to better negotiate media formats during call setup has not yet been defined. This will be addressed at a later date.

11. Format Renegotiation After Call Setup

> Please note that this section is incomplete. A very high level approach to format renegotiation has been outlined below, but many details pertaining to exactly how this will work are not yet defined. Format renegotiation has been marked as one of the later implementation phases and the design will be completely re-evaluated and proven before implementation.

11.1. Problem Overview

When it is at all possible, it is always better to have two bridged channels share the same media formats for their audio streams than to have to perform translation. Translation for audio is expensive, but translation for video is exponentially more expensive that audio. Because of the computational complexity involved with translating video, the concept of being able to renegotiate media after a call is estabilshed in an attempt to get the device to do the translation for us is very important. Right now Asterisk lacks this ability.

11.2. Making ast_channel_make_compatible() Smarter

Every time a channel is bridged with another channel a call to ast_channel_make_compatible() is made. This function takes the two channels to be bridged as input and figures out all the translation paths and intermediate media formats that need to be set in order for the two channels to talk to each other. With protocols like SIP, it is possible to renegotiate the call parameters after call setup has taken place. By placing a feature in ast_channel_make_compatible() that can make the two channels aware of each other's native media format before translation takes place, it is possible for one side to re-negotiate its session to switch to the same media format used by the other side. When this is possible, Asterisk is able to avoid translation completely.

11.2.1. How Renegotiation Works

At the heart of renegotiation is the introduction of a channel option called
AST_OPTION_FORMAT_RENEGOTIATE and a structure called
ast_option_renegotiate_param. The ast_format_renegotiate_param structure is passed as the
data for the AST_OPTION_FORMAT_RENEGOTIATE's query and set actions. This structure
contains both a format to renegotiate for each stream renegotation must take place on, a function
pointer containing the place a channel tech must report the result of its renegotiation attempt,
and an internal structure used to determine what action to take next after a channel tech reports
the renegotiation attempt.

On query, the ast_option_renegotiate_param structure is passed down to the channel tech pvt
containing information about all the formats and streams to renegotiate. The result of a query
request indicates whether or not the channel tech is capable of attempting renegotiation with the
formats provided or not. Queries are performed synchronously, meaning the result of a query
request must never block for a network transaction to take place.

On set, the ast_option_renegotiate_param structure is passed down to the channel tech pvt
containing both the formats and streams to renegotiate along with a place to report the result of
the renegotiation. Renegotiation is event driven, meaning that the channel tech pvt is given the
renegotation parameters and it must report back at a later time the result of the renegotiation
attempt. This allows the set operation to avoid blocking the bridge code by performing the
renegotation asynchronously.

During ast_channel_make_compatible(), if it is determined that translation is required to make
two channels compatible both channels are queried using the
AST_OPTION_FORMAT_RENEGOTIATE option and ast_option_renegotiate_param structures.
After the queries, if either of the two channels are capable of renegotiating the set action is used
on best candidate to attempt the renegotiation. If the channel used for the first renegotiation
attempt reports a failure, a second attempt at renegotiation may take place for the bridged
channel if neither channel has hung up.

11.2.2. Renegotiation with Intermediary Translation

- Make Compatible Flow of Events
    - ast_channel_make_compatible() is invoked
    - read and write formats are different between channels for at least one stream
    - translation paths are built for streams requiring translation
    - query to AST_OPTION_FORMAT_RENEGOTIATE is made on both channels
    - if candidate for renegotation exists, renegotiation parameters are set to the channel using
      AST_OPTION_FORMAT_RENEGOTIATE
    - channels are bridged
- Asynchronous Renegotation Flow of Events
    - channel tech is set with renegotation paramters using AST_OPTION_FORMAT_RENEGOTIATE
    - channel tech attempts renegotiation and reports result to renegotiation parameter result function
        - on SUCCESS: new format is set for renegotated stream and translation path goes away
        - on FAILURE: result function attempts renegotation with bridged channel if possible, else translation must remain

11.2.3. Renegotiation with no Intermediary Translation

- Make Compatible Flow of Events
    - ast_channel_make_compatible() is invoked
    - channel's read and write formats are different for at least one stream
    - **NO** translation path is possible to make channels compatible
    - query to AST_OPTION_FORMAT_RENEGOTIATE is made to both channels
    - if best candidate for renegotiation is found, renegotiation parameters are set to the channel using
      AST_OPTION_FORMAT_RENEGOTIATE

- channel is bridged
- media for incompatible streams are blocked for a period of time while renegotiation takes place
- Asynchronous Renegotiation Flow of Events
  - channel tech is set with renegotiation parameters using AST_OPTION_FORMAT_RENEGOTIATE.
  - channel tech attempts renegotiation and reports result to renegotiation parameter result function
    - on SUCCESS: new format is set for renegotiated stream and translation path goes away
    - on FAILURE: result function attempts renegotiation with bridged channel if possible
  - if renegotiation fails on both channels, depending on the stream in question media is either indefinitely blocked or both channels are hung up

12. Implementation Phases

With a project of this size, it is important to break down the implementation into manageable phases. Each phase of development contains a set of steps which act as milestones. These steps must be small enough to be attainable within a week to two week period but complete enough to not break any Asterisk functionality once they are introduced. Once a step is complete, it should be reviewed and committed into trunk. This allows progress to be made in a maintainable way.

12.1. Phase 1: Re-architect how media is represented and how translation paths are built

From the user perspective, no functionality changes will be present during this phase.

- Step 1
  - Define new format unique ID system using numbers rather than bits. Allow this definition to remain unused during this step except by the new APIs.
  - Create Ast Format API + unit tests.
  - Create Ast Capibility API + unit tests.
  - Create IAX2 Conversion layer for ast_format and ast_cap objects. Create unit tests and leave this layer inactive until conversion to new APIs takes place.

- Step 2
  - Define translation cost table.
  - Replace current matrix algorithm with new matrix algorithm using predefined costs from table.
  - Continue to use computational costs for tie breaking translators with identical src and dst formats.
  - Create table for mapping format ids to matrix index values. This is required once the conversion from the format bit field representation to a numeric value takes place and will allow for a smoother transition.

- Step 3
  - Replace old format unique ID system with the new system. This will temporarily break all asterisk media functionality.
  - Add media functionality back into Asterisk by replacing all instances of format_t with ast_format and ast_cap.
  - Completely remove format_t type def.

12.2. Phase 2: Exercise the functionality introduced by formats with attributes

This is done by introducing the SILK codec and allowing H.264 to be negotiated with format attributes.

- Step 1
  - Define SILK format in Asterisk.
  - Create SILK format attribute interface.
  - Make SILK translators to and from signed linear.
  - Add the ability to define custom media formats with attributes in user configuration.
  - Extend the rtp mapping code to allow chan_sip to advertise SILK appropriately in SDPs.

- Step 2
  - Create H.264 format attribute interface.
  - Extend codecs.conf to allow custom H.264 format definitions.
  - Extend chan_sip to be able to correctly advertise and negotiate H.264 with attributes in SDPs.

12.3. Phase 3: Extend Asterisk to handle multiple media streams

- Step 1

- Create Ast Channel Stream API
- Define default audio stream by replacing current audio stream formats and translators on a channel with an ast_channel_stream structure.
- Define default video stream by introducing a new ast_channel_stream structure used solely for negotiating the primary video stream.

- Step 2
  - Add the stream id field to the ast_frame structure.
  - Block the ability to read anything other than the default streams with all current channel drivers and applications.
  - Introduce new ast_read functionality for reading auxiliary streams when it is explicitly requested.

- Step 3
  - Exercise the new ability to build video translation paths using an FFMPEG addon translation module.

12.4. Phase 4: Format Renegotiation after call setup

Allowing calls to renegotiate their media formats after call setup is perhaps the most practical functionality introduced by this project. Due to the way multiple media streams will be represented in Asterisk, this ability to represent multiple streams is prerequisite for format renegotiation be implemented correctly. That is the primary reasoning for pushing back the introduction of this functionality to a later phase.

- Step 1
  - Re-evaluate design. Define use cases and prove concept with a set of sequence diagrams.
  - Test interoperability of renegotiation use cases using sipp scenarios against common SIP devices.

- Step 2
  - Implement core functionality changes required to detect and attempt format renegotiation with channel techs.
  - Implement chan_sip configuration options and functionality required to allow format renegotiation triggered by the Asterisk core to occur after call setup.

Media Overhaul

1. Project Requirements

> ⛔ This section is incomplete.

1.1. Relevant Problems That Exist Today

- Codec negotiation (both with Asterisk, and across a bridge)
  - Support for audio codecs with attributes (SILK)
  - Support for video codecs with attributes
- Limitation on the number of codecs Asterisk can support
- Translation paths are audio specific (with no concept of attributes)
- There is no way to renegotiate codecs after a call is up
- Conferencing is limited to 8 kHz
- There is no way to easily get Asterisk to pass through a media type that it does not understand (proprietary data)
- Once Asterisk supports codecs with attributes, users will need to be able to specify codecs with attributes
- Asterisk is not able to handle a call with more than one audio/video/text stream (only one stream per type).
- Asterisk has no RTCP support relevant to audio and video synchronization
- Asterisk does not support Gtalk video

1.2. Phase 1 Requirements

Rework media representation completely across all of Asterisk, while maintaining existing functionality. Only add functionality that is required to exercise what has been done.

- Design a new way to represent codecs
  - translation interface
  - capabilities
  - ast_frame handling

- Initial call setup codec negotiation
- everything that touches media ...
- Exercise what we have done so far
  - Add support for SILK (and its attributes)
  - Add support for H.264 attributes
- Custom format definitions with attributes (for setting preferences)

1.3. Phase Later Requirements

- Codec re-negotiation
- Improved conferencing (dynamic sample rate support)
- Video transcoding (an implementation that proves it works)
- A&V sync in RTCP (research required)
- GTalk Video Support
- Support for unknown media types for pass-through (research required)
- Support for more than one stream of the same type (audio/video/text)

SIP Security Events

Project Requirements

Asterisk 1.8 introduced the security events API, which is defined in `include/asterisk/security_events.h`. The SIP channel driver, `chan_sip`, needs to be updated to take advantage of this API for reporting events that potentially have security implications.

> ⊖ This section is incomplete.

Summary

Problem: Attacks or unintentional configuration problems on various IP-based communications mechanisms (VoIP signalling, or IM, or media channels, or manager or...?) can lead to conditions in Asterisk that create denial-of-service circumstances, permit brute-force password attacks, or message/media confusion. Other attack results, including application or system security issues, may be exposed as a result of high-volume packet floods or malformed packets from trusted or untrusted sources.

Root problem: Asterisk has no sophisticated method of defining packet or signaling profile groups and taking action based on desired or undesired patterns.

Proposed Solution: Implement attack detection on a per-channel driver basis, reporting up to a generic socket API for blocking/logging. Create named ACLs for passive as well as dynamic packet examination/filtering. Create dynamic filter capability based on attack detection, and allow all channels to utilize a common filtering framework. Permit external exportation of passive and dynamic filters to external third-party attack mitigation tools.

