



AMI v2 Specification

Created by Matt Jordan, last modified on Dec 20, 2013

- Introduction
 - Scope
 - Versioning
- Terminology
- Protocol Overview
- Semantics and Syntax
 - Message Sending and Receiving
 - Message Layout
 - Common Fields
 - Actions
 - Events
 - Channel Interaction/Lifetime
 - Basic Channel Lifetime
 - Channel Variables
 - DTMF
 - Dialplan Execution
 - Dialing and Origination
 - Bridging
 - Two Party Bridging
 - Transfers
 - Local Channel Optimization
 - Masquerades
- Transports
- Security Considerations
 - Class Authorizations
 - Access Control Lists
 - Authorization
- AMI Configuration
 - General Settings
 - Client Settings

Introduction

This Asterisk Manager Interface (AMI) specification describes the relationship between Asterisk and an external entity wishing to communicate with Asterisk over the AMI protocol. It describes:

- An overview of the AMI protocol
- The operations AMI provides external entities wishing to control Asterisk
- Basic formatting of AMI message structures
- Guaranteed operations, configuration control, and other information provided by Asterisk in AMI v2.x

Scope

This specification describes AMI version 2, specifically for Asterisk 12. This specification provides details on the functional, operational and design requirements for AMI version 2. Note that this does not include a comprehensive listing of the AMI configuration file parameters or messages that a system interfacing over AMI in Asterisk 12 will send/receive; however, it does provide a baseline of the supported features and messages provided in AMI version 2. This specification should be used in conjunction with the documented AMI actions and events in Asterisk 12 to encompass the full range of functionality provided by AMI in Asterisk 12.

In addition, this specification provides interface requirements levied on AMI by Stasis, a message bus internal to Asterisk. It conveys sufficient detail to understand how AMI attaches to the Stasis message bus and interacts with other entities on Stasis.

This specification is intended for all parties requiring such information, including software developers, system designers and testers responsible for implementing the interface.

Versioning

Starting in Asterisk 12, AMI now follows semantic versioning. The initial release of Asterisk 12 has an AMI version of 2.0.0.

A full description of semantic versioning can be found at <http://semver.org/>.

Terminology

Term	Definition
Action	A command issued to Asterisk from an external entity via AMI
Client	An external entity communicating with Asterisk via AMI over some transport mechanism
Event	A message sent from Asterisk to an external entity via AMI
Field	A key/value pair that exists in either an action or event
Stasis	The internal framework that AMI is built on top of

Protocol Overview

Asterisk provides a number of interfaces that serve different purposes. Say, for example, we wanted to manipulate a call between Alice and Bob via some external mechanism. Depending on what we wanted to do with the call, we may use one or more interfaces to manipulate the channels that make up the call between Alice and Bob.

Alice calls Bob and...	Interface
... we want to use a local script to execute some logic on Alice's channel	AGI
... we want to execute a script on a remote machine on Bob's channel	FastAGI
... we want to put Alice into an IVR with fine grained media control, where the IVR is written outside of <code>extensions.conf</code>	ExternalIVR
... we want to control Alice and Bob's underlying channel objects at some asynchronous time	AMI (possibly with AsyncAGI)
... we want to write our own Dialling application to control both Alice and Bob	ARI

In general, AMI is used to manage Asterisk and its channels. It does not determine what actions are executed on a particular channel - the dialplan and/or an AGI interface does that - but it does allow a client to control call generation, aspects of call flow, and other internals of Asterisk.


At its heart, AMI is an asynchronous message bus: it spills **events** that contain information about the Asterisk system over some transport. In response, clients may request that Asterisk takes some **action**. These two concepts - actions and events - make up the core of what is AMI. As AMI is asynchronous, as events occur in Asterisk they are immediately sent to the clients. This means that actions issued by entities happen without any synchronization with the events being received, even if those events occur in response to an action. It is the responsibility of entities to associate event responses back to actions.

Clients wishing to use AMI act as clients and connect to Asterisk's AMI server over a supported transport mechanism. Authentication may or may not be enabled, depending on the configuration. Once connected, events can be automatically spilled to the connected clients, or limited in a variety of fashions. A connected client can send an action to the AMI server at any time. Depending on the allowed authorizations, the action may be allowed or disallowed.

