APPENDIX A

VoIP Channels

VoIP channels in Asterisk represent connections to the protocols they support. Each protocol you wish to use requires a configuration file, containing general parameters defining how your system handles the protocol as well as specific parameters for each channel (or device) you will want to reference in your dialplan. In this appendix, we'll take an in-depth look at the IAX and SIP configuration files.

IAX

The IAX configuration file (*iax.conf*) contains all of the configuration information Asterisk needs to create and manage IAX protocol channels. The sections in the file are separated by headings, which are formed by a word framed in square brackets (`[]`). The name in the brackets will be the name of the channel, with one notable exception: the `[general]` section, which is not a channel, is the area where global protocol parameters are defined.

This section examines the various general and channel-specific settings for *iax.conf*. We will define each parameter, and then give an example of its use. Certain options may have several valid arguments. These arguments are listed beside the option, separated with the pipe symbol (`|`). For example, `bandwidth=low|medium|high` means that the `bandwidth` option accepts one of the values `low`, `medium`, or `high` as its argument.

You can insert comments anywhere in the *iax.conf* file, by preceding the comment text with the semicolon character (`;`). Everything to the right of the semicolon will be ignored. Feel free to use comments liberally.

General IAX Settings

The first non-comment line in your *iax.conf* file must be the heading `[general]`. The parameters in this section will apply to all connections using this protocol, unless defined differently in a specific channel's definition. Since some of these settings can

be defined on a per-channel basis, we have identified settings that are always global with the tag "(global)" and those that can optionally be configured for individual channels with the tag "(channel)." If you define a channel parameter under the `[general]` section, you do not need to define it in each channel; its value becomes the default. Keep in mind that setting a parameter in the `[general]` section does not prevent you from setting it differently for specific channels; it merely makes this setting the default. Also keep in mind that not defining these parameters may, in some cases, cause a system default to be used instead.

Here are the parameters that you can configure:

`accountcode` *(channel)*
: The account code can be defined on a per-user basis. If defined, this account code will be assigned to a call record whenever no specific user account code is set. The `accountcode` name configured will be used as the *filename.csv* in the */var/log/asterisk/cdr-csv/* directory to store Call Detail Records (CDRs) for the user/peer/friend.

```
accountcode=iax-username
```

`allow` *and* `disallow` *(channel)*
: Specific codecs can be allowed or disallowed, limiting codec use to those preferred by the system designer. `allow` and `disallow` can also be defined on a per-channel basis. Keep in mind that `allow` statements in the `[general]` section will carry over to each of the channels, unless you reset with a `disallow=all`. Codec negotiation is attempted in the order in which the codecs are defined. Best practice suggests that you define `disallow=all`, followed by explicit `allow` statements for each codec you wish to use. If nothing is defined, `allow=all` is assumed.

```
disallow=all
allow=ulaw
allow=gsm
allow=ilbc
```

`amaflags` *(channel)*
: Automatic Message Accounting (AMA) is defined in the Telcordia Family of Documents listed under FR-AMA-1. These documents specify standard mechanisms for generation and transmission of CDRs. You can specify one of four AMA flags to apply to all IAX connections.

```
amaflags=default|omit|billing|documentation
```

`authdebug` *(global)*
: You can minimize the amount of authorization debugging by disabling it with `authdebug=no`. Authorization debugging is enabled by default if not explicitly disabled.

```
authdebug=no
```

`autokill` *(global)*
: To minimize the danger of stalling when a host is unreachable, you can set `autokill` to `yes` to specify that any new connection should be torn down if an `ACK`

is not received within 2,000 ms. (This is obviously not advised for hosts with high latency.) Alternatively, you can replace `yes` with the number of milliseconds to wait before considering a peer unreachable. `autokill` configures the wait for all IAX2 peers, but you can configure it differently for individual peers with the use of the `qualify` command.

```
autokill=1500
```

`bandwidth` *(channel)*

`bandwidth` is a shortcut that may help you get around using `disallow=all` and multiple `allow` statements to specify which codecs to use. The valid options are:

`high`
: Allows all codecs (G.723.1, GSM, ulaw, alaw, G.726, ADPCM, slinear, LPC10, G.729, Speex, iLBC).

`medium`
: Allows all codecs except slinear, ulaw, and alaw.

`low`
: Allows all medium codecs except G.726 and ADPCM.

```
bandwidth=low|medium|high
```

`bindport` *and* `bindaddr` *(global)*

These optional parameters allow you to control the IP interface and port on which you wish to accept IAX connections. If omitted, the port will be set to 4569, and all IP addresses in your Asterisk system will accept incoming IAX connections. If multiple bind addresses are configured, only the defined interfaces will accept IAX connections. The address 0.0.0.0 tells Asterisk to listen on all interfaces.

```
bindport=4569
bindaddr=192.168.0.1
```

`codecpriority` *(channel)*

The `codecpriority` option controls which end of an inbound call leg will have priority over the negotiation of codecs. If set in the `[general]` section, the selected options will be inherited by all user entries in the channel configuration file; however, they can be defined in the individual user entries for more granular control. If set in both the `[general]` and user sections, the user entry will override that which is configured in the `[general]` section. If this parameter is not configured, the value defaults to `host`.