Details

Details on threats: There are several vulnerabilities that can occur when exposing the various IP-based protocol stacks to the public (or private) Internet(s). These include but are not limited to:

1. Denial of Service from one or more attack origin IP addresses
   a. Random packet garbage
   b. Syntactically valid but session-invalid packet garbage
   c. Session-valid but malformed packet garbage

2. Brute-force password attacks on VoIP protocols from one or more IP addresses
3. System penetration via buffer overflow or other methods

Other problems that a robust filtering/rate-limiting package may provide:

1. Protection against DDoS amplification
2. Protection against runaway UA messaging
3. Arbitrary re-direction of VoIP call signaling to alternate servers

Resolved:

- It is desirable to have named access-lists that are available across all channel drivers

- It is desirable to have both IP-generic (flooding) security as well as protocol-specific filtering built into Asterisk

- It is desirable to have each channel driver able to define what is "bad" and then describe what actions to take in the event of "bad" packets.

- It is desirable to have a packet filtering mechanism that is re-usable across all channel types.

- It may be desirable to have the ability to export actions to external programs based on filters defined in the channels, as whatever protective measures that are built into Asterisk may be insufficient or otherwise inappropriate based on the attack method.

- ACLs and actions should be capable of acting on channel-specific events, IP src/dst pattern matching, port pattern matching, session pattern matching, volume pattern matching

- Actions based on "bad" events should be both passive (standard ACLs) and active (dynamic filters that change based on behaviors)

- Actions and filters should **APPLY IN BOTH DIRECTIONS OF IP COMMUNICATIONS.** This means as an example and somewhat counter-intuitively, that SIP packets outbound from Asterisk to some remote endpoint should be able to be limited. This is in response to some remote devices being incapable of handling large volumes of certain packet types, or having call volumes (CPS) of a particular limit. It also helps reduce the (unusual) chance of Asterisk being an amplifier in DoS attacks.

- Dynamic filters should be stored across instantiations of Asterisk, including timers.

- External filters should be flushed and re-added at instantiations of Asterisk BEFORE channels are loaded.

Programming:

- RED and other dynamic algorithms for filtering may be found in ipfw, which is BSD-licensed (note: Luigi Rizzo is/was an ipfw programmer?)

- named and grouped ACLs are part of ipfw also

- All syntaxes and configurations should support IPv4 and IPv6 addresses and methods to avoid the nightmare of backwards-reengineering when IPv6 is embedded

References:

- Kamailio RED and Rate-Limit:
  - http://kamailio.org/docs/modules/1.4.x/ratelimit.html
- Kamailio PIKE module:
  - http://kamailio.org/docs/modules/1.4.x/pike.html
- Postfix Anvil daemon
  - http://www.postfix.org/anvil.8.html

Channel-specific notes:

SIP:

- "calls per second" - should apply only to INVITEs?

- RTP media dynamic filters: should support dynamic updating of filters based on RTP expected from one or more endpoints. This should block the problem with unauthorized media insertion, or media cross-confusion.

- methods - match on Methods as described by http://www.iana.org/assignments/sip-parameters

- actions: all actions have an optional ability to also be applied to dynamic filters. All actions have the optional ability to log a message. All actions have the ability to inform external programs.

Action types of -

- permit
- deny (drop silently)
- deny (with reply to non-local end)

- replies: replies can be any SIP message in the 3xx, 4xx, 5xx, or 6xx fields as noted by http://www.iana.org/assignments/sip-parameters. Note that 3xx replies need to be able to contain different redirection pointers to allow load shedding.

Sample configuration file

```
[general]
; blah blah blah
;
; flush old rules - should be done here? specifiable?


; How often do we write out rules to astdb? In seconds.
astdb-inteval=20


; stanzas can be of two "types" - "filter" or "action"


[keys]
; maybe we put in this file the lists of all the certs that
;  we use elsewhere in Asterisk as a centralized config.
;


[filterA]
type=filter
;
;
; src-range - an IP address range, in CIDR notation, which describes
;  the origins of an IP packet.  One IP address range per line,
;  though there can be multiple lines describing address ranges.
;  IP addresses specified without CIDR notation will be interpreted
;  as /32 ranges automatically.
;
src-range=128.0.0.0/8
src-range=124.0.0.0/8
;
;
; dst-range - an IP address range, in CIDR notation, which describes
;  the destination of an IP packet.  One IP address range per line,
```

```
;    though there can be multiple lines describing address ranges.
;    IP addresses specified without CIDR notation will be interpreted
;    as /32 ranges automatically.
;    **IF NO DST-RANGE IS SPECIFIED, ASTERISK WILL ALWAYS CREATE A
;    LIST OF /32's DESCRIBING ALL IP ADDRESSES LOCAL TO THE SYSTEM**
;    (excluding localhost.)  If you create a dst-range, be careful to
;    specifically include all your interfaces otherwise only the
specified
;    dst-range will be used.  This also means if you create a filter
that
;    is for some reason incomplete, it will fire for an unpredictably
large
;    number of packets since they will all match at least one of your
;    interfaces as a destination.
;
;dst-range=10.0.0.0/8
;
;
; Proto is the protocol of the packets.  Currently, only TCP and UDP
;    are supported since those are the only protocols Asterisk should
;    be seeing!
;
proto=UDP,TCP
;
;
; Port ranges.  Can be single integers (0-65535), ranges,
;    or a comma delimited list of both.  There can be multiple
;    ports descriptions, which are accretive.
;
ports=5060-5062,50600
;
;
; Deep packet data.
;
; authentication-name is the username or authentication identifier
;    that is used in the authentication portion of the protocol being
;    used.
;       SIP: This is the Contact: name to the left of the @ sign
;      IAX2: This is username IE in the start of the session.
;      H323: This is **blah in the **blah.
;      SCCP: This is **blah in the **blah.
;       AMI: This is the Username: portion of the AMI auth
transaction.
;    Note that using authentication-name is more computationally
intensive,
;      since the packet reaches the channel driver and has to be
```

```
examined after
;    what is often a "half-authentication" process.
;   Special keyword: you can specify the special username of "^ANY"
which
;    will create a dynamic list of authentication names over the
time-interval
;    you specify in the filter (see below.)  This means that you can
try to
;    catch brute-force attackers who are attacking particular
;authentication-name=alice
;


;
;
; Volumes.  While simple packet filters may be used, additional more
complex
;   algorithms can be invoked as additional trigger methods included
in filters.
;   These are sometimes related specifically to channel-types, or
sometimes
;   they are more generic.


signaling packets per second
initiations or commands per second
     manager commands
     usernames on SIP, IAX2, ami, etc.
     INVITE, UPDATE, REGISTERs, etc (see iana methods list)
     IAX2 new session requests
     IAX2 message elements
     H.323 new call requests
     XMPP messaging requests
     MGCP
     http server
     SCCP

media mapping (include unmapping rule, include nostore options)
initiation connection failures:
     usernames on SIP, IAX2, ami, etc.
     bad call creations per second
     bad TCP socket connections
     bad TCP/TLS socket connections (bad keys, no data, etc?)
bad auth per second
     bad username/password for manager
     bad key exchanges for TLS
     bad http-digest exchange for SIP
     bad password on IAX
```

```
bad packets per second
     malformed packets on SIP, IAX, H.323, etc.



[actionB]
;
; Set the type to "action" to specify that this is something that
;   describes a set of things to do.
;
type=action
;
; actions can be:
;   accept
;   drop
;   drop-response
;
; "accept" will allow the packet through unmodified.
; "drop" will deny the packet, and send no message back
;    to the originator.
; "drop-response" will send a message
;
action=drop-response
;
;
; drop-method describes how packets are dropped. There are various
;   ways to select packets to drop, ranging from completely dropping
;   all packets, to
drop-method=

;
; sip-drop-response
;
; This describes the specific behaviors we can have SIP use to
;   respond to action requests.  Each protocol has its own set of
;   specific response capabilities.
;
; The global variable $SIP-DROP-RESPONSE can be used here, in
;   case your system has some logic that gets executed in dialplans
;   that would set that value from time to time.  This could be
;   used to set a different failover/load shedding system destination
;   via a 3xx reply.
;
; NOTE: Some 3xx replies are handled specially if specified.  They
will look
;   at the value stored in sip-drop-response in a special way, as
```

```
described
;   below. Note that the "sip-drop-response-msg" message will take
precedence over a
;   global variable $SIP-DROP-RESPONSE-MSG which can be used to
semi-dynamically
;   change the forwarding destination.
;
;     301=Moved Permanently.  If you include a full URI (i.e.:
"alice@foo.com")
;         then that full URI is what will be sent back as the
Contact.   If you
;         only include a domain portion, the original user from the
INVITE will
;         be filled in to the Contact: with the new domain as the
suffix.
;     301=Moved Permanently.  If you include a full URI (i.e.:
"alice@foo.com")
;         then that full URI is what will be sent back as the
Contact.   If you
;         only include a domain portion, the original user from the
INVITE will
;         be filled in to the Contact: with the new domain as the
suffix.
;     305=Use Proxy.  Include only a domain name of a new proxy.
;     380=Alternative Service.  Include only a domain name of a new
proxy.
;
sip-drop-response=305
sip-drop-response-msg=poor-impulse-control@floodproof.domain.net




; ext-action - External Action
;   syntax: command name
;   example: /bin/sh my-iptables-script.sh
;
; The ext-action identifier allows various external programs
;   to be informed of events and then to act on those events
;   as necessary.  This may be something like a script that calls
;   an entry into iptables, or perhaps creates an ACL on some
;   more sophisticated off-platform firewall component.
;
; These are the elements that are written out to the
;   system call.  These are filled in by the Asterisk security
;   event system.  Your script/application can listen to or
```

```
;   ignore the fields as necessary.
;
;   ARG1 = IP address of violation origin/filter request (includes
CIDR netmask)
;   ARG2 = destination IP address of violation origin/filter request
(includes CIDR netmask)
;   ARG3 = port destination of violation packet
;   ARG4 = prototype of violation packet (TCP, UDP)
;   ARG5 = action as reported by internal Asterisk modules (ACCEPT,
DROP)
;   ARG6 = number of iterations that has triggered pattern match
;   ARG7 = duration expectation of action (0 for infinite)
;   ARG8 = interface name of reporting interface (i.e.: eth0, eth1,
etc.)
;   ARG9 = longint counter which is incremented by +1 every time
ext-action is called
;
ext-action="iptables -A INPUT -s ${ARG1} -j DROP"
;
; Other examples:
;ext-action="iptables -A INPUT -p ${ARG4} -s ${ARG1} --dport ${ARG3}
-m limit --limit 10/min -j ACCEPT;  iptables -A INPUT -p ${ARG4} -s
${ARG1} --dport ${ARG3} -j DROP"
;
; Which translates to this in a situation where 172.16.3.2 on SIP is
the attacker:
;   iptables -A INPUT -p UDP -s 172.16.3.2/32 --dport 5060 -m limit
--limit 10/min -j ACCEPT;  iptables -A INPUT -p UDP -s 172.16.3.2/32
--dport 5060 -j DROP
;
; logging




[filterC]
; filters are parsed in order, like with the dialplan.
;
; Each filter includes:
;   pattern - what is being matched?
;   action  - what is done when a match is made?
;   logging - what is the notification method?
```

```
include=filterC
include=actionB

logging constantly, logging on first, logging on interval?

possibly include pcap output


; Additional CLI commands
;
```

```
;
```