More information on the various ways a client can be configured can be seen in [AMI Configuration](#).

Sometimes, the term **command** may be used instead of the term **action**. With respect to AMI actions, command is synonymous with action, and the two can be treated the same. For the sake of consistency, we've attempted to use the term **action** where possible.

Historically, AMI has existed in Asterisk as its own core component manager. AMI events were raised throughout Asterisk encoded in an AMI specific format, and AMI actions were processed and passed to the functions that implemented the logic. In Asterisk 12, AMI has been refactored to sit on top of Stasis, a generic, protocol independent message bus internal to Asterisk. From the perspective of clients wishing to communicate with Asterisk over AMI very little has changed; internally, the Stasis representation affords a much higher degree of flexibility with how messages move through Asterisk. It also provides a degree of uniformity for information that is propagated to interested parties.

 Unknown macro: 'table'

Semantics and Syntax

Message Sending and Receiving

By default, AMI is an asynchronous protocol that sends events immediately to clients when those events are available. Likewise, clients are free to send actions to AMI at any time, which may or may not trigger additional events. The exception to this is when the connection is over HTTP; in that scenario, events are only transmitted as part of the response to an HTTP POST.

Various options for configuration of clients can control which events are sent to a client. Events can be whitelisted/blacklisted explicitly via event filters, or implicitly by [class authorizations](#).

Message Layout

AMI is an ASCII protocol that provides bidirectional communication with clients. An AMI message – action or event – is composed of fields delineated by the '\r\n' characters. Within a message, each field is a key value pair delineated by a ':'. A single space MUST follow the ':' and precede the value. Fields with the same key may be repeated within an AMI message. An action or event is terminated by an additional '\r\n' character.

```
Event: Newchannel
Privilege: call,all
Channel: PJSIP/misspiggy-00000001
Uniqueid: 1368479157.3
ChannelState: 3
ChannelStateDesc: Up
CallerIDNum: 657-5309
CallerIDName: Miss Piggy
ConnectedLineName:
ConnectedLineNum:
AccountCode: Pork
Priority: 1
Exten: 31337
Context: inbound
```

This is syntactically equivalent to the following ASCII string:

```
Event: Newchannel\r\nPrivilege: call,all\r\nChannel: PJSIP/misspiggy-00000001\r\nUniqueid: 136847
```

Actions are specified in a similar manner. Note that depending on the message, some keys can be repeated.

```
Action: Originate
ActionId: SDY4-12837-123878782
Channel: PJSIP/kermit-00000002
Context: outbound
Exten: s
Priority: 1
CallerID: "Kermit the Frog" <123-4567>
Account: FrogLegs
Variable: MY_VAR=frogs
Variable: HIDE_FROM_CHEF=true
```

In addition, no ordering is implied on message specific keys. Hence, the following two messages are semantically the same.

```
Action: Originate
ActionId: SDY4-12837-123878782
Channel: PJSIP/kermit-00000002
Context: outbound
Exten: s
Priority: 1
CallerID: "Kermit the Frog" <123-4567>
Account: FrogLegs
Variable: MY_VAR=frogs
Variable: HIDE_FROM_CHEF=true
```

```
ActionId: SDY4-12837-123878782
Action: Originate
Variable: HIDE_FROM_CHEF=true
Variable: MY_VAR=frogs
Channel: PJSIP/kermit-00000002
Account: FrogLegs
Context: outbound
Exten: s
CallerID: "Kermit the Frog" <123-4567>
Priority: 1
```

This is also true for events, although by convention, the `Event` key is the first key in the event. If an action or event contains duplicate keys, such as `Variable`, the order in which Asterisk processes said keys is the order in which they occur within the action or event.

Keys are case insensitive. Hence, the following keys are equivalent:

```
Action: Originate
```

```
ACTION: Originate
```

```
action: Originate
```

The case sensitivity for values is left up to the context in which they are interpreted.

Common Fields

Actions

General Fields

This section lists fields that apply generally to all actions.

Action

Action specifies the action to execute within Asterisk. Each value corresponds to a unique action to execute within Asterisk. The value of the **Action** field determines the allowed fields within the rest of the message. By convention, the first field in any action is the **Action** field.