Valid options include:

`caller`
: The inbound caller has priority over the host.

`host`
: The host has priority over the inbound caller.

`disabled`
: Codec preferences are not considered—this is the default behavior before the implementation of codec preferences.

`reqonly`
: Codec preferences are ignored, and the call is accepted only if the requested codec is available.

```
codecpriority=caller|host|disabled|reqonly
```

`delayreject` *(global)*
: If an incorrect password is received on an IAX channel, this will delay the sending of the `REGREQ` or `AUTHREP` reject messages, which will help to secure against brute-force password attacks. The delay time is 1,000 ms.

```
delayreject=yes|no
```

`forcejitterbuffer` *(channel)*
: Since Asterisk attempts to bridge channels (endpoints) directly together, the endpoints are normally allowed to perform jitter buffering themselves. However, if the endpoints have a poor jitter buffer implementation, you may wish to force Asterisk to perform jitter buffering no matter what. You can force jitter buffering to be performed with `forcejitterbuffer=yes`.

```
forcejitterbuffer=yes
```

`jitterbuffer` *(channel)*
: *Jitter* refers to the varying latency between packets. When packets are sent from an end device, they are sent at a constant rate with very little latency variation. However, as the packets traverse the Internet, the latency between the packets may become varied; thus, they may arrive at the destination at different times, and possibly even out of order.

  The jitter buffer is, in a sense, a staging area where the packets can be reordered and delivered in a regulated stream. Without a jitter buffer, the user may perceive anomalies in the stream, experienced as static, strange sound effects, garbled words, or, in severe cases, missed words or syllables.

  The jitter buffer affects only data received from the far end. Any data you transmit will not be affected by your jitter buffer, as the far end will be responsible for the de-jittering of its incoming connections.

  The jitter buffer is enabled with the use of `jitterbuffer=yes`.

```
jitterbuffer=yes|no
```

`language` *(channel)*
: This sets the language flag to whatever you define. The global default language is English. The language that is set is sent by the channel as an information element. It is also used by applications such as `SayNumber( )` that have different files for different languages. Keep in mind that languages other than English are not explicitly installed on the system, and it is up to you to configure the system to ensure that the language you specify is handled properly.

```
language=en
```

`mailboxdetail` *(global)*

If `mailboxdetail` is set to `yes`, the new/old message count is sent to the user, instead of a simple statement of whether new and old messages exist. `mailboxdetail` can also be set on a per-peer basis.

```
mailboxdetail=yes
```

`maxjitterbuffer` *(channel)*

This parameter is used to set the maximum size of the jitter buffer, in milliseconds. Be sure not to set `maxjitterbuffer` too high, or you will needlessly increase your latency.

```
maxjitterbuffer=500
```

`regcontext` *(channel)*

By specifying the context that contains the actions to perform, you can configure Asterisk to perform a number of actions when a peer registers to your server. This option works in conjunction with `regexten`, by specifying the extension to execute. If no `regexten` is configured, the peer name is used as the extension. Asterisk will dynamically create and destroy a `NoOp` at priority 1 for the extension. All actions to be performed upon registration should start at priority 2. More than one `regexten` may be supplied, if separated by an `&`. `regcontext` can be set on a per-peer basis or globally.

```
regcontext=registered-phones
```

`regexten` *(channel)*

The `regexten` option is used in conjunction with `regcontext` to specify the extension to be executed within the configured context. If `regexten` is not explicitly configured, the peer name is used as the extension to match.

```
regexten=myphone
```

`resyncthreshold` *(channel)*

The resynchronize threshold is used to resynchronize the jitter buffer if a significant change is detected over a few frames, assuming that the change was caused by a timestamp mixup. The resynchronization threshold is defined as the measured jitter plus the `resyncthreshold` value, defined in milliseconds.

```
resyncthreshold=1000
```

`tos` *(global)*

Asterisk can set the Type of Service (TOS) bits in the IP header to help improve performance on routers that respect TOS bits in their routing calculations. The following values are valid:

`lowdelay`
: Minimize delay.

`throughput`
: Maximize throughput.

`reliability`
: Maximize reliability.

`mincost`
: Minimize cost.