T.38 Gateway

Functional Requirements

Use case 1:

```
(A) Calling T.38 endpoint <-->
  (B) Channel Driver 1<-->
    (C) Asterisk Core <-->
  (D) T.38 Gateway Object <-->
  (E) Channel Driver 2<-->
(F) Called T.30 endpoint
```


In this use case, the calling endpoint is T.38 equipped and the called endpoint is not (supports

T.30 only); in addition, the calling endpoint is properly configured to not initiate T.38 re-INVITEs itself, but relies on the receiving gateway (Asterisk+T.38 Gateway) to do so.

during call setup, called endpoint is ringing, gateway does nothing

when called endpoint has answered, gateway listens for FAX preamble (using V.21 detector) generated by called endpoint

if preamble is not detected after 15 seconds from call answer, gateway removes itself from the audio path (this may need to be optional)

if preamble is detected, gateway mutes audio path in both directions (generating silence towards both endpoints). gateway waits one second, then constructs T.38 negotiation request as a control frame, using parameters previously set via FAXOPT() dialplan function, and sends control frame to core

if T.38 negotiation fails (times out, or is rejected), gateway removes itself from the audio path (thus clearing the 'mute' state in both directions)

if T.38 negotiation succeeds (request is accepted), gateway constructs T.38 session in FAX stack, feeds it negotiated parameters, and starts session; if this fails, gateway causes channel(s) to be hung up and reports appropriate errors

while T.38 session is active, gateway receives T.38 media frames from calling endpoint and passes them to FAX stack; gateway receives T.38 media frames from FAX stack and sends them to calling endpoint. gateway also receives audio frames from called endpoint and passes them to FAX stack, and receives audio frames from FAX stack and sends them to called endpoint

when T.38 session is complete, gateway sends T.38 negotiation request to revert to audio mode

if audio mode reversion fails, gateway causes channel(s) to be hung up

if audio mode reversion succeeds, gateway removes itself from audio path

Use case 2:

```
(A) Calling T.38 endpoint <-->
  (B) Channel Driver 1 <-->
    (C) Asterisk Core <-->
    (D) T.38 Gateway Object <-->
  (E) Channel Driver 2 <-->
(F) Called T.30 endpoint
```

In this use case, the calling endpoint is T.38 equipped and the called endpoint is not (supports T.30 only); in addition, the calling endpoint
is improperly configured to initiate T.38 re-INVITEs itself, not waiting for the receiving gateway to do so.

during call setup, called endpoint is ringing, gateway does nothing

when called endpoint has answered, gateway listens for FAX preamble (using V.21 detector) generated by called endpoint

if preamble is not detected after 15 seconds from call answer, gateway removes itself from the audio path (this may need to be optional)

if preamble is detected, gateway mutes audio path in both directions (generating silence towards both endpoints). gateway waits one second,
and during this time a T.38 negotiation request is received from the calling endpoint; gateway processes this request using parameters
previously set using FAXOPT() dialplan function

if the T.38 negotiation request cannot be accepted, gateway sends a T.38 negotiation failure response, and removes itself from the audio path
(thus clearing the 'mute' state in both directions)

if T.38 negotiation succeeds (request can be accepted), gateway sends a T.38 negotiation success response, constructs T.38 session in FAX stack,
feeds it negotiated parameters, and starts session; if this fails, gateway causes channel(s) to be hung up and reports appropriate errors

while T.38 session is active, gateway receives T.38 media frames from calling endpoint and passes them to FAX stack; gateway receives T.38 media frames from FAX stack and sends them to calling endpoint. gateway also receives audio frames from called endpoint and passes them to FAX
stack, and receives audio frames from FAX stack and sends them to called endpoint

when T.38 session is complete, gateway sends T.38 negotiation request to revert to audio mode
(this may be received from calling endpoint first,
though)

if audio mode reversion fails, gateway causes channel(s) to be hung up

if audio mode reversion succeeds, gateway removes itself from audio path

Use case 3:

```
(A) Calling T.30 endpoint <-->
  (B) Channel Driver 1 <-->
    (C) Asterisk Core <-->
    (D) T.38 Gateway Object <-->
  (E) Channel Driver 2 <-->
(F) Called T.38 endpoint
```

In this use case, the calling endpoint is not T.38 equipped but the called endpoint is. In addition, the called endpoint is improperly configured to rely on the calling endpoint to initiate T.38 re-INVITEs.

during call setup, called endpoint is ringing, gateway does nothing

when called endpoint has answered, gateway listens for FAX preamble (using V.21 detector) generated by called endpoint

if preamble is not detected after 15 seconds from call answer, gateway removes itself from the audio path (this may need to be optional)

if preamble is detected, gateway mutes audio path in both directions (generating silence towards both endpoints). gateway waits one second, then constructs T.38 negotiation request as a control frame, using parameters previously set via FAXOPT() dialplan function, and sends control frame to called endpoint

if T.38 negotiation fails (times out, or is rejected), gateway removes itself from the audio path (thus clearing the 'mute' state in both

directions)

if T.38 negotiation succeeds (request is accepted), gateway constructs T.38 session in FAX stack, feeds it negotiated parameters, and starts
session; if this fails, gateway causes channel(s) to be hung up and reports appropriate errors

while T.38 session is active, gateway receives T.38 media frames from called endpoint and passes them to FAX stack; gateway receives T.38
media frames from FAX stack and sends them to called endpoint. gateway also receives audio frames from calling endpoint and passes them to FAX
stack, and receives audio frames from FAX stack and sends them to calling endpoint

when T.38 session is complete, gateway sends T.38 negotiation request to revert to audio mode

if audio mode reversion fails, gateway causes channel(s) to be hung up

if audio mode reversion succeeds, gateway removes itself from audio path

Use case 4:

```
(A) Calling T.30 endpoint <-->
  (B) Channel Driver 1 <-->
    (C) Asterisk Core <-->
    (D) T.38 Gateway Object <-->
  (E) Channel Driver 2 <-->
(F) Called T.38 endpoint
```

In this use case, the calling endpoint is not T.38 equipped but the called endpoint is. In addition, the called endpoint is properly configured to initiate T.38 re-INVITEs itself.

during call setup, called endpoint is ringing, gateway does nothing

when called endpoint has answered, gateway listens for FAX preamble (using V.21 detector) generated by called endpoint

if preamble is not detected after 15 seconds from call answer, gateway removes itself from the audio path (this may need to be optional)

if preamble is detected, gateway mutes audio path in both directions (generating silence towards both endpoints). gateway waits one second, and during this time a T.38 negotiation request is received from the called endpoint; gateway processes this request using parameters previously set using FAXOPT() dialplan function

if the T.38 negotiation request cannot be accepted, gateway sends a T.38 negotiation failure response, and removes itself from the audio path
(thus clearing the 'mute' state in both directions)

if T.38 negotiation succeeds (request can be accepted), gateway sends a T.38 negotiation success response, constructs T.38 session in FAX stack,
feeds it negotiated parameters, and starts session; if this fails, gateway causes channel(s) to be hung up and reports appropriate errors

while T.38 session is active, gateway receives T.38 media frames from called endpoint and passes them to FAX stack; gateway receives T.38
media frames from FAX stack and sends them to called endpoint. gateway also receives audio frames from calling endpoint and passes them to FAX
stack, and receives audio frames from FAX stack and sends them to calling endpoint

when T.38 session is complete, gateway sends T.38 negotiation request to revert to audio mode

if audio mode reversion fails, gateway causes channel(s) to be hung up

if audio mode reversion succeeds, gateway removes itself from audio path

Asterisk Developer Conference Call

The Asterisk Developer Conference call is held every other Thursday.

Please note that this call staggers every other instance to accommodate participation from other time zones.

Shared calendar URL: http://devcon.digium.com/calendar/AsteriskDevCall

The Asterisk Developer Conference Call can be accessed via the following methods:

sip:100@devcon.asterisk.org
SKYPE: devcall.digium
pstn: 1.877.274.7510
Digium internal: 500003

Bridge: Shared calendar URL: http://devcon.digium.com/calendar/AsteriskDevCall
Pin Code: Shared calendar URL: http://devcon.digium.com/calendar/AsteriskDevCall

Please join us on IRC at #asterisk-dev during the call for real time communication and file sharing.

The format for the call is:

00:00 - 05:00: Welcome & Announcements
05:00 - 20:00: Old Business
20:00 - 40:00: New Business
40:00 - 60:00: Open Floor

All participants are muted by default and should press "*" to access the conference menu followed by "1" to unmute in order to speak.

Please make best efforts to join from handsets or use a headset in conjunction with a software phone or SKYPE to maximize audio quality.

We look forward to your participation!

Asterisk Developer Call - 01272011 - 1700 EST

Digium Developer Conference Bridge:

Bridge: 15071

PIN: 35806

Proposed Agenda:

Welcome & Announcements: Bryan Johns / Russell Bryant / Leif Madsen

- Introduction of Digium participants
- Overview of developer call format and future sessions

Old Business: None. Inaugural call.

New Business:

- Overview of current 1.10 planned features and development status
- Overview of David Vossel's media-specific contribution efforts and associated updates

Open Floor

Notes:

1. Russell Bryant delivers an update on Asterisk 1.10 development roadmap
2. Discussion of major overhaul of media handling (defer details to David Vossel overview)
3. Fax handling improvements explanation. Adding T.38 gateway support
4. David Vossel overview of new media handling architecture - Review of proposal from the wiki
5. David Vossel and Russell Bryant give an overview of the bamboo test suite and its contribution to undertake larger projects
6. Suggestion made to solicit new business items from attendees each week for following week's call

Asterisk Developer Call - 02032011 - 1000 EST

Digium Developer Conference Bridge:

Bridge: 15071

PIN: 35806

Proposed Agenda:

Welcome & Announcements: Bryan Johns / Russell Bryant / Leif Madsen

- Introduction of Digium participants
- Overview of developer call format and future sessions

Old Business: None. Inaugural call.

New Business:

- Overview of current 1.10 planned features and development status
- Overview of David Vossel's media-specific contribution efforts and associated updates

Open Floor

Notes:

1. Russell Bryant delivers an update on Asterisk 1.10 development roadmap
2. Discussion of major overhaul of media handling (defer details to David Vossel overview)
3. Fax handling improvements explanation. Adding T.38 gateway support
4. David Vossel overview of new media handling architecture - Review of proposal from the wiki
5. David Vossel and Russell Bryant give an overview of the bamboo test suite and its contribution to undertake larger projects
6. Suggestion made to solicit new business items from attendees each week for following week's call

Asterisk Developer Call - 02102011 - 1700 EST

Digium Developer Conference Bridge:

Bridge: 15071

PIN: 35806

Proposed Agenda:

Welcome & Announcements: Bryan Johns / Russell Bryant / Leif Madsen

- Introduction of Digium participants
- Overview of developer call format and future sessions

Old Business:

- Media handling revision update

New Business:

- Discussion of developer call purpose
- Discussion of developer call frequency
- Call for future conference call topics

Open Floor

Notes:

Asterisk Developer Call - 02172011 - 1000 EST

Digium Developer Conference Bridge:

Bridge: 26052

PIN: 35806

Proposed Agenda:

Welcome & Announcements: Bryan Johns / Malcolm / Leif Madsen

Old Business:

- None discussed

New Business:

- Detailed discussion of call frequency and call content.
- Suggestions made to alternate weekly between calls focused on development topics and calls focused on usage topics.
- Usage calls will be combined with Asterisk Tech Tips webinars to be held every other week.
- Developer calls will be held every other week and will contain predetermined topic discussions.