ActionId

ActionId is a universal unique identifier that can optionally be provided with an action. If provided in an action, events that are related to that action will contain the same **ActionId** value, allowing a client to associate actions with events that were caused by that action.

It is recommended that clients always provide an **ActionId** for each action they submit.

It is up to the client to ensure that the **ActionId** provided with an **Action** is unique.

Channels

This section lists fields that apply generally to all actions that interact upon an Asterisk channel. Note that an Action that interacts with a channel *must* supply the *Channel* field.

Upgrading

In the past, AMI clients would have to contend with channel rename events. As Asterisk will now no longer change the name of a channel during its lifetime, this is no longer necessary.

Channel

The Asterisk channel name. A channel name is provided by AMI to clients during a **Newchannel** event. A channel name can be viewed as the handle to a channel.

Uniqueid

A universal unique identifier for the channel. In systems with multiple Asterisk instances, this field can be used to construct a globally unique identifier for a channel, as a channel name may occur multiple times across Asterisk instances.

Events

General Fields

This section lists fields that apply generally to all events.

Event

The unique name of the event being raised. The value of the **Event** field determines the rest of the contents of the message. By convention, the **Event** field is the first field in an AMI message.

ActionId

If present, the Action's corresponding [ActionId](#) that caused this event to be created. If an Action contained an **ActionId**, any event relating the success or failure of that action **MUST** contain an **ActionId** field with the same value.

Privilege

The class authorizations associated with this particular event. The class authorizations for a particular event are in a comma-delineated list. For more information, see [class authorizations](#).

Event responses to an Action only occur if the Action was executed, which means the user had the appropriate class authorization. Therefore they will not have a Privilege field.

Channels

This section lists fields that apply generally to all events that occur due to interactions upon an Asterisk channel.

Events that relate multiple channels will prefix these fields with an event specific role specifier. For example, a **DialBegin** or a **DialEnd** event will prefix the outbound channel's fields with **Dest**. So, the **Channel** field is the **DestChannel** field; the **Uniqueid** field is the **DestUniqueid** field, etc.

Channel

The current Asterisk channel name. This corresponds to the [Channel](#) field in actions.

Uniqueid

A universal unique identifier for the channel. This corresponds to the [Uniqueid](#) field in actions.

ChannelState

The current state of the channel, represented as an integer value. The valid values are:

Value	State	Description
0	Down	Channel is down and available.
1	Rsrvd	Channel is down, but reserved.
2	OffHook	Channel is off hook.
3	Dialing	The channel is in the midst of a dialing operation.
4	Ring	The channel is ringing.
5	Ringing	The remote endpoint is ringing. Note that for many channel technologies, this is the same as Ring.
6	Up	A communication path is established between the endpoint and Asterisk.
7	Busy	A busy indication has occurred on the channel.
8	Dialing Offhook	Digits (or equivalent) have been dialed while offhook.
9	Pre-ring	The channel technology has detected an incoming call and is waiting for a ringing indication.
10	Unknown	The channel is an unknown state.

Depending on the underlying channel technology, not all states will be used. Channels typically begin in either the Down or Up states.

ChannelStateDesc

The text description of the channel state. This will be one of the State descriptions in the table in [ChannelState](#).

CallerIDNum

The current caller ID number. If the caller ID number is not known, the string "<unknown>" is returned instead.

CallerIDName

The current caller ID name. If the caller ID name is not known, the string "<unknown>" is returned instead.

ConnectedLineNum

The current connected line number. If the connected line number is not known, the string "<unknown>" is returned instead.

ConnectedLineName

The current connected line name. If the connected line name is not known, the string "<unknown>" is returned instead.

AccountCode

The channel's accountcode.

Context

The current context in the dialplan that the channel is executing in.

Exten

The current extension in the dialplan that the channel is executing in.

Priority

The current priority of the current context, extension in the dialplan that the channel is executing in.

ChanVariable

Channel variables specific to a channel can be conveyed in each AMI event related to that channel. When this occurs, each variable is referenced in a **ChanVariable** field. The value of a **ChanVariable** field will always be of the form `key=value`, where `key` is the name of the channel variable and `value` is its value.