`none`
: No bits set.

```
tos=lowdelay|throughput|reliability|mincost|none
```

`trunk` *(channel)*
: IAX2 trunking enables Asterisk to send media (as mini-frames) from multiple channels using a single header. The reduction in overhead makes the IAX2 protocol more efficient when sending multiple streams to the same endpoint (usually another Asterisk server).

```
trunk=yes|no
```

`trunkfreq` *(channel)*
: `trunkfreq` is used to control how frequently you send trunk messages, in milliseconds. Trunk messages are sent in conjunction with the `trunk=yes` command.

```
trunkfreq=20
```

Retrieving Dialplan Information from a Remote Asterisk Box

Asterisk can retrieve dialplan information from another Asterisk box with the use of a `switch =>` statement. When this occurs, the Asterisk IAX channel driver must wait for a reply from the remote box before it can continue with other IAX-related processes. This is especially troubling when you have multiple `switch` statements nested throughout multiple boxes—if a `switch` statement has to traverse several boxes, there could be an appreciable delay before a result is returned.

When the global `iaxcompat` option is set to `yes`, Asterisk will spawn a separate thread when the `switch` lookup is being performed. The use of this thread allows the main IAX channel driver to continue on with other processes while the thread waits for the reply. A small performance hit is incurred with this option.

```
iaxcompat=yes|no
```

register Statements

The register switch (`register =>`) is used to register your Asterisk box to a remote server—this lets the remote end know where you are, in case you are configured with a dynamic IP address. Note that `register` statements are used only when the remote end has you configured as a *peer*, and when `host=dynamic`.

The basic format for a `register` statement is:

```
register => username:password@remote-host
```

The *password* is optional (if not configured on the remote system).

Alternatively, you can specify an RSA key by framing the appropriate RSA key name* in square brackets ([]):

```
register => username:[rsa-key-name]@remote-host
```

By default, `register` requests will be sent via port 4569. You can direct them to a different port by explicitly specifying it, as follows:

```
register => username:password@remote-host:1234
```

IAX Channel Definitions

With the general settings defined, we can now define our channels. Defining a guest channel is recommended whenever you want to accept anonymous IAX calls. This is a very common way for folks in the Asterisk community to contact one another. Before you decide that this is not for you, keep in mind that anyone whom you want to be able to connect to you via IAX (without you specifically configuring an account for them) will need to connect as a guest. This account, in effect, becomes your "IAX phone number." Your guest channel definition should look something like this:

```
[guest]
type=user
context=incoming
callerid="Incoming IAX Guest"
```

No doubt the spammers will find a way to harass these addresses, but in the short term this has not proven to be a problem. In the long term, we'll probably use DUNDi.†

If you wish to accept calls from the Free World Dialup network, Asterisk comes with a predefined security key that ensures that anonymous connections cannot spoof an incoming Free World Dialup call. You'll want to set up an `iaxfwd` channel:

```
[iaxfwd]
type=user
context=incoming
auth=rsa
inkeys=freeworlddialup
```

If you have resources advertised on a DUNDi network, the associated user must be defined in *iax.conf*:

```
[dundi]
type=user
dbsecret=dundi/secret
context=dundi-incoming
```

* Asterisk RSA keys are typically located in */var/lib/asterisk/keys/*. You can generate your own keys using the *astkeygen* script.

† See Chapter 9 for more information regarding DUNDi.

IAX Authentication

IAX provides authentication mechanisms to allow for a reasonable level of security between endpoints. This does not mean that the audio information cannot be captured and decoded, but it does mean that you can more carefully control who is allowed to make connections to your system. Three levels of security are supported on IAX channels. The `auth` option defines which authentication method to use on the channel: `plaintext`, `md5`, or `rsa`.

`plaintext`, in IAX, offers very little security. While it will prevent connection to the channel unless a valid password is supplied, the fact that the password is stored in *iax.conf* in plain text and is transmitted and received as plain text makes this a very insecure authentication method.

`md5` improves the security on the network connection; however, both ends still require a plain-text `secret` in the *iax.conf* file. Here's how it works: Box A requests a connection with Box B, which in turn replies with an authorization request including a randomly generated number. Box A then generates an MD5 hash using the value supplied in the `secret` field of *iax.conf* and the random number from Box B. The hash is returned in the authorization reply, and Box B compares it to the hash it generated locally. If the hashes match, authorization is granted.

`rsa` provides the most security. Before using RSA, each end must create a public and private key pair through the *astgenkey* script, typically located in */usr/src/asterisk/contrib/scripts/*. The public key must then be given to the far end. Each end of the circuit must include the public key of the far end in its channel definition, using the `inkeys` and `outkey` parameters.

RSA keys are stored in */var/lib/asterisk/keys/*. Public keys are named *name*`.pub`; private keys are named *name*`.key`. Private keys must be encrypted with 3DES.

If you have IAX-based devices (such as an IAXy), or IAX-based users at a remote node, you may want to provide them with their own channels with which to connect to the system.