Open Floor

- Malcolm gives abbreviated update on 1.10 media handling overhaul and availability of wideband conferencing. Encourages attendees to test it.

Notes:

Asterisk Developer Call - 03102011 - 1700 EST

Digium Developer Conference Bridge:

Bridge: 15071

PIN: 35806

Proposed Agenda:

Welcome & Announcements: Bryan Johns / Russell Bryant / Leif Madsen

- Introduction of Digium participants
- Overview of developer call format and future sessions

Old Business:

- Media handling revision update

New Business:

- Discussion of developer call purpose
- Discussion of developer call frequency
- Call for future conference call topics

Open Floor

Notes:

Asterisk Developer Call - 05122011 - 1700 EST

Digium Developer Conference Bridge:

Bridge: 15071

PIN: 35806

Proposed Agenda:

Welcome & Announcements: Bryan Johns / Russell Bryant

- Introduction of Digium participants

Old Business:

None

New Business:

- AstriCon 2011
- Asterisk 1.8 development update
- Asterisk 1.10 development update
- Migration of 1.4/1.6 to security maintenance only

Open Floor

Notes:

Asterisk Developer Call - 05262011 - 1000 EST

Digium Developer Conference Bridge:

Bridge: 26052

PIN: 35806

Proposed Agenda:

Welcome & Announcements: Bryan Johns

Old Business:

- None discussed

New Business:

- General development update
- Discussion of new tech tips topics

Open Floor

Notes:

Asterisk Developer Call - 06232011 - 1000 EST

Digium Developer Conference Bridge:

Bridge: 26052

PIN: 35806

Proposed Agenda:

Welcome & Announcements: Bryan Johns

Old Business:

- None discussed

New Business:

Open Floor

Notes:

Ben Klang only participant in attendance. Brief conversation about 1.2 connectivity issues to 1.8.2. Discussion tabled and call concluded.

Asterisk Versions

There are multiple supported feature frozen releases of Asterisk. Once a release series is made available, it is supported for some period of time. During this initial support period, releases include changes to fix bugs that have been reported. At some point, the release series will be deprecated and only maintained with fixes for security issues. Finally, the release will reach an End of Life, where it will no longer receive changes of any kind.

The type of release defines how long it will be supported. A Long Term Support (LTS) release will be fully supported for 4 years, with one additional year of maintenance for security fixes. Standard releases are supported for a shorter period of time, which will be at least one year of full support and an additional year of maintenance for security fixes.

The following table shows the release time lines for all releases of Asterisk, including those that have reached End of Life.

Release Series	Release Type	Release Date	Security Fix Only	EOL
1.2.X		2005-11-21	2007-08-07	2010-11-21
1.4.X	LTS	2006-12-23	2011-04-21	2012-04-21
1.6.0.X	Standard	2008-10-01	2010-05-01	2010-10-01
1.6.1.X	Standard	2009-04-27	2010-05-01	2011-04-27
1.6.2.X	Standard	2009-12-18	2011-04-21	2012-04-21
1.8.X	LTS	2010-10-21	2014-10-21	2015-10-21
1.10.X	Standard	2011-10-12	2012-10-12	2013-10-12

New releases of Asterisk will be made roughly once a year. Releases marked as LTS releases will be much less frequent. Within a given release series that is fully supported, bug fix updates are provided roughly every 4 weeks. For a release series that is receiving only maintenance for security fixes, updates are made on an as needed basis.

If you're not sure which one to use, choose either the latest release for the most up to date features, or the latest LTS release for a platform that may have less features, but will usually be around longer.

The schedule for Asterisk releases is visualized below (which is subject to change at any time):

FOSDEM 2011 Open Source Telephony Devroom

Overview

An Open Source Telephony Devroom has been organized at FOSDEM 2011 in Brussels on Sunday, February 6th, 2011. We will be holding a day full of presentations on development topics in the area of open source telephony. 30 minute presentations will be made from 09:00 until 17:00. We have been assigned room AW1.124 - provided with a projector, WiFi, and 59 seats.

Proposals have already been submitted and accepted. The following schedule will be observed.

Please direct discussion about this devroom to the telephony-devroom mailing list hosted on http://lists.fosdem.org/. If you would like to contact the devroom organizer directly, please contact Russell Bryant <russell at digium.com>.

Feel free to forward this along to any people or mailing lists that you think would be interested in this event.

Schedule

Time	Session	Speaker(s)
09:00	Welcome to the Open Source Telephony Devroom	Russell Bryant
09:15	Introduction to Asterisk Development	Russell Bryant
09:45	Session Q&A	
09:55	Break	
10:00	Digital PSTN Connectivity with Asterisk	Jakub Klausa
10:30	Session Q&A	
10:40	Break	

10:45	Mobicents 2.0, The Open Source Java Communication Platform	Jean Deruelle
11:15	Session Q&A	
11:25	Break	
11:30	Scaling Location Services in Large SIP Networks with Kamailio	Henning Westerholt, Marius Zbihlei
12:00	Session Q&A	
12:10	Break	
12:15	Unifying SIP and Web Worlds with Lua	Daniel-Constantin Meirla
12:45	Session Q&A	
12:55	Break	
13:00	XiVO IPPBX OpenHardware	Xavier Carcelle
13:30	Session Q&A	
13:40	Break	
13:45	SIP Communicator: Building a Multi-Protocol Multi-OS Communications Client	Emil Ivov
14:15	Session Q&A	
14:25	Break	
14:30	Asterisk SCF Development Interfaces	Kevin P. Fleming
15:00	Session Q&A	
15:10	Break	
15:15	Developing rich VoIP Applications with SIPSIMPLE SDK	Saúl Ibarra Corretgé
15:45	Session Q&A	
15:55	Break	
16:00	Unified Communications - Future (Yate and YateClient)	Diana Cionoiu
16:30	Session Q&A	
16:40	Concluding Remarks	
17:00	End	

Location

FOSDEM 2011 will take place at the ULB Campus Solbosch, Brussels, Belgium.

Avenue Franklin D. Roosevelt, 50
1050 Bruxelles

The Open Source Telephony Devroom will be in room 1.124 in building AW as shown on this map:

Main FOSDEM entrance

Asterisk 1.10 Documentation

Asterisk 1.10 is still in development. This is just a staging area for some documentation that is in progress and specific to Asterisk 1.10.

ConfBridge 1.10

Overview

Asterisk, since its early days, has offered a conferencing application called MeetMe (app_meetme.so). MeetMe provides DAHDI-mixed software-based bridges for multi-party audio conferencing. MeetMe is used by nearly all Asterisk implementations - small office, call center, large office, feature-server, third-party application, etc. It has been extremely successful as an audio bridge.

Over time, several significant limitations of MeetMe have been encountered by its users. Among these are two of distinction: MeetMe requires DAHDI for mixing, and is thus limited to 8kHz (PSTN) audio sampling rates; and MeetMe is delivered in a fairly static form, it does not provide extensive configuration options.

To address these limitations, a new conferencing application, based upon the ConfBridge application introduced in Asterisk 1.6.0, is now available with Asterisk 1.10. This new ConfBridge application replaces the older ConfBridge application. It is not intended to be a direct replacement for MeetMe, it will not provide feature parity with the MeetMe application. Instead, the new ConfBridge application delivers a completely redesigned set of functionality that most users will find more than sufficient, and in many ways better, for their conferencing needs.

ConfBridge Concepts

ConfBridge provides four internal concepts:

1. Conference Number
2. Bridge Profile
3. User Profile
4. Conference Menu

A **Conference Number** is a numerical representation for an instance of the bridge. Callers joined to the same conference number will be in the same conference bridge; they're connected. Callers joined to different conference numbers are not in the same conference bridge; they're separated. Conference Numbers are assigned in the dialplan. Unlike MeetMe, they're not pre-reserved.

A **Bridge Profile** is a named set of options that control the behavior of a particular conference bridge. Each bridge must have its own profile. A single bridge cannot have more than one Bridge Profile.

A **User Profile** is a named set of options that control the user's experience as a member of a particular bridge. Each user participating in a bridge can have their own individual User Profile.

A **Conference Menu** is a named set of options that are provided to a user when they present DTMF keys while connected to the bridge. Each user participating in a bridge can have their own individual Conference Menu.

ConfBridge Configuration

ConfBridge Profiles and Menus are configured in the confbridge.conf configuration file - normally located at /etc/asterisk/confbridge.conf. The file contains three reserved sections:

- [general]
- [default_bridge]
- [default_user]

The **[general]** section is currently unused, but is reserved for future use.

The **[default_bridge]** section contains all options invoked when ConfBridge is instantiated from the dialplan without a bridge profile argument.

The **[default_user]** section contains all options invoked when ConfBridge is instantiated from the dialplan without a user profile argument.

Each section contains a **type** definition. The type definition determines the function of the section. The three **types** are:

- bridge
- user
- menu

bridge is used to denote Bridge Profile section definitions.

user is used to denote User Profile section definitions.

menu is used to denote Conference Menu section definitions.

All other sections, defined by a section identifier encapsulated in square brackets, are user-definable.