Bridges

BridgeUniqueId

A unique identifier for the bridge, which provides a handle to actions that manipulate bridges.

BridgeType

The type of the bridge. Bridge types determine how a participant in a bridge can behave. For example, a 'base' bridge is a bridge that has few inherent properties or features associated with it, while a 'parking' bridge is one used for a parking application. Specific modules within Asterisk will determine the type of bridge that is created.

Note that this is not the same as how media within a bridge is mixed. How media is mixed between participants in a bridge is determined by the **BridgeTechnology**.

BridgeTechnology

How the media can be mixed within a bridge. Specific modules in Asterisk provide different bridge technologies that can be used by Asterisk to alter how media passes between the participants. For a given bridge, the **BridgeTechnology*** can also change as the number and type of participants change. The most common bridge technologies are:

- `holding_bridge` – normal participants joining the bridge may receive audio, but audio sent from a normal participant is dropped. Special participants, known as announcers, may broadcast their audio to all normal participants.
- `native_dahdi` – a native bridge between DAHDI channels. Media is passed directly between all participants.
- `native_rtp` – a native bridge between channels that use RTP for media. Media is passed directly between all participants.
- `simple_bridge` – a two-party bridge between any two channels. Media is passed through the Asterisk core between the two participants.
- `softmix` – a multi-party bridge between one or more participants. All media from all participants is mixed together and sent to all participants.

BridgeCreator

Some bridges are created as the result of a particular application or action. If so, the bridge will specify who created it. If the bridge was not created as a result of any particular application or action, the field will have the value `<unknown>`.

BridgeName

Some bridges are created with a names as a result of their application. If so, the bridge will specify the name given to it. If the bridge was created without a name, the field will have the value `<unknown>`.

BridgeNumChannels

The number of channels currently in the bridge.

Action Responses

When an Action is submitted to AMI, the success or failure of the action is communicated in subsequent events.

Response

Contains whether or not the action succeeded or failed. Valid values are "Success" or "Error". Events that are in response to an action MUST include this field.

EventList

Some actions will cause a chain of events to be created. Events that are a response to an action that causes such a sequence will contain the EventList field with a value of "start". When all generated events have been sent, a final event will be sent containing the EventList field with the value "complete".

If, for some reason, an error occurs and the events cannot be sent, an event will be sent with an EventList field that contains the value "cancelled".

Note that the events that mark the completion or cancellation of an event list are not technically action responses, and have their own specific event types.

Message

An optional text message that provides additional contextual information regarding the success or failure of the action.

Actions

The supported actions for Asterisk 12 are listed here:

[Asterisk 12 AMI Actions](#)

While new AMI Actions may be added over the lifetime of Asterisk 12, existing AMI Actions will **not** be removed.

Optional fields may be added to an existing AMI action with altering the AMI version number, but required fields will **not** be added or removed.

Events

The supported events for Asterisk 12 are listed here:

[Asterisk 12 AMI Events](#)

While new AMI Events may be added over the lifetime of Asterisk 12, existing AMI Events will **not** be removed.

Fields may be added to an existing AMI event without altering the AMI version number, but existing fields will **not** be removed.

Channel Interaction/Lifetime

While channels are independent of AMI, they have a large implication on the events sent out over AMI. Many of the events in AMI correspond to changes in channel state. While AMI is an asynchronous protocol, there is some ordering with respect to the events that are relayed for a particular channel. This section provides the basic event relationships that are guaranteed through AMI.

Basic Channel Lifetime

All channels begin with a **Newchannel** event. A **Newchannel** will always contain the following fields:


- The current [Channel](#) name that acts as a handle to the channel for that channel's lifetime for a single Asterisk system.
- The [Uniqueid](#) for the channel, that allows systems to have a globally unique identifier for the channel.

Changes in the state of the channel, i.e., the [ChannelState](#) field, are conveyed via **Newstate** events.

Notification of a Channel being disposed of occurs via a **Hangup** event. A **Hangup** signals the termination of the channel associated with the Uniqueid. After the **Hangup** event, no further events will be raised in relation to the channel with that Uniqueid, and the communication between the endpoint and Asterisk via that channel is terminated.