Let's say you have a user on a remote node for whom you want to define an IAX channel. We'll call this hypothetical channel `sushi`. The channel definition might look something like this:

```
[sushi]
type=user
context=local_users
auth=md5,plaintext,rsa
secret=wasabi
notransfer=yes
jitterbuffer=yes
callerid="Happy Tempura" <(800) 555-1234>
accountcode=seaweed
deny=0.0.0.0/0.0.0.0
```

```
permit=192.168.1.100/255.255.255.0
language=en
```

Incoming calls for this channel will arrive in the context `local_users` and will ask the system to accept the Caller ID `Happy Tempura <(800) 555-1234>`. The system will be willing to accept MD5, plain-text, or RSA authentication from this user, so long as the password `wasabi` is provided and the call comes from the IP address `192.168.1.100`. All calls related to this channel will be assigned the account code `seaweed`. Because we've defined `notransfer`, the media path for this channel will always pass through Asterisk; it cannot be redirected to another IAX node.

If you yourself are a remote node, and you need to connect into a remote node as a user, you would define that main node as your peer:

```
[sashimi_platter]
type=peer
username=sushi
secret=wasabi
host=192.168.1.101
qualify=yes
trunk=yes
```

A *peer* can be referenced from the dialplan with the name contained in square brackets but authenticate with a different username. The `host` is specified using either IP dotted notation or a fully qualified domain name (FQDN). You can determine the latency between you and the remote host, and whether the peer is alive, with `qualify=yes`. To minimize the amount of overhead for multiple calls going to the same peer, you can `trunk` them.

Trunking is unique to IAX and is designed to take advantage of the fact that two large sites may have multiple simultaneous VoIP connections between them. IAX trunking reduces overhead by loading several channels into each signaling packet. You can enable trunking for a channel with `trunk=yes` in *iax.conf*.

Figure A-1 shows a channel with trunking disabled, and Figure A-2 shows a channel with trunking enabled.

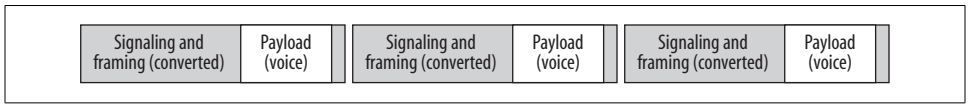


Figure A-1. Trunking disabled

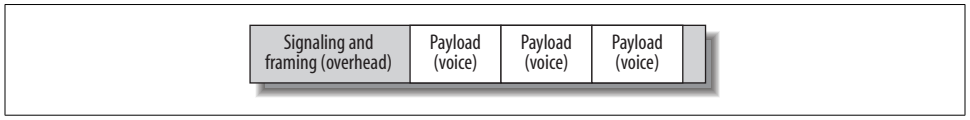


Figure A-2. Trunking enabled

Channel-specific parameters

Now, let's take a look at the channel-specific parameters:

`callerid`
: You can set a suggested Caller ID string for a user or peer with `callerid`. If you define a Caller ID field for a user, any calls that come in on that channel will have that Caller ID assigned to them, regardless of what the far end sends to you. If you define Caller ID for a *peer*, you are requesting that the far end use that to identify you (although you have no way of ensuring it will do so). If you want incoming users to be able to define their own Caller IDs (i.e., for guests), make sure you do not set the `callerid` field.

```
callerid=John Smith <(800) 555-1234>
```

`defaultip`
: The `defaultip` setting complements `host=dynamic`. If a host has not yet registered with your server, you'll attempt to send messages to the default IP address configured here.

```
defaultip=192.168.1.101
```

`inkeys`
: You can use the `inkeys` option to authenticate a user with the use of an RSA key. To associate more than one RSA key with a user channel definition, separate the key names with a colon (`:`). Any one of those keys will be sufficient to validate a connection. The "inkey" is the public key you distribute to your users.

```
inkeys=server_one:server_two
```

`mailbox`
: If you associate a mailbox with a peer within the channel definition, voicemail will send a message waiting indication (MWI) to the nodes on the end of that channel. If the mailbox number is in a voicemail context other than `default`, you can specify it as *`mailbox@context`*. To associate multiple mailboxes with a single peer, use multiple `mailbox` statements.

```
mailbox=1000@internal
```

`outkey`
: You can use the `outkey` option to authenticate a peer with the use of an RSA key. Only one RSA key may be used for outgoing authentication. The "outkey" is not distributed; it is your private key.

```
outkey=private_key
```

`qualify`
: You can set `qualify` to `yes`, `no`, or a time in milliseconds. If you set `qualify=yes`, `PING` messages will be sent periodically to the remote peers to determine whether they are available and what the latency between replies is. The peers will respond with `PONG` messages. A peer will be determined unreachable if no reply is received within 2,000 ms (to change this default, instead set `qualify` to the number of milliseconds to wait for the reply).