Example

This is an example, using invalid options and functions, of a confbridge.conf configuration file, displaying the organizational layout. The various options and functions are described later in this page.

```
[general]
; comments are preceded by a comma
;
; the general section is blank
;
[default_bridge]
type=bridge
; Bridge Profile options go here
myoption=value
myoption2=othervalue
;
[default_user]
type=user
; User Profile options go here
myoption=value
myoption2=othervalue
;
[sample_menu]
type=menu
; Conferece Menu options go here
DTMF=function
otherDTMF=otherFunction
;
```

Bridge Profile Configuration Options

A Bridge Profile provides the following configuration options:

Option	Values	Description
type	bridge	Set this to bridge to configure a bridge profile
max_members	integer; e.g. 50	Limits the number of participants for a single conference to a specific number. By default, conferences have no participant limit. After the limit is reached, the conference will be locked until someone leaves. Admin-level users are exempt from this limit and will still be able to join otherwise-locked, because of limit, conferences.
record_conference	yes/no	Records the conference call starting when the first user enters the room, and ending when the last user exits the room. The default recorded filename is 'confbridge-<name of conference bridge>-<start time>.wav and the default format is 8kHz signed linear. By default, this option is disabled. This file will be located in the configured monitoring directory as set in asterisk.conf
record_file	path, e.g. /tmp/myfiles	When record_conference is set to yes, the specific name of the recorded file can be set using this option. Note that since multiple conferences may use the same Bridge profile, this can cause issues, depending on the configuration. It is recommended to only use this option dynamically with the CONFBRIDGE() dialplan function. This allows the recorded name to be specified and a unique name to be chosen. By default, the recorded file is stored in Asterisk's spool/monitory directory, with a unique filename starting with the 'confbridge' prefix.

internal_sample_rate	auto, 8000, 12000, 16000, 24000, 32000, 44100, 48000, 96000, 192000	Sets the internal native sample rate at which to mix the conference. The "auto" option allows Asterisk to adjust the sample rate to the best quality / performance based on the participant makeup. Numbered values lock the rate to the specified numerical rate. If a defined number does not match an internal sampling rate supported by Asterisk, the nearest sampling rate will be used instead.
mixing_interval	10, 20, 40, 80	Sets, in milliseconds, the internal mixing interval. By default, the mixing interval of a bridge is 20ms. This setting reflects how "tight" or "loose" the mixing will be for the conference. Lower intervals provide a "tighter" sound with less delay in the bridge and consume more system resources. Higher intervals provide a "looser" sound with more delay in the bridge and consume less resources
sound_join	filename	The sound played to the bridge when a user joins, typically some kind of beep sound
sound_leave	filename	The sound played to the bridge when a user leaves, also typically some kind of beep sound
sound_has_joined	filename	The sound played as a user intro, e.g. "xxxx has joined the conference."
sound_has_left	filename	The sound played as a user parts the conference, e.g. "xxxx has left the conference."
sound_kicked	filename	The sound played to a user who has been kicked from the conference.
sound_muted	filename	The sound played to a user when the mute option is toggled on.
sound_unmuted	filename	The sound played to a user when the mute option is toggled off.
sound_only_person	filename	The sound played when a user is the only person in the conference.
sound_only_one	filename	The sound played to a user when there is only one other person in the conference.
sound_there_are	filename	The sound played when announcing how many users there are in a conference.
sound_other_in_party	filename	Used in conjunction with the sound_there_are option, used like "sound_there_are" <number of participants> "sound_other_in_party"
sound_place_into_conference	filename	The sound played when someone is placed into a conference, after waiting for a marked user.
sound_wait_for_leader	filename	The sound played when a user is placed into a conference that cannot start until a marked user enters.
sound_leader_has_left	filename	The sound played when the last marked user leaves the conference.
sound_get_pin	filename	The sound played when prompting for a conference PIN
sound_invalid_pin	filename	The sound played when an invalid PIN is entered too many (3) times
sound_locked	filename	The sound played to a user trying to join a locked conference.
sound_locked_now	filename	The sound played to an Admin-level user after toggling the conference to locked mode.
sound_unlocked_now	filename	The sound played to an Admin-level user after toggling the conference to unlocked mode.
sound_error_menu	filename	The sound played when an invalid menu option is entered.

Example

In this example, a Bridge Profile called "fancybridge" will be created. It will be configured to allow up to 20 callers, and will be set to mix at 10ms (tight mixing) at an automatic sampling rate. Additionally, it will be recorded.

```
[fancybridge]
type=bridge
max_members=20
mixing_interval=10
internal_sample_rate=auto
record_conference=yes
```

User Profile Configuration Options

A User Profile provides the following configuration options:

Option	Values	Description
type	user	Set this to user to configure a user profile
admin	yes/no	Sets if the user is an Admin or not. By default, no.
marked	yes/no	Sets if the user is Marked or not. By default, no.
startmuted	yes/no	sets if the user should start out muted. By default, no.
music_on_hold_when_empty	yes/no	Sets whether music on hold should be played when only one person is in the conference or when the user is waiting on a marked user to enter the conference. By default, off.
quiet	yes/no	When set to "yes," enter/leave prompts and user introductions are not played. By default, no.
announce_user_count	yes/no	Sets if the number of users in the conference should be announced to the caller. By default, no.
announce_user_count_all	yes/no; or an integer	Sets if the number of users should be announced to all other users in the conference when someone joins. When set to a number, the announcement will only occur once the user count is above the specified number
announce_only_user	yes/no	Sets if the only user announcement should be played when someone enters an empty conference. By default, yes.
wait_marked	yes/no	Sets if the user must wait for another marked user to enter before joining the conference. By default, no.
end_marked	yes/no	If enabled, every user with this option in their profile will be removed from the conference when the last marked user exists the conference.
dsp_drop_silence	yes/no	Drops what Asterisk detects as silence from entering into the bridge. Enabling this option will drastically improve performance and help remove the buildup of background noise from the conference. This option is highly recommended for large conferences, due to its performance improvements.
dsp_talking_threshold	integer in milliseconds	The time, in milliseconds, by default 160, of sound above what the DSP has established as base-line silence for a user, before that user is considered to be talking. This value affects several options: 1. Audio is only mixed out of a user's incoming audio stream if talking is detected. If this value is set too loose, the user will hear themselves briefly each time they begin talking until the DSP has time to establish that they are in fact talking. 2. When talker detection AMI events are enabled, this value determines when talking has begun, which causes AMI events to fire. If this value is set too tight, AMI events may be falsely triggered by variants in the background noise of the caller. 3. The drop_silence option depends on this value to determine when the user's audio should be mixed into the bridge after periods of silence. If this value is too loose, the beginning of a user's speech will get cut off as they transition from silence to talking.

dsp_silence_threshold	integer in milliseconds	The time, in milliseconds, by default 2500, of sound falling within what the DSP has established as the baseline silence, before a user is considered to be silent. The best way to approach this option is to set it slightly above the maximum amount of milliseconds of silence a user may generate during natural speech. This value affects several operations: 1. When talker detection AMI events are enabled, this value determines when the user has stopped talking after a period of talking. If this value is set too low, AMI events indicating that the user has stopped talking may get faslely sent out when the user briefly pauses during mid sentence. 2. The drop_silence option depends on this value to determine when the user's audio should begin to be dropped from the bridge, after the user stops talking. If this value is set too low, the user's audio stream may sound choppy to other participants.
talk_detection_events	yes/no	Sets whether or not notifications of when a user begins and ends talking should be sent out as events over AMI. By default, no.
denoise	yes/no	Whether or not a noise reduction filter should be applied to the audio before mixing. By default, off. This requires codec_speex to be built and installed. Do not confuse this option with drop_silence. denoise is useful if there is a lot of background noise for a user, as it attempts to remove the noise while still preserving the speech. This option does not remove silence from being mixed into the conference and does come at the cost of a slight performance hit.
jitterbuffer	yes/no	Whether or not to place a jitter buffer on the caller's audio stream before any audio mixing is performed. This option is highly recommended, but will add a slight delay to the audio and will incur a slight performance penalty. This option makes use of the JITTERBUFFER dialplan function's default adaptive jitter buffer. For a more fine-tuned jitter buffer, disable this option and use the JITTERBUFFER dialplan function on the calling channel, before it enters the ConfBridge application.
pin	integer	Sets if the user must enter a PIN before joining the conference. The user will be prompted for the PIN.
announce_join_leave	yes/no	When enabled, this option prompts the user for their name when entering the conference. After the name is recorded, it will be played as the user enters and exists the conference. By default, no.
dtmf_passthrough	yes/no	Whether or not DTMF received from users should pass through the conference to other users. By default, no.

Example

In this example, we will create a user profile called "fancyuser" that includes music on hold, user count announcements, join/leave announcements, silence detection, noise reduction and requires a PIN of 456.

```
[fancyuser]
type=user
music_on_hold_when_empty=yes
announce_user_count_all=yes
announce_join_leave=yes
dsp_drop_silence=yes
denoise=yes
pin=456
```

Conference Menu Configuration Options

A Conference Menu provides the following configuration options:

Option	Values	Description
type	menu	Set this to menu to configure a conference menu

playback	(<name of audio file1>&<name of audio file2>&...)	Plays back an audio file, or a string of audio files chained together using the & character, to the user and then immediately returns them to the conference.
playback_and_continue	(<name of audio file 1>&<name of audio file 2>&...)	Plays back an audio file, or a series of audio files chained together using the & character, to the user while continuing the collect the DTMF sequence. This is useful when using a menu prompt that describes all of the menu options. Note that any DTMF during this action will terminate the prompt's playback.
toggle_mute		Toggles mute on and off. When a user is muted, they will not be able to speak to other conference users, but they can still listen to other users. While muted, DTMF keys from the caller will continue to be collected.
no_op		This action does nothing. Its only real purpose exists for being able to reserve a sequence in the configuration as a menu exit sequence.
decrease_listening_volume		Decreases the caller's listening volume. Everything they hear will sound quieter.
increase_listening_volume		Increases the caller's listening volume. Everything they hear will sound louder.
reset_listening_volume		Resets the caller's listening volume to the default level.
decrease_talking_volume		Decreases the caller's talking volume. Everything they say will sound quieter to other callers.
increase_talking_volume		Increases the caller's talking volume. Everything they say will sound louder to other callers.
reset_talking_volume		Resets the caller's talking volume to the default level.
dialplan_exec	(context,exten,priority)	Allows one to escape from the conference and execute commands in the dialplan. Once the dialplan exits, the user will be put back into the conference.
leave_conference		Allows a user to exit the conference and continue execution in the dialplan.
admin_kick_last		Allows an Admin to remove the last participant from the conference. This action only works for users whose User Profiles set them as conference Admins.
admin_toggle_conference_lock		Allows an Admin to toggle locking and unlocking the conference. When the conference is locked, only other Admin users can join. When the conference is unlocked, any user may join up to the limit defined by the max_members Bridge Profile option. This action only works for users whose User Profiles set them as conference Admins.

Example

In this example, we'll create a menu called "fancymenu." This menu will utilize many of the options listed above. We will construct a features menu that plays when the user enters the * character. Since we will do this using the playback_and_continue option, we will define other menu items as being a "subset" of the * command, e.g. *4, so that once the user presses *, they can listen to the menu options and then press the specific "after-star" option, e.g. 4, to affect the option. Additionally, we will duplicate those same sub-features as non-* features, so that the user does not need to have entered the * menu structure in order to affect the options, they can just press the key, e.g. "4" at any time, regardless of whether or not they're in the *-tree.

```
[fancymenu]
type=menu
*=playback_and_continue(conf-togglemute&press&digits/1&silence/1&conf-
```

Of particular note in this example, we're calling the dialplan_exec option. Here, we're specifying "addcaller,1,1." This means that when someone dials 3, Asterisk will escape them out of the bridge momentarily to go execute priority 1 of extension 1 in the addcaller context of the dialplan (extensions.conf). Our dialplan, including the addcaller context, in this case, might look like:

```
[addcaller]
exten => 1,1,Originate(SIP/otherpeer,exten,conferences,100,1)

[conferences]
exten => 100,1,ConfBridge(1234)
```

Thus, when someone dials "3" while in the bridge, they'll Originate a call from the dialplan that puts SIP/otherpeer into the conference. Once the dial has completed, the person that dialed "3" will find themselves back in the bridge, with the other participants.