The examples in this specification do not show all of the fields in every event. For a full listing of all of the fields, see the documentation for the specific event in [Asterisk 12 AMI Events](#).

Example

 Unknown macro: 'table'

Channel Variables

For each channel variable that is changed, a **VarSet** event is sent to the client. The **VarSet** event contains the new value of the appropriate channel variable. Note that channel variables can also be conveyed in [ChanVariable](#) fields.

DTMF

DTMF is indicated via a **DTMFBegin/DTMFEnd** events. A **DTMFEnd** event MUST convey the duration of the DTMF tone in milliseconds.


Behavior Change

The combination of **DTMFBegin/DTMFEnd** events replaces the removed **DTMF** event.

Dialplan Execution

As a channel executes operations in the dialplan, those operations are conveyed via a **NewExten** event. Each transition to a new combination of context, extension, and priority will trigger a **NewExten** event.

Example

 Unknown macro: 'table'

Dialing and Origination

Dial operations always result in two events: a **DialBegin** event that signals the beginning of the dial to a particular destination, and a **DialEnd** event that signals the end of the dialing. In parallel dialing situations, **DialBegin/DialEnd** events MUST be sent for each channel dialed. For each **DialBegin** event sent, there MUST be a corresponding **DialEnd** event.

In dialing situations with a caller and a called party, the **DialBegin** and **DialEnd** events convey information about both channels. The calling channel uses the standard channel field names, while the called party's field names are prefixed with "Dest". In dialing situations where there is no caller, such as when Asterisk originates an outbound call via a call file, only the called channel is represented in the events. The channel field names are still prefixed with "Dest" in this case; the standard channel field names are **not** present in the event in this case.

A **DialEnd** occurs whenever Asterisk knows the final state of the channel that it was attempting to establish. The status is communicated in the DialStatus field.

Behavior Change

The **DialBegin/DialEnd** events replace the **Dial** event. Note that the **Dial** event signaling the end of dialing would not normally be sent until after bridging was complete; this operation will now occur when the dial operation has determined the status of a particular called channel.

Simple Successful Dial

Unknown macro: 'table'

Simple Failed Dial

Unknown macro: 'table'

Parallel Dial

Unknown macro: 'table'

Bridging

A bridge contains 0 or more channels. When a channel is in a bridge, it has the potential to communicate with other channels within the bridge. Before channels enter a bridge, a **BridgeCreate** event is sent, indicating that a bridge has been created. When a bridge is destroyed, a **BridgeDestroy** event is sent. All channels within a bridge **MUST** leave a bridge prior to the **BridgeDestroy** event being sent.

When a channel enters a bridge, a **BridgeEnter** event is raised. When a channel is put into a bridge, it is implied that the channel can pass media between other channels in the bridge. This is not guaranteed, as other properties on the channel or bridge may restrict media flow. For example, bridges with a BridgeTechnology type of *holding_bridge* implicitly restrict the media flow between channels. Likewise, media may be restricted in multi-party conference bridges based on user role permissions, such as when a conference leader mutes all participants in a conference. The **BridgeEnter** event does indicate, however, that a potential relationship between channels in a bridge exists.

When a channel leaves a bridge, a corresponding **BridgeLeave** event is raised. A **BridgeLeave** event **MUST** mean that the channel that left the bridge can no longer pass media to other channels still in the bridge. This does not necessarily mean that the channel is being hung up; rather, that it is no longer in a communication path with some other set of channels.

In all cases, if a channel has a **BridgeEnter** event, it **MUST** have a corresponding **BridgeLeave** event. If a channel is hung up and it is in a bridge, a **BridgeLeave** event **MUST** precede the **Hangup** event.

If a transfer operation is performed, a transfer event of some type **MUST** be raised for the channels involved in the transfer when the success or failure of the transfer is determined. Similarly, if a channel enters a parking lot, a **ParkedCall** event **MUST** be raised for the channel prior to it entering the bridge that represents the parking lot.

If a property of a bridge is changed, such as the BridgeTechnology changing from a simple two-party bridge to a multi-party bridge, then the **BridgeUpdate** event is sent with the updated parameters.