`sendani`
: The SS7 PSTN network uses Automatic Number Identification (ANI) to identify a caller, and Caller ID is what is delivered to the user. The Caller ID is generated from the ANI, so it's easy to confuse the two. Blocking Caller ID sets a privacy flag on the ANI, but the backbone network still knows where the call is coming from.

```
sendani=yes
```

ANI has been around for a while. Its original purpose was to deliver the billing number of the originating party on a long-distance call to the terminating office. Unlike Caller ID, ANI does not require SS7, as it can be transmitted using DTMF. Also, ANI cannot be blocked.

SIP

Just as with IAX, the SIP configuration file (*sip.conf*) contains configuration information for SIP channels. The headings for the channel definitions are formed by a word framed in square brackets (`[]`)—again, with the exception of the `[general]` section, where we define global SIP parameters. Don't forget to use comments generously in your *sip.conf* file. Precede the comment text with a semicolon; everything to the right will be ignored.

General SIP Parameters

The following options are to be used within the `[general]` section of *sip.conf*:

`allowguest`
: If set to `no`, this disallows guest SIP connections. The default is to allow guest connections. SIP normally requires authentication, but you can accept calls from users who do not support authentication (i.e., do not have a `secret` field defined). Certain SIP appliances (such as the Cisco Call Manager v4.1) do not support authentication, so they will not be able to connect if you set `allowguest=no`.

```
allowguest=no
```

`bindaddr` *and* `bindport`
: These optional parameters allow you to control the IP interface and port on which you wish to accept SIP connections. If omitted, the port will be set to 5060, and all IP addresses in your Asterisk system will accept incoming SIP connections. If multiple bind addresses are configured, only those interfaces will listen for connections. The address 0.0.0.0 tells Asterisk to listen on all interfaces.

```
bindaddr=0.0.0.0
bindport=5060
```

`callevents`
: Set this to `yes` when you want SIP to generate Manager events. This will be important if you have external programs that use the Asterisk Manager interface, such as the Flash Operator Panel.

```
callevents=yes
```

`checkmwi`
: This option specifies the default amount of time, in seconds, between mailbox checks for peers.

```
checkmwi=30
```

`compactheaders`
: You can set `compactheaders` to `yes` or `no`. If it's set to `yes`, the SIP headers will use a compact format, which may be required if the size of the SIP header is larger than the maximum transmission unit (MTU) of your IP headers, causing the IP packet to be fragmented. Do not use this option unless you know what you are doing.

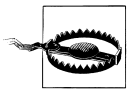
```
compactheaders=no
```

`defaultexpirey`
: This sets the default SIP registration expiration time, in seconds, for incoming and outgoing registrations. A client will normally define this value when it initially registers, so the default value you set here will be used only if the client does not specify a timeout when it registers. If you are registering to another user agent server (UAS), this is the registration timeout that it will send to the far end.

```
defaultexpirey=300
```

`externhost`
: `externhost` takes a fully qualified domain name as its argument. If Asterisk is behind NAT, the SIP header will normally use the private IP address assigned to the server. If you set this option, Asterisk will perform periodic DNS lookups on the hostname and replace the private IP address with the IP address returned from the DNS lookup.

```
externhost=my.hostname.tld
```



The use of `externhost` is not recommended in production systems, because if the IP address of the server changes, the wrong IP address will be set in the SIP headers until the next lookup is performed. The use of `externip` is recommended instead.

`externip`
: `externip` takes an IP address as its argument. If Asterisk is behind NAT, the SIP header will normally use the private IP address assigned to the server. The remote server will not know how to route back to this address; thus, it must be replaced with a valid, routable address.

```
externip=216.239.39.104
```

`externrefresh`
: If `externhost` is used, `externrefresh` configures how long, in seconds, should pass between DNS lookups.

```
externrefresh=30
```

`localnet`
: `localnet` is used to tell Asterisk which IP addresses are considered local, so that the address in the SIP header can be translated to that specified by `externip` or the IP address can be looked up with `externhost`.

```
localnet=192.168.1.0/24
localnet=172.16.0.0/16
```

`maxexpirey`
: This sets the maximum amount of time, in seconds, until a peer's registration expires.

```
maxexpirey=3600
```

`notifymimetype`
: This takes as its argument a string specifying the MIME type used for the message waiting notification (MWI) in the SIP `NOTIFY` message. The most common setting for this field is `text/plain`, although it can be customized if need be.

```
notifymimetype=text/plain
```

`pedantic`
: You can set `pedantic` to `yes` or `no`. Setting it to `yes` enables slow pedantic checking for phones that require it, such as the Pingtel, and enables more strict SIP RFC compliancy. In an effort to improve performance, SIP RFC compliance is not normally strictly adhered to.

```
pedantic=yes
```

`realm`
: This option sets the realm for digest authentication. Set `realm` to your fully qualified domain name, which must be globally unique.

```
realm=my.hostname.tld
```