ConfBridge Dialplan Syntax

The syntax for the new ConfBridge application is as follows:

```
ConfBridge([confno][,bridge_profile[,user_profile[,menu]]])
```

The ConfBridge application takes the following arguments

- confno - The conference number
- bridge_profile - The Bridge Brofile name from confbridge.conf. When left blank, a dynamically built Bridge Profile created by the CONFBRIDGE dialplan function is searched for on the channel and, if available, used. If no dynamic profile is found, the "default_bridge" profile found in confbridge.conf is used.

  > ⊖ It is important to note that while User Profiles are unique for each participant, Bridge Profiles are unique to the bridge, **not** the user. So you can only create one Bridge Profile per conference

- user_profile - The User Profile name from confbridge.conf. When left blank, a dynamically built User Profile created by the CONFBRIDGE dialplan function is searched for on the channel and, if available, used. If no dynamic profile is present, the "default_user" profile found in confbridge.conf is used.
- menu - The Conference Menu name from confbridge.conf. No menu is applied by default if this option is left blank.

Example 1
In this example, callers will be joined to conference number 1234, using the default Bridge Profile, the default User Profile, and no Conference Menu.

```
exten => 1,1,Answer()
exten => 1,n,ConfBridge(1234)
```

Example 2
In this example, callers will be joined to conference number 1234, with the default Bridge Profile, a User Profile called "1234_participants" and a Conference Menu called "1234_menu."

```
exten => 1,1,Answer()
exten => 1,n,ConfBridge(1234,,1234_participants,1234_menu)
```

ConfBridge Dialplan Functions

The CONFBRIDGE dialplan function is used to set customized Bridge and/or User Profiles on a channel for the ConfBridge application. It uses the same options defined in confbridge.conf and allows the creation of dynamic, dialplan-driven conferences.

Syntax

```
CONFBRIDGE(type,option)
```

- type - Refers to which type of profile the option belongs to. Type can be either "bridge" or "user."
- option - Refers to the confbridge.conf option that is to be set dynamically on the channel. This can also refer to an existing Bridge or User Profile by using the keyword "template." In this case, an existing Bridge or User Profile can be appended or modified on-the-fly.

Examples

Example 1
In this example, the custom set User Profile on this channel enables announce_join_leave (so users will be announced as they come and go), sets users to join muted (so that they're not able to speak), and pushes them into bridge "1."

```
exten => 1,1,Answer()
exten => 1,n,Set(CONFBRIDGE(user,announce_join_leave)=yes)
exten => 1,n,Set(CONFBRIDGE(user,startmuted)=yes)
exten => 1,n,ConfBridge(1)
```

Example 2
In this example, we will include an existing User Profile, the default_user User Profile as defined in confbridge.comf, and we will set additional parameters (admin and marked) that aren't already defined in the default_user User Profile.

```
exten => 1,1,Answer()
exten => 1,n,Set(CONFBRIDGE(user,template)=default_user)
exten => 1,n,Set(CONFBRIDGE(user,admin)=yes)
exten => 1,n,Set(CONFBRIDGE(user,marked)=yes)
exten => 1,n,ConfBridge(1)
```

The CONFBRIDGE_INFO dialplan function is used to retrieve information about a conference, such as locked/unlocked status and the number of parties including admins and marked users.

Syntax

```
CONFBRIDGE_INFO(type,conf)
```

- type - Refers to which information type to be retrieved. Type can be either "parties," "admins," "marked," or "locked."
- conf - Refers to the name of the conference being referenced.

The CONFBRIDGE_INFO function returns a non-negative integer for valid conference identifiers, 0 or 1 for locked, and "" for invalid conference identifiers.

ConfBridge CLI Options

ConfBridge offers several options that may be invoked from the Asterisk CLI.

confbridge kick <conference> <channel>

Removes the specified channel from the conference, e.g.:

```
*CLI> confbridge kick 1111 SIP/mypeer-00000000
Kicking SIP/mypeer-00000000 from confbridge 1111
```

confbridge list

Shows a summary listing of all bridges, e.g.:

```
*CLI> confbridge list
Conference Bridge Name          Users   Marked Locked?
=============================== ====== ====== ========
1111                                 1      0 unlocked
```

confbridge list <conference>

Shows a detailed listing of participants in a specified conference, e.g.:

```
*CLI> confbridge list 1111
Channel                        User Profile     Bridge Profile    Menu
============================== ================ ================
================
SIP/mypeer-00000001            default_user     1111
sample_user_menu
```

confbridge lock <conference>

Locks a specified conference so that only Admin users can join, e.g.:

```
*CLI> confbridge lock 1111
Conference 1111 is locked.
```

confbridge unlock <conference>

Unlocks a specified conference so that only Admin users can join, e.g.:

```
*CLI> confbridge unlock 1111
Conference 1111 is unlocked.
```

confbridge mute <conference> <channel>

Mutes a specified user in a specified conference, e.g.:

```
*CLI> confbridge mute 1111 SIP/mypeer-00000001
Muting SIP/mypeer-00000001 from confbridge 1111
```

confbridge unmute <conference> <channel>

Unmutes a specified user in a specified conference, e.g.:

```
*CLI> confbridge unmute 1111 SIP/mypeer-00000001
Unmuting SIP/mypeer-00000001 from confbridge 1111
```

confbridge record start <conference> <file>

Begins recording a conference. If "file" is specified, it will be used, otherwise, the Bridge Profile record_file will be used. If the Bridge Profile does not specify a record_file, one will be automatically generated in Asterisk's monitor directory. Usage:

```
*CLI> confbridge record start 1111
Recording started
*CLI>    == Begin MixMonitor Recording
ConfBridgeRecorder/conf-1111-uid-618880445
```

confbridge record stop <confererence>

Stops recording the specified conference, e.g.:

```
*CLI> confbridge record stop 1111
Recording stopped.
*CLI>    == MixMonitor close filestream (mixed)
   == End MixMonitor Recording
ConfBridgeRecorder/conf-1111-uid-618880445
```

confbridge show menus

Shows a listing of Conference Menus as defined in confbridge.conf, e.g.:

```
*CLI> confbridge show menus
--------- Menus -----------
sample_admin_menu
sample_user_menu
```

confbridge show menu <menu name>

Shows a detailed listing of a named Conference Menu, e.g.:

```
*CLI> confbridge show menu sample_admin_menu
Name: sample_admin_menu
*9=increase_talking_volume
*8=no_op
*7=decrease_talking_volume
*6=increase_listening_volume
*4=decrease_listening_volume
*3=admin_kick_last
*2=admin_toggle_conference_lock
*1=toggle_mute
*=playback_and_continue(conf-adminmenu)
```

confbridge show profile bridges

Shows a listing of Bridge Profiles as defined in confbridge.conf, e.g.:

```
*CLI> confbridge show profile bridges
--------- Bridge Profiles -----------
1111
default_bridge
```

confbridge show profile bridge <bridge>

Shows a detailed listing of a named Bridge Profile, e.g.:

```
*CLI> confbridge show profile bridge 1111
---------------------------------------------
Name:                 1111
Internal Sample Rate: 16000
Mixing Interval:      10
Record Conference:    no
Record File:          Auto Generated
Max Members:          No Limit
sound_only_person:    conf-onlyperson
sound_has_joined:     conf-hasjoin
sound_has_left:       conf-hasleft
sound_kicked:         conf-kicked
sound_muted:          conf-muted
sound_unmuted:        conf-unmuted
sound_there_are:      conf-thereare
sound_other_in_party: conf-otherinparty
sound_place_into_conference: conf-placeintoconf
sound_wait_for_leader:       conf-waitforleader
sound_get_pin:        conf-getpin
sound_invalid_pin:    conf-invalidpin
sound_locked:         conf-locked
sound_unlocked_now:   conf-unlockednow
sound_lockednow:      conf-lockednow
sound_error_menu:     conf-errormenu
```

confbridge show profile users

Shows a listing of User Profiles as defined in confbridge.conf, e.g.:

```
*CLI> confbridge show profile users
--------- User Profiles -----------
awesomeusers
default_user
```

confbirdge show profile user <user>

Shows a detailed listing of a named Bridge Profile, e.g.:

```
*CLI> confbridge show profile user default_user
--------------------------------------------
Name:                   default_user
Admin:                  false
Marked User:            false
Start Muted:            false
MOH When Empty:         enabled
MOH Class:              default
Quiet:                  disabled
Wait Marked:            disabled
END Marked:             disabled
Drop_silence:           enabled
Silence Threshold:      2500ms
Talking Threshold:      160ms
Denoise:                disabled
Talk Detect Events:     disabled
DTMF Pass Through:      disabled
PIN:                    None
Announce User Count:    enabled
Announce join/leave:    enabled
Announce User Count all: enabled
```

ConfBridge Asterisk Manager Interface (AMI) Actions

ConfbridgeList

Lists all users in a particular ConfBridge conference. ConfbridgeList will follow as separate events, followed by a final event called ConfbridgeListComplete

Example

```
Action: ConfbridgeList
Conference: 1111

Response: Success
EventList: start
Message: Confbridge user list will follow

Event: ConfbridgeList
Conference: 1111
CallerIDNum: malcolm
CallerIDName: malcolm
Channel: SIP/malcolm-00000000
Admin: No
MarkedUser: No

Event: ConfbridgeListComplete
EventList: Complete
ListItems: 1
```

ConfbridgeListRooms

Lists data about all active conferences. ConfbridgeListRooms will follow as separate events, followed by a final event called ConfbridgeListRoomsComplete.

Example

```
Action: ConfbridgeListRooms

Response: Success
EventList: start
Message: Confbridge conferences will follow

Event: ConfbridgeListRooms
Conference: 1111
Parties: 1
Marked: 0
Locked: No

Event: ConfbridgeListRoomsComplete
EventList: Complete
ListItems: 1
```

ConfbridgeMute

Mutes a specified user in a specified conference.

Example

```
Action: ConfbridgeMute
Conference: 1111
Channel: SIP/mypeer-00000001


Response: Success
Message: User muted
```

ConfbridgeUnmute

Unmutes a specified user in a specified conference.

Example

```
Action: ConfbridgeUnmute
Conference: 1111
Channel: SIP/mypeer-00000001


Response: Success
Message: User unmuted
```

ConfbridgeKick

Removes a specified user from a specified conference.

Example

```
Action: ConfbridgeKick
Conference: 1111
Channel: SIP/mypeer-00000001


Response: Success
Message: User kicked
```

ConfbridgeLock

Locks a specified conference.

Example

```
Action: ConfbridgeLock
Conference: 1111

Response: Success
Message: Conference locked
```

ConfbridgeUnlock

Unlocks a specified conference.

Example

```
Action: ConfbridgeUnlock
Conference: 1111

Response: Success
Message: Conference unlocked
```

ConfbridgeStartRecord

Starts recording a specified conference, with an optional filename. If recording is already in progress, an error will be returned. If RecordFile is not provided, the default record_file as specified in the conferences Bridge Profile will be used. If record_file is not specified, a file will automatically be generated in Asterisk's monitor directory.

Example

```
Action: ConfbridgeStartRecord
Conference: 1111

Response: Success
Message: Conference Recording Started.

Event: VarSet
Privilege: dialplan,all
Channel: ConfBridgeRecorder/conf-1111-uid-1653801660
Variable: MIXMONITOR_FILENAME
Value: /var/spool/asterisk/monitor/confbridge-1111-1303309869.wav
Uniqueid: 1303309869.6
```

ConfbridgeStopRecord

Stops recording a specified conference.