Two Party Bridging

Parties are bridged by virtue of them entering a bridge, as indicated by a **BridgeEnter**. When parties are no longer talking, a **BridgeLeave** event is sent for each channel that leaves the bridge.

Example - Two Party Bridge

Unknown macro: 'table'

In this scenario, it was perfectly acceptable for either Kermit or Gonzo's channels to continue after the bridge was broken. Since this represents the most basic two-party call scenario, once one party decided to hang up, the other party was also hung up on.

Transfers

Transfer information is conveyed with either a **BlindTransfer** or **AttendedTransfer** event, which indicates information about the transfer that took place. **BridgeLeave/BridgeEnter** events are used to indicate which channels are talking in which bridges at different stages during the transfer.


Transfers do a Lot

Depending on the type of transfer and the actions taken, channels will move in and out of a lot of bridges. The purpose of the two transfer events is to convey to the AMI client the overall completed status of the transfer after the users have completed their actions. With Blind Transfers, this typically happens very quickly: Asterisk simply has to determine that the destination of the transfer is a valid extension in the dialplan.

Attended transfers, on the other hand, can involve a lot more steps. Parties can consult, toggle back and forth between consultations, and merge bridges together. Channels can be transferred to a dialplan application directly, and not to another party! As such, Asterisk will send the **AttendedTransfer** event when it knows whether or not the Attended Transfer has completed successfully, and will attempt to convey as much information as possible about the final status of the transfer.

For more information on these events, see [BlindTransfer](#) and [AttendedTransfer](#).

Example - Blind Transfer

 Unknown macro: 'table'

Local Channel Optimization


Local channels have an option wherein they can be optimized away if both halves of a Local channel are in a bridge. This option is set on Local channel creation, and is communicated back to the AMI clients in the **LocalBridge** event in the LocalOptimization field. When a Local channel optimization occurs, a **LocalOptimizationBegin** event is sent that indicates the channels involved in the optimization. When the optimization has completed and the parties can now converse without the Local channel, a **LocalOptimizationEnd** event is sent.

Two actions can take place when a Local channel optimizes between two bridges.

1. If, after the Local channel optimization, either bridge contains only a single channel, then a single channel in one of the bridges is moved to the bridge that has the other channel. This is conveyed by a sequence of **BridgeLeave/BridgeEnter** events.
2. If, after the Local channel optimization, the bridges contain multiple parties, the bridges will be merged together. A **BridgeMerge** event is sent when this occurs. Channels will then be merged from one bridge to the other, denoted by a sequence of **BridgeLeave/BridgeEnter** events.

It is not defined which channel is moved first or which bridge wins during a bridge merge. That is an implementation detail left up to Asterisk. Suffice to say, if a Local channel is optimized away, Asterisk attempts to rebridge the channels left over as fast as possible to prevent any loss in audio.

Example - Optimizing Local Channel between two PJSIP Channels

 Unknown macro: 'table'

Masquerades

Masquerades are gone

In the past, masquerades occurred rather frequently - most often in any scenario where a transfer occurred or where a `pbx_thread` needed to be associated with a channel. This has now changed. Masquerades now rarely occur, and are never communicated to AMI clients. From the perspective of AMI clients, nothing changes - you still use your handle to a channel to communicate with it, regardless of the presence (or lack thereof) of a masquerade operation.

This section only exists to explicitly call out the fact that Masquerades are gone.

Transports

AMI supports the following transport mechanisms:

- TCP/TLS
- HTTP/HTTPS

When clients connect over HTTP/HTTPS, AMI events are queued up for retrieval. Events queued up for a client are automatically retrieved and sent in the response to any POST operation. The **WaitEvent** action can be used to wait for and retrieve AMI events.

Security Considerations

AMI supports security at the transport level via TLS using OpenSSL.

For specific security considerations and best practice, please see the [README-SERIOUSLY.bestpractices.txt](#) included with Asterisk.

Class Authorizations

Do not rely on class authorizations for security. While they provide a means to restrict a client's access to sets of functionality, there are often ways of achieving similar functionality through multiple mechanisms. Do **NOT** assume that because a class authorization has not been granted to a client, that they can't find a way around it. In general, view class authorizations as a coarse grained way of providing sets of filters.