`recordhistory`
: You can set `recordhistory` to `yes` or `no` to enable or disable SIP history recording for all channels. (See `sip history` and `sip no history` in Appendix E.)

```
recordhistory=yes
```

`relaxdtmf`
: You can set `relaxdtmf` to `yes` or `no`. Setting it to `yes` will relax the DTMF detection handling. Use this if Asterisk is having a difficult time determining the DTMF on the SIP channel. Note that this may cause "talkoff," where Asterisk incorrectly detects DTMF when it should not.

```
relaxdtmf=yes
```

`srvlookup`
: DNS SRV records are a way of setting up a logical, resolvable address where you can be reached. This allows calls to be forwarded to different locations without the need to change the logical address. By using SRV records, you gain many of the advantages of DNS, whereas disabling them removes the ability to place SIP calls based on domain names. (Note that if multiple records are returned, Asterisk will use only the first.) DNS SRV record lookups are recommended. To enable them, set `srvlookup=yes` in the `[general]` section of *sip.conf*.

```
srvlookup=yes
```

`tos`
: Asterisk can set the Type of Service (TOS) bits in the IP header to help improve performance on routers that respect TOS bits in their routing calculations. The following values are valid:

  `lowdelay`
  : Minimize delay.

  `throughput`
  : Maximize throughput.

  `reliability`
  : Maximize reliability.

  `mincost`
  : Minimize cost.

  `none`
  : No bits set.

```
tos=lowdelay|throughput|reliability|mincost|none
```

`useragent`
: `useragent` takes as its argument a string specifying the value for the `useragent` field in the SIP header. The default value is `asterisk`.

```
useragent=asterisk
```

`videosupport`
: You can set `videosupport` to `yes` or `no`. Setting it to `yes` will enable SIP video support. Video support works only between two endpoints—Asterisk does not support video conferencing at this time.

```
videosupport=yes
```

SIP Channel Definitions

Now that we've covered the global SIP parameters, we will discuss the channel-specific parameters. These parameters can be defined for a user, a peer, or both (as noted in parentheses):

`accountcode` *(both)*

The account code can be defined on a per-user basis. If defined, this account code will be assigned to a call record whenever no specific user account code is set. The `accountcode` name configured will be used as the *filename.csv* in the */var/log/asterisk/cdr-csv/* directory to store CDRs for the user/peer/friend.

```
accountcode=iax-username
```

`allow` *and* `disallow` *(both)*

Specific codecs can be allowed or disallowed, limiting codec use to those preferred by the system designer. `allow` and `disallow` can also be defined on a per-channel basis. Keep in mind that allow statements in the `[general]` section will carry over to each of the channels, unless you reset with a `disallow=all`. Codec negotiation is attempted in the order in which the codecs are defined. Best practice suggests that you define `disallow=all`, followed by explicit `allow` statements for each codec you wish to use. If nothing is defined, `allow=all` is assumed.

```
disallow=all
allow=ulaw
allow=gsm
allow=ilbc
```

`amaflags` *(both)*

Automatic Message Accounting (AMA) is defined in the Telcordia Family of Documents listed under FR-AMA-1. These documents specify standard mechanisms for generation and transmission of CDRs. You can specify one of four AMA flags (`default`, `omit`, `billing`, or `documentation`) to apply to all SIP connections.

```
amaflags=documentation
```

`callerid` *(both)*

You can set a suggested Caller ID string for a user or peer with `callerid`. If you define a Caller ID field for a user, any calls that come in on that channel will have that Caller ID assigned to them, regardless of what the far end sends to you. If Caller ID is defined for a peer, you are requesting that the far end use that to identify you (keep in mind, however, that you have no way to ensure that it will do so). If you want incoming callers to be able to define their own Caller IDs (i.e., for guests), make sure you do not set the `callerid` field.

```
callerid=John Smith <(800) 555-1234>
```

`callgroup` *and* `pickupgroup` *(both)*

You can use the `callgroup` parameter to assign a channel definition to one or more groups, and you can use the `pickupgroup` option in conjunction with this parameter to allow a ringing phone to be answered from another extension. The `pickupgroup` option is used to control which callgroups a channel may pick up—a channel is given authority to answer another ringing channel if it is assigned to the same `pickupgroup` as the ringing channel's callgroup. By default, remote ringing extensions can be answered with `*8` (this is configurable in the *features.conf* file).

```
callgroup=1,3-5
pickupgroup=1,3-5
```

`canreinvite` *(both)*

The SIP protocol tries to connect endpoints directly. However, Asterisk must remain in the transmission path between the endpoints if it is required to detect DTMF. (For more information, see Chapter 4.)

```
canreinvite=no
```

`context` *(both)*[*]

A context is assigned to a channel definition to direct incoming calls into the matching context in *extensions.conf*, where call handling is performed (see Chapters 4 and 5). Any channel connecting to an Asterisk machine has to have a context defined into which it will arrive. The context is essential for any `user` channel definition—if you do not define a context, incoming calls will be directed to the `default` context.

```
context=incoming
```