Example

```
Action: ConfbridgeStopRecord
Conference: 1111

Response: Success
Message: Conference Recording Stopped.

Event: Hangup
Privilege: call,all
Channel: ConfBridgeRecorder/conf-1111-uid-1653801660
Uniqueid: 1303309869.6
CallerIDNum: <unknown>
CallerIDName: <unknown>
Cause: 0
Cause-txt: Unknown
```

ConfBridge Asterisk Manager Interface (AMI) Events

ConfbridgeStart

This event is sent when the first user requests a conference and it is instantiated

Example

```
Event: ConfbridgeStart
Privilege: call,all
Conference: 1111
```

ConfbridgeJoin

This event is sent when a user joins a conference - either one already in progress or as the first user to join a newly instantiated bridge.

Example

```
Event: ConfbridgeJoin
Privilege: call,all
Channel: SIP/mypeer-00000001
Uniqueid: 1303309562.3
Conference: 1111
CallerIDnum: 1234
CallerIDname: mypeer
```

ConfbridgeLeave

This event is sent when a user leaves a conference.

Example

```
Event: ConfbridgeLeave
Privilege: call,all
Channel: SIP/mypeer-00000001
Uniqueid: 1303308745.0
Conference: 1111
CallerIDnum: 1234
CallerIDname: mypeer
```

ConfbridgeEnd

This event is sent when the last user leaves a conference and it is torn down.

Example

```
Event: ConfbridgeEnd
Privilege: call,all
Conference: 1111
```

ConfBridgeTalking

This event is sent when the conference detects that a user has either begin or stopped talking.

Start talking Example

```
Event: ConfbridgeTalking
Privilege: call, all
Channel: SIP/mypeer-00000001
Uniqueid: 1303308745.0
Conference: 1111
TalkingStatus: on
```

Stop talking Example

```
Event: ConfbridgeTalking
Privilege: call, all
Channel: SIP/mypeer-00000001
Uniqueid: 1303308745.0
Conference: 1111
TalkingStatus: off
```

Usage Notes, FAQ and Other

There are many points to consider when using the new ConfBridge appliation. Some will be examined here.

Mixing Interval

The mixing interval for a conference is defined in its Bridge Profile. The allowable options are 10, 20, 40, and 80, all in milliseconds. Usage of 80ms mixing intervals is only supported for conferences that are sampled at 8, 12, 16, 24, 32, and 48kHz. Usage of 40ms intervals includes all of the aforementioned sampling rates as well as 96kHz. 192kHz sampled conferences are only supported at 10 and 20ms mixing intervals. These limitations are imposed because higher mixing intervals at the higher sampling rates causes large increases in memory consumption. Adventurous users may, through changing of the MAX_DATALEN define in bridge_softmix.c allow 96kHz and 192kHz sampled conferences to operate at longer intervals - set to 16192 for 96kHz at 80ms or 32384 for 192kHz at 80ms, recompile, and restart.

Maximizing Performance

In order to maximize the performance of a given machine for ConfBridge purposes, there are several steps one should take.

- Enable dsp_drop_silence is enabled in the User Profile.
  - This is the **single most** important step one can take when trying to increase the number of bridge participants that a single machine can handle. Enabling this means that the audio of users that aren't speaking isn't mixed in with the bridge.
- Lengthen mixing_interval in the Bridge Profile.
  - The default interval is 20ms. Other options are 10, 40, and 80ms. Lower values provide a "tighter" sound, but require substantially more CPU. Higher values provider a "looser" sound, and consume substantially less CPU. Setting the value to 80 provides the highest number of possible participants.
- Connect clients at the same sampling rate.
  - Requiring the bridge to resample between clients that use codecs with different sampling rates is an expensive operation. If all clients are dialed in to the bridge at the same sampling rate, and the bridge operates at that same rate, e.g. 16kHz, then the number of possible clients will be maximized.
- Run Asterisk with a higher priority.
  - By default, Asterisk operates at a relatively normal priority, as compared to other processes on the system. To maximize the number of possible clients, Asterisk should be started using the **-p** (realtime) flag. If the load becomes too large, this can negatively impact the performance of other processes, including the console itself - making it difficult to remotely administer a fully loaded system.

As the number of clients approaches the maximum possible on the given machine, given its processing capabilities, audio quality will suffer. Following the above guidelines will increase the number of connected clients before audio quality suffers.

Sound Prompts

The following Conference Menu and Bridge Profile options should have sound files associated with them. At the present time, prior to the actual release of Asterisk 1.10, many of these sound files do not yet exist; though some already do. Comments or suggestions on phrasings are welcome.

- decrease_listening_volume - "…to decrease the audio volume from other participants" and "To decrease the audio volume from other participants…"
- increase_listening_volume - "…to increase the audio volume from other participants" and "To increase the audio volume from other participants…"
- reset_listening_volume - "…to reset the audio volume of the conference to the default level" and "To reset the audio volume of the conference to the default level…"
- decrease_talking_volume - "…to decrease your speaking volume to other participants" and "To decrease your speaking volume to other participants…"
- increase_talking_volume - "…to increase your speaking volume to other participants" and "To increase your speaking volume to other participants…"
- reset_talking_volume - "…to reset your speaking speaking volume to the default level" and "To reset your speaking volume to the default

level…"
- admin_kick_last - "…to remove the participant who most recently joined the conference" and "To remove the participant who most recently joined the conference…"
- sound_muted - "You are now muted"
- sound_wait_for_leader - "The conference will begin when the leader arrives." Also add "The conference will begin when our **glorious leader** arrives"
- sound_locked_now - "The conference is now locked."
- toggle_mute - "...to mute or unmute yourself" and "To mute or unmute yourself..." and an extended prompt that says "...to mute or unmute yourself. When you are muted, you cannot send audio to other participants; however, you will still hear audio from other unmuted participants."
- leave_conference - "...to leave the conference" and "To leave the conference..."
- admin_toggle_conference_lock "...to lock or unlock the conference" and "To lock or unlock the conference..." and an extended prompt that says "...to lock or unlock the conference. When a conference is locked, only conference administrators can join."
- sound_has_joined - "...has joined the conference."
- sound_has_left - "...has left the conference."
- sound_kicked - "You have been removed from the conference."
- sound_unmuted - "You are no longer muted."
- sound_only_person - "You are currently the only participant in the conference."
- sound_there_are - "There are currently..."
- sound_other_in_party - "...participants in the conference."
- sound_place_into_conference - "There conference will now begin."
- sound_leader_has_left - "The conference has ended."
- sound_get_pin - "Please enter your personal identification number, followed by the pound or hash key."
- sound_invalid_pin - "You have entered too many invalid personal identification numbers."
- sound_locked - "The conference is currently locked and cannot be joined."
- sound_unlocked_now - "The conference is now unlocked."
- sound_error_menu - "You have entered an invalid option."
- exit_menu - "...to exit the menu." and "To exit the menu..."

Device State information for the CCSS generic agent.

There are currently not many devices that have native support for CCSS. Even as the devices become available there may be other reasons why one may choose to not take advantage of the native abilities and stick with the generic implementation. The generic implementation is quite capable and could be greatly enhanced by adding device state capabilities. A phone could then subscribe to the device state with a BLF key in conjunction with Asterisk hints.

The advantages of the device state information would allow a single button to: request CCSS, cancel a CCSS request, and display the current state of a CCSS request.

For example, you may have a single button that when not lit, there is no active CCSS request. When you press that button, the dialplan can query the `DEVICE_STATE()` associated with that caller to determine whether they should be calling `CallCompletionRequest()` or `CallCompletionCancel()`. If there is currently a pending request, then the dialplan would cancel it. This also has the advantage of showing the true state of a request, which is an asynchronous call, even when `CallCompletionRequest()` thinks it was successful. The actual request could ultimately fail. Once lit, further feedback can be provided to the caller about the current state of their request since it will be updated by the CCSS State Machine as appropriate.

The `DEVICE_STATE` mapping is configurable since the BLF being used on a given phone type may vary. The idea is to allow some level of customization as to the phone's behavior.

As an example, you may want the BLF key to go solid once you have requested a callback. You may then want the LED to blink (typically ringing) when either the callback is in process, which is a visual indication that the incoming call is the desired callback. You may want it to blink when the callee is ready but you are busy, giving you a visual indication that the target is available as you may want to get off the line so that the callback can be successful.

Device state information is sent back via the `ast_devstate_prov_add()` callback for any generic CCSS device as it traverses through the state machine. You simply provide a map between `CC_STATE` values and the corresponding `AST_DEVICE` state values.

You could then generate hints against these states similar to what is possible today with Custom Devstates or MeetMe states. For example, you may have an extension 3000 that is currently associated with device `SIP/3000`. You could then create a feature code for that extension that may look something like:

```
exten => *823000,hint,ccss:sip/3000
```

You would then subscribe a BLF button to `*823000` which would point to the dialplan that handled CCSS requests/cancels using the available `DEVICE_STATE()` information about `ccss:sip/3000` to make the decision about what to do.

Asterisk Packages

1. Introduction

Installing a binary distribution of Asterisk makes it easier to maintain your system. Using the package management tools that are included with your Linux distribution, you can install and update software without manually managing dependencies (libraries and utilities on which applications rely).

Currently Asterisk is available for automated binary installation using APT for Debian and Ubuntu or YUM utility on CentOS 5 Linux and RedHat Enterprise Linux 5.

2. APT (Debian / Ubuntu)

The goal with the deb repositories is simply to provide the latest Asterisk versions, such as Asterisk 1.8, for older versions of distributions where distribution policy prohibits incorporating our latest version. As of now, we have Asterisk packages available for the following distribution versions:

- Debian 6.0 (squeeze)
- Ubuntu 10.04 (lucid)
- Ubuntu 10.10 (maverick)
- Ubuntu 11.04 (natty)

2.1. Prerequisites

2.1.1. Dependencies

```
$ sudo apt-get install python-software-properties
```

2.1.2. Importing the public key

```
$ sudo apt-key adv --keyserver pgp.mit.edu --recv-keys 175E41DF
```

2.2. Adding the release branch

2.2.1. Adding the repository

```
$ sudo add-apt-repository "deb http://packages.asterisk.org/deb
`lsb_release -cs` main"
$ sudo add-apt-repository "deb-src http://packages.asterisk.org/deb
`lsb_release -cs` main"
```

2.3. Adding the proposed branch (Optional)

The proposed branch includes updates which are waiting to be moved into the release branch after some testing. They may never reach recommended or they may be replaced with a more recent update. Enabling this is reasonable if you want to participate in testing minor updates, or know that your specific problem has been solved here but the package hasn't reached recommended yet.