Events and actions are automatically classified with particular class authorizations. Clients can be configured to support some set of class authorizations, filtering the actions that they can perform and events that they receive. The supported class authorizations are listed below.

Class Type	Description
system	The item is associated with something that reports on the status of the system or manipulates the system in some fashion
call	The item is associated with calls, i.e., state changes in a call, etc.
log	The item is associated with the logging subsystem
verbose	The item is associated with verbose messages
command	The item is associated with execution of CLI commands through AMI
agent	The item is associated with Queue Agent manipulation
user	The item is associated with user defined events
config	The item is associated with manipulating the configuration of Asterisk
dtmf	The item is associated with DTMF manipulation
reporting	The item is associated with querying information about the state of the Asterisk system
cdr	The item is associated with CDR manipulation
dialplan	The item is associated with dialplan execution
originate	The item is associated with originating a channel

Class Type	Description
agi	The item is associated with AGI execution
cc	The item is associated with call completion
aoc	The item is associated with Advice of Charge
test	The item is associated with some test action
message	The item is associated with out of call messaging
security	The item is associated with a security event in Asterisk
all	The item has all class authorizations associated with it
none	The item has no class authorization associated with it

Access Control Lists

Access Control Lists can be used to filter connections based on address. If an attempt to connect from an unauthorized address is detected, the connection attempt will be rejected.

Authorization

Authorization can be provided via the **Login** action. If a client fails to provide a valid username/password, the connection attempt and any subsequent actions will be rejected. Events will not be sent until the client provides authorized credentials.

The actions that are excluded from successful login are:

- **Login**
- **Logoff**
- **Challenge**

AMI Configuration

AMI supports the following configuration options. Note that additional configurations MAY be specified; however, these configuration options are valid for Asterisk 12.

General Settings

Option	Type	Description	Default
enabled	Boolean	Enable AMI	no
webenabled	Boolean	Enable AMI over HTTP/HTTPS	no
port	Integer	The port AMI's TCP server will bind to	5038
bindaddr	IP Address	The address AMI's TCP server will bind to	0.0.0.0
tlsenable	Boolean	Enable TLS over TCP	no
tlsbindaddr	IP Address	The address AMI's TCP/TLS server will bind to	0.0.0.0:5039

Option	Type	Description	Default
tlscertfile	String	The full path to the TLS certificate to use	/tmp/asterisk.pem
tlsprivatekey	String	The full path to the private key. If no path is specified, <i>tlscertfile</i> will be used for the private key.	/tmp/private.pem
tlscipher	String	The string specifying which SSL ciphers to use. Valid SSL ciphers can be found at http://www.openssl.org/docs/apps/ciphers.html#CIPHER_STRINGS	
allowmultiplelogin	Boolean	Allow multiple logins for the same user. If set to no, multiple logins from the same user will be rejected.	Yes
timestampevents	Boolean	Add a Unix epoch *Timestamp* field to all AMI events	No
authlimit	Integer	The number of unauthenticated clients that can be connected at any time	

Client Settings

Note that the name of the client settings context is the username for the client connection.

Option	Type	Description	Default
secret	String	The password that must be provided by the client via the Login action.	
deny	ACL	An address/mask to deny in an ACL. This option may be present multiple times.	
permit	ACL	An address/mask to allow in an ACL. This option may be present multiple times.	
acl	String	A Named ACL to apply to the client.	
setvar	String	A channel variable key/value pair (using the nomenclature VARIABLE=value) that will be set on all channels originated from this client	
eventfilter	RegEx	This option may be present multiple times. This options allows clients to whitelist or blacklist events. A filter is assumed to be a whitelist unless preceeded by a '!'. Evaluation of the filters is as follows: <ul style="list-style-type: none"> • If no filters are configured all events are reported as normal. • If there are white filters only: implied black all filter processed first, then white filters. • If there are black filters only: implied white all filter processed first, then black filters. • If there are both white and black filters: implied black all filter processed first, then white filters, and lastly black filters. 	
read	String	A comma delineated list of the allowed class authorizations applied to events	all
write	String	A comma delineated list of the allowed class authorizations applied to actions	all

The item has all class authorizations associated with it