`defaultip` *(peer)*

The `defaultip` setting complements `host=dynamic`. If a host has not yet registered with your server, you'll attempt to send messages to the default IP address configured here.

```
defaultip=192.168.1.101
```

`deny` *(both)*

Specific IP addresses and ranges can be controlled with the `deny` option. To restrict access from a range of IP addresses, use a subnet mask—for example, `deny=192.168.1.0/255.255.255.0`. You can also deny all addresses with `deny=0.0.0.0/0.0.0.0` and then allow only certain addresses with the `permit` command. Be aware of the security implications of this setting. (See also `permit`.)

```
deny=0.0.0.0/0.0.0.0
```

`disallow` *(both)*

See `allow`.

`dtmfmode` *(both)*

You can set `dtmfmode` to `inband`, `rfc2833`, or `info`. DTMF digits can be sent either in band (as part of the audio stream), or out of band (as signaling information), using the RFC 2833 or INFO methods. The `inband` method only works reliably when using an uncompressed codec such as G.711, ulaw, or alaw. The recommended method is to use `rfc2833`; however, some devices—such as those by Grandstream—support the `info` method.

```
dtmfmode=rfc2833
```

[*] You should be aware of an unusual scenario that will require a context definition for a peer. If a SIP call that originated from your system returns to your system for some reason (perhaps because a call you sent to a remote Asterisk box has been forwarded back to you), that call will authenticate not from a `user` definition (as would be expected), but rather from a `peer` definition. Since peers don't normally have contexts, this will cause such a call to arrive in the `default` context. While this will work, the `default` context shouldn't really be used to handle incoming calls. The solution is to define a context, on a per-peer basis, for any peers that may return calls to you (such as in a call-forwarding situation, or if the remote end is a SIP proxy server). To experiment with this, you can call your Free World Dialup number; the call will come right back to you.

`fromdomain` *(peer)*

This allows you to set the domain in the `From:` field of the SIP header. It may be required by some providers for authentication.

```
fromdomain=my.hostname.tld
```

`fromuser` *(peer)*

This allows you to set the username with which to authenticate. The name contained within the square brackets of the channel definition is usually used, but this can be overridden with the `fromuser` option. This allows a channel definition to be referenced with a name other than that used to authenticate.

```
fromuser=john_smith
```

`host` *(peer)*

This configures the host to which this peer is to connect. Use a fully qualified domain name.

```
host=remote.hostname.tld
```

`incominglimit` *(both)*

This option limits the total number of simultaneous calls for a peer or user. It sets the max number of simultaneous outgoing calls for a peer, or the max number of incoming calls for a user.

```
incominglimit=3
```

`insecure` *(both)*

When an `INVITE` is received from a remote location, Asterisk attempts to authenticate the string of characters before the `@` sign on the `INVITE` line received in the SIP header with the name of a channel definition in *sip.conf*. If the remote end is a user agent, it will authenticate based on a `user` definition. However, if the remote end is a SIP proxy service, it will authenticate on the `peer` entry. When calls come from a provider such as Free World Dialup, which acts as a proxy for the true remote end who is calling you, that provider cannot authenticate the call on behalf of the endpoint. Since it would be impractical to have an authentication configured for every FWD user, and since FWD cannot respond to a 407 Proxy Authentication Required response, there must be an alternate way to allow calls from these callers.

If you set `insecure=invite`, you'll determine which peer to match on by comparing the IP address or hostname and port number to those provided in the `Contact` field of the SIP header with the `host` and `port` options in *sip.conf*. If a match is found, authentication will not be required on the initial `INVITE`, and the call will be allowed.

If you have multiple endpoints behind a NAT device, you need to enable `insecure=port` to match only against the IP address. To not require authentication on the incoming `INVITE` for the peer, set `insecure=invite,port`.

```
insecure=invite
```

`language` *(both)*

This sets the language flag to whatever you define. The global default language is English. The language that is set is sent by the channel as an information element. It is also used by applications such as `SayNumber( )` that have different files for different languages. Keep in mind that languages other than English are not explicitly installed on the system, and it is up to you to configure the system to ensure that the language you specify is handled properly.

```
language=en
```

`mailbox` *(peer)*

If you associate a `mailbox` with a peer within the channel definition, voicemail will send a message waiting indication to the nodes on the end of that channel. If the mailbox number is in a voicemail context other than `default`, you can specify it as *`mailbox@context`*. To associate multiple mailboxes with a single peer, use multiple `mailbox` statements.

```
mailbox=1000@internal
```

`md5secret` *(both)*

If you do not wish to have plain-text secrets in your *sip.conf* files, you can use `md5secret` to configure the MD5 hash that can be used for authentication. To generate the MD5 hash from the Linux console, use the following command:

```
# echo -n "username:realm:secret" | md5sum
```