2.3.1. Adding the repository

```
$ sudo add-apt-repository "deb http://packages.asterisk.org/deb
`lsb_release -cs`-proposed main"
$ sudo add-apt-repository "deb-src http://packages.asterisk.org/deb
`lsb_release -cs`-proposed main"
```

2.4. Installing Asterisk

2.4.1. Update APT

```
$ sudo apt-get update
```

2.4.2. Asterisk

```
$ sudo apt-get install asterisk-1.8
```

2.4.3. DAHDI

```
$ sudo apt-get install asterisk-dahdi
```

3. YUM (CentOS / Red Hat)

3.1. Prerequisites

3.2. Adding the repository

```
$ vi /etc/yum.repos.d/centos-asterisk.repo
```

```
[asterisk-current]
name=CentOS-$releasever - Asterisk - Current
baseurl=http://packages.asterisk.org/centos/$releasever/current/$basea
```

```
$ vi /etc/yum.repos.d/centos-digium.repo
```

```
[digium-current]
name=CentOS-$releasever - Digium - Current
baseurl=http://packages.digium.com/centos/$releasever/current/$basearc
```

3.3. Installing Asterisk

3.3.1. Asterisk

```
# yum install asterisk18 asterisk18-configs
```

3.3.2. DAHDI

```
# yum install dahdi-linux dahdi-tools libpri
```

Prerequisites

All of Ubuntu's Code, Translations, Packages, Bugs, access control lists, team information, etc.
live in Launchpad. So for you to be able to contribute to bug discussions, upload packages,
contribute code and translations, it's important that you:

- Create an account on launchpad.

- Create a GPG key and import it.
- Create a SSH key and import it.

Creating a Build Environment

Install Ubuntu 10.04 (Lucid)

Installing Ubuntu 10.04 (Lucid)

Enable Backports

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository "deb http://ca.archive.ubuntu.com/ubuntu/
$(lsb_release --short --codename)-backports main restricted universe
multiverse"
```

Upgrade Lucid to the latest release:

```
$ sudo apt-get update
$ sudo apt-get dist-upgrade
$ sudo apt-get autoremove
$ sudo reboot
```

Install required software

```
$ sudo apt-get install build-essential devscripts ubuntu-dev-tools
debhelper dh-make pbuilder svn-buildpackage debian-archive-keyring
ccache
```

pbuilder

```
$ sudo mkdir -p /var/cache/pbuilder/ccache
```

```
$ sudo vi /etc/pbuilder/pbuilderrc
```

/etc/pbuilder/pbuilderrc

```
export CCACHE_DIR=/var/cache/pbuilder/ccache
export PATH="/usr/lib/ccache:${PATH}"
EXTRAPACKAGES=ccache
BINDMOUNTS=${CCACHE_DIR}

# Binaries only.
DEBBUILDOPTS="-b"

# Codenames for Debian suites according to their alias. Update these
when
# needed.
UNSTABLE_CODENAME="sid"
TESTING_CODENAME="wheezy"
STABLE_CODENAME="squeeze"
STABLE_BACKPORTS_SUITE="$STABLE_CODENAME-backports"

# List of Debian suites.
DEBIAN_SUITES=($UNSTABLE_CODENAME $TESTING_CODENAME $STABLE_CODENAME
    "unstable" "testing" "stable")

# List of Ubuntu suites. Update these when needed.
UBUNTU_SUITES=("natty" "maverick" "lucid" "karmic" "jaunty" "hardy")

# Mirrors to use. Update these to your preferred mirror.
DEBIAN_MIRROR="ftp.us.debian.org"
UBUNTU_MIRROR="mirrors.kernel.org"

# Optionally use the changelog of a package to determine the suite
to use if
# none set.
if [ -z "${DIST}" ] && [ -r "debian/changelog" ]; then
    DIST=$(dpkg-parsechangelog | awk '/^Distribution: / {print $2}')
    # Use the unstable suite for certain suite values.
    if $(echo "experimental UNRELEASED" | grep -q $DIST); then
        DIST="$UNSTABLE_CODENAME"
    fi
fi

# Optionally set a default distribution if none is used. Note that
you can set
# your own default (i.e. ${DIST:="unstable"}).
: ${DIST:="$(lsb_release --short --codename)"}

# Optionally change Debian release states in $DIST to their names.
case "$DIST" in
    unstable)
```

```bash
                DIST="$UNSTABLE_CODENAME"
                ;;
        testing)
                DIST="$TESTING_CODENAME"
                ;;
        stable)
                DIST="$STABLE_CODENAME"
                ;;
esac

# Optionally set the architecture to the host architecture if none
set. Note
# that you can set your own default (i.e. ${ARCH:="i386"}).
: ${ARCH:="$(dpkg --print-architecture)"}

NAME="$DIST"
if [ -n "${ARCH}" ]; then
    NAME="$NAME-$ARCH"
    DEBOOTSTRAPOPTS=("--arch" "$ARCH" "${DEBOOTSTRAPOPTS[@]}")
fi
BASETGZ="/var/cache/pbuilder/$NAME-base.tgz"
# Optionally, set BASEPATH (and not BASETGZ) if using cowbuilder
# BASEPATH="/var/cache/pbuilder/$NAME/base.cow/"
DISTRIBUTION="$DIST"
BUILDRESULT="/var/cache/pbuilder/$NAME/result/"
APTCACHE="/var/cache/pbuilder/$NAME/aptcache/"
BUILDPLACE="/var/cache/pbuilder/build/"

if $(echo ${DEBIAN_SUITES[@]} | grep -q $DIST); then
    # Debian configuration
    MIRRORSITE="http://$DEBIAN_MIRROR/debian/"
    COMPONENTS="main contrib non-free"
    DEBOOTSTRAPOPTS=("${DEBOOTSTRAPOPTS[@]}"
"--keyring=/usr/share/keyrings/debian-archive-keyring.gpg")
    # This is for enabling backports for the Debian stable suite.
    if $(echo "$STABLE_CODENAME stable" | grep -q $DIST); then
        EXTRAPACKAGES="$EXTRAPACKAGES debian-backports-keyring"
        OTHERMIRROR="$OTHERMIRROR | deb
http://www.backports.org/debian $STABLE_BACKPORTS_SUITE $COMPONENTS"
    fi
elif $(echo ${UBUNTU_SUITES[@]} | grep -q $DIST); then
    # Ubuntu configuration
    MIRRORSITE="http://$UBUNTU_MIRROR/ubuntu/"
    COMPONENTS="main restricted universe multiverse"
    DEBOOTSTRAPOPTS=("${DEBOOTSTRAPOPTS[@]}"
"--keyring=/usr/share/keyrings/ubuntu-archive-keyring.gpg")
```

```
    OTHERMIRROR="$OTHERMIRROR|deb ${MIRRORSITE}
${DISTRIBUTION}-updates ${COMPONENTS}|deb ${MIRRORSITE}
${DISTRIBUTION}-security ${COMPONENTS}"
else
```

```
        echo "Unknown distribution: $DIST"
        exit 1
fi
```

Debian

```
$ for x in unstable testing stable; do for y in i386 amd64; do sudo
DIST=${x} ARCH=${y} pbuilder create; done; done
```

Ubuntu

```
$ for x in lucid maverick natty; do for y in i386 amd64; do sudo
DIST=${x} ARCH=${y} pbuilder create; done; done
```

svn-buildpackage

```
$ vi ~/.svn-buildpackage.conf
```

```
svn-builder=debuild
svn-noautodch
```

quilt

```
$ vi ~/.quiltrc
```

```
QUILT_PATCHES="debian/patches"

QUILT_PATCH_OPTS="--unified-reject-files"
QUILT_REFRESH_ARGS="-p ab --no-timestamps --no-index"
QUILT_DIFF_OPTS="--show-c-function"
QUILT_DIFF_ARGS="-p ab --no-timestamps --no-index --color=auto"
```

devscripts

```
$ vi ~/.devscripts
```

```
DEBCHANGE_RELEASE_HEURISTIC=changelog
DEBCHANGE_MULTIMAINT_MERGE=yes
DEBCHANGE_MAINTTRAILER=no
DEBUILD_ROOTCMD=fakeroot
DEBUILD_LINTIAN=yes
DEBUILD_LINDA=yes
DEFAULT_DEBRELEASE_DEBS_DIR=../build-area/
USCAN_DESTDIR=../tarballs
```

Create a GPG Key

https://help.ubuntu.com/community/GnuPrivacyGuardHowto

```
$ vi ~/.bashrc
```

```
export DEBFULLNAME='Paul Belanger'
export DEBEMAIL='pabelanger@digium.com'
export GPGKEY=8C3B0FA6
export EDITOR=vi
```

See also

* Ubuntu Packaging Guide

Updating an Ubuntu Package

New upstream release

Checkout source

```
$ mkdir -p ~/digium
$ cd ~/digium
$ svn http://blah.org/svn/blah
```

Upstream tarball

```
$ uscan --verbose
```

Update the changelog file

```
$ dch -e
```

Update patches

```
$ while quilt push; do quilt refresh; done
```

Release package

```
$ dch -r
```

Build package source

```
$ svn-buildpackage -S
```

Compile package

```
$ dput ppa:pabelanger/testing ../build-area/*.changes
```

rebuildd

Introduction

Prerequisites

Creating a Build Environment

Getting started

```
sudo apt-get install rebuildd reprepro apache2
```

reprepro

```
$ sudo adduser --system --shell /bin/bash --gecos 'Reprepro Daemon'
--group --disabled-password reprepro
```

```
$ sudo su reprepro
```

```
$ cd ~
$ mkdir bin conf incoming
```

```
$ vi ~/conf/distributions
```

distributions

```
Suite: lucid-proposed
Version: 10.04
Codename: lucid-proposed
Architectures: i386 amd64 source
Components: main
SignWith: yes
Log: logfile
  --changes ~/bin/build_sources
```

```
$ vi ~/conf/incoming
```

incoming

```
Name: incoming
IncomingDir: incoming
Allow: lucid-proposed
Cleanup: on_deny on_error
TempDir: tmp
```

```
$ vi ~/conf/apache.conf
```

apache.conf

```
Alias /debian /home/reprepro/
<Directory /home/reprepro>
        Options +Indexes
        AllowOverride None
        order allow,deny
        allow from all
</Directory>
```

```
$ vi ~/bin/build_sources
```

```bash
#!/bin/bash

action=$1
release=$2
package=$3
version=$4
changes_file=$5

# Only care about packages being added
if [ "$action" != "accepted" ]
then
 exit 0
fi

# Only care about source packages
echo $changes_file | grep -q _source.changes
if [ $? = 1 ]
then
 exit 0
fi

# Kick off the job
echo "$package $version 1 $release"  | sudo rebuildd-job add
```

```
$ reprepro -V -b . createsymlinks
$ reprepro -V -b . processincoming incoming
```

```
$ exit
```

rebuildd

```
$ sudo vi /etc/default/rebuildd
```

```
START_REBUILDD=1
START_REBUILDD_HTTPD=1
DISTS="lucid"
```

Also see

- http://alioth.debian.org/scm/viewvc.php/**checkout**/mirrorer/docs/manual.html?revision=HEAD&root=mirrorer
- http://inodes.org/2009/09/14/building-a-private-ppa-on-ubuntu/

Working with Source Packages

```
$ sudo apt-get build-dep asterisk-1.8
```

```
$ DEB_BUILD_OPTIONS="debug" apt-get -b source asterisk-1.8
```