Be sure to use the `–n` flag, or `echo` will add a `\n` to the end of the string; the line feed will then be calculated into the MD5 hash, creating the incorrect hash. The *`realm`*, if not specified with the `realm` option (discussed in the list of general SIP parameters), defaults to `asterisk`. If both an `md5secret` and a `secret` are specified in the same channel definition, the `secret` will be ignored.

```
md5secret=0bcbe762982374c276fb01af6d272dca
```

`musicclass` *(both)*

This option sets the default Music on Hold class.

```
musicclass=classical
```

`nat` *(both)*

You can set `nat` to `yes`, `no`, or `never`. If you set it to `yes`, Asterisk ignores the IP address in the SIP and SDP headers and responds to the address and port in the IP header. The `never` option is for devices that cannot handle `rport` in the SIP header, such as the Uniden UIP200.

```
nat=yes
```

`permit` *(both)*

See `deny`.

`pickupgroup` *(both)*

See `callgroup`.

`port` *(peer)*

You can use this to define the port on which to listen for SIP signaling, if you want to listen on a nonstandard port. (The default port for SIP signaling is 5060.)

```
port=5060
```

`progressinband` *(both)*

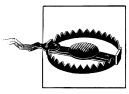
You can set `progressinband` to `yes`, `no`, or `never`, to configure whether or not to generate in-band ringing. Normally, Asterisk will send the progress of a call via a few methods, such as 183 Session Progress, 180 Ringing, 486 Busy, and so on. If you set `progressinband=yes`, Asterisk will indicate the call progress in band by generating tones.

```
progressinband=yes
```

`promiscredir` *(both)*

You can set `promiscredir` to `yes` or `no`. Normally, when you perform call forwarding on a phone, Asterisk will use the Local channel (for example, *Local/18005551212@peer*). If you set `promiscredir=yes`, Asterisk will use the SIP channel instead, which enables you to forward the calls to remote boxes.

```
promiscredir=yes
```



Note that if Asterisk performs a redirect to itself when `promiscredir=yes`, the system will receive an `INVITE` with the same Caller ID and detect a loop to itself. SIP does not have the ability to perform a hairpin call, so the channel will then be destroyed.

`qualify` *(peer)*

You can set `qualify` to `yes`, `no`, or a time in milliseconds. If you set `qualify=yes`, `NOTIFY` messages will be sent periodically to the remote peers to determine whether they are available and what the latency between replies is. A peer is determined unreachable if no reply is received within 2,000 ms (to change this default, instead set `qualify` to the number of milliseconds to wait for the reply). Use this option in conjunction with `nat=yes` to keep the path through the NAT device alive.

```
qualify=yes
```

`regcontext` *(peer)*

By specifying the context that contains the actions to perform, you can configure Asterisk to perform a number of actions when a peer registers to your server. This option works in conjunction with `regexten`, by specifying the extension to execute. If no `regexten` is configured, the peer name is used as the extension. Asterisk will dynamically create and destroy a `NoOp` at priority 1 for the extension. All actions to be performed upon registration should start at priority 2. More than one `regexten` may be supplied, if separated by an `&`. `regcontext` can be set on a per-peer basis or globally.

```
regcontext=peer_registrations
```

`regexten` *(peer)*
: The `regexten` option is used in conjunction with `regcontext` to specify the extension that is executed within the configured context. If `regexten` is not explicitly configured, the peer name is used as the extension to match.

```
regexten=1000
```

`rtpholdtimeout` *(peer)*
: This takes as its argument an integer, specified in seconds. It terminates a call if no RTP data is received while on hold. The value of `rtpholdtimeout` must be greater than that of `rtptimeout`. (See also `rtptimeout`.)

```
rtpholdtimeout=120
```

`rtptimeout` *(peer)*
: This takes as its argument an integer, specified in seconds. It terminates a call if no RTP data is received within the time specified.

```
rtptimeout=60
```

`secret` *(both)*
: This sets the password to use for authentication.

```
secret=welcome
```

`setvar` *(both)*
: This sets a channel variable, which will be available when a channel to the peer or user is created and will be destroyed when the call is hung up. For example, to set the channel variable `foo` with a value of `bar`, use `setvar=foo=bar`.

```
setvar=foo=bar
```

`username` *(peer)*
: The `username` field allows you to attempt contact with a peer before it has registered with you. At registration, a SIP device tells Asterisk which SIP URI to use to contact it. The username is used in conjunction with `defaultip` to create the SIP URI in the SIP `INVITE` header. This might be useful following a reboot, in order to place a call. The endpoints will not attempt to register with the server until their registration timeouts expire, so you will not know their locations. For non-dynamic hosts, you will require the username to be specified, as it is used to construct the authorization username.

```
username=john_smith
```