pystrix Documentation

Release 1.2.0

Neil Tallim

Jan 18, 2023

Contents

1 Example usage **3**
- 1.1 Asterisk Management Interface (AMI) . . . 3
- 1.2 Asterisk Gateway Interface (AGI) . . . 7
- 1.3 Fast Asterisk Gateway Interface (FastAGI) . . . 7

2 Asterisk Gateway Interface (AGI) **9**
- 2.1 Core . . . 9
- 2.2 Members . . . 17

3 Asterisk Management Interface (AMI) **21**
- 3.1 Actions . . . 21
- 3.2 Events . . . 40
- 3.3 Members . . . 60

Index **65**

pystrix's main design goal is to provide a consistent, easy-to-extend API for interacting with Asterisk. This documentation exists to prevent you from needing to go trawling through source code when all you really need to do is see a class's constructor and read about what it does.

CHAPTER 1

Example usage

The following sections contain a single, well-documented-but-minimal example of how to use pystrix for the associated Asterisk function.

1.1 Asterisk Management Interface (AMI)

A simple, if verbose, AMI implementation is provided below, demonstrating how to connect to Asterisk with MD5-based authentication, how to connect callback handlers for events, and how to send requests for information:

```
import time

import pystrix

#Just a few constants for logging in. Putting them directly into code is usually a␣
↪bad idea.
_HOST = 'localhost'
_USERNAME = 'admin'
_PASSWORD = 'wordpass'

class AMICore(object):
    """
    The class that will be used to hold the logic for this AMI session. You could␣
↪also just work
    with the `Manager` object directly, but this is probably a better approach for␣
↪most
    general-purpose applications.
    """
    _manager = None #The AMI conduit for communicating with the local Asterisk server
    _kill_flag = False #True when the core has shut down of its own accord

    def __init__(self):
        #The manager supports Python's native logging module and has optional␣
↪features; see its
```

(continues on next page)

(continued from previous page)

```
        #constructor's documentation for details.
        self._manager = pystrix.ami.Manager()

        #Before connecting to Asterisk, callback handlers should be registered to␣
↪avoid missing
        #any events.
        self._register_callbacks()

        try:
            #Attempt to connect to Asterisk
            self._manager.connect(_HOST)

            #The first thing to be done is to ask the Asterisk server for a challenge␣
↪token to
            #avoid sending the password in plain-text. This step is optional, however,
↪ and can
            #be bypassed by simply omitting the 'challenge' parameter in the Login␣
↪action.
            challenge_response = self._manager.send_action(pystrix.ami.core.
↪Challenge())
            #This command demonstrates the common case of constructing a request␣
↪action and
            #sending it to Asterisk to await a response.

            if challenge_response and challenge_response.success:
                #The response is either a named tuple or None, with the latter␣
↪occuring in case
                #the request timed out. Requests are blocking (expected to be near-
↪instant), but
                #thread-safe, so you can build complex threading logic if necessary.
                action = pystrix.ami.core.Login(
                 _USERNAME, _PASSWORD, challenge=challenge_response.result['Challenge
↪']
                )
                self._manager.send_action(action)
                #As with the Challenge action before, a Login action is assembled and␣
↪sent to
                #Asterisk, only in two steps this time, for readability.
                #The Login class has special response-processing logic attached to it␣
↪that
                #causes authentication failures to raise a ManagerAuthException error,
↪ caught
                #below. It will still return the same named tuple if you need to␣
↪extract
                #additional information upon success, however.
            else:
                self._kill_flag = True
                raise ConnectionError(
                 "Asterisk did not provide an MD5 challenge token" +
                 (challenge_response is None and ': timed out' or '')
                )
        except pystrix.ami.ManagerSocketError as e:
            self._kill_flag = True
            raise ConnectionError("Unable to connect to Asterisk server: %(error)s" %
↪{
             'error': str(e),
            })
```

(continues on next page)

(continued from previous page)

```
        except pystrix.ami.core.ManagerAuthError as reason:
            self._kill_flag = True
            raise ConnectionError("Unable to authenticate to Asterisk server:
↪%(reason)s" % {
             'reason': reason,
            })
        except pystrix.ami.ManagerError as reason:
            self._kill_flag = True
            raise ConnectionError("An unexpected Asterisk error occurred: %(reason)s"
↪% {
             'reason': reason,
            })

        #Start a thread to make is_connected() fail if Asterisk dies.
        #This is not done automatically because it disallows the possibility of␣
↪immediate
        #correction in applications that could gracefully replace their connection␣
↪upon receipt
        #of a `ManagerSocketError`.
        self._manager.monitor_connection()

    def _register_callbacks(self):
        #This sets up some event callbacks, so that interesting things, like calls␣
↪being
        #established or torn down, will be processed by your application's logic. Of␣
↪course,
        #since this is just an example, the same event will be registered using two␣
↪different
        #methods.

        #The event that will be registered is 'FullyBooted', sent by Asterisk␣
↪immediately after
        #connecting, to indicate that everything is online. What the following code␣
↪does is
        #register two different callback-handlers for this event using two different
        #match-methods: string comparison and class-match. String-matching and class-
↪resolution
        #are equal in performance, so choose whichever you think looks better.
        self._manager.register_callback('FullyBooted', self._handle_string_event)
        self._manager.register_callback(pystrix.ami.core_events.FullyBooted, self._
↪handle_class_event)
        #Now, when 'FullyBooted' is received, both handlers will be invoked in the␣
↪order in
        #which they were registered.

        #A catch-all-handler can be set using the empty string as a qualifier,␣
↪causing it to
        #receive every event emitted by Asterisk, which may be useful for debugging␣
↪purposes.
        self._manager.register_callback('', self._handle_event)

        #Additionally, an orphan-handler may be provided using the special qualifier␣
↪None,
        #causing any responses not associated with a request to be received. This␣
↪should only
        #apply to glitches in pre-production versions of Asterisk or requests that␣
↪timed out
```

(continues on next page)

(continued from previous page)

```
        #while waiting for a response, which is also indicative of glitchy behaviour. This
        #handler could be used to process the orphaned response in special cases, but is likely
        #best relegated to a logging role.
        self._manager.register_callback(None, self._handle_event)

        #And here's another example of a registered event, this time catching Asterisk's
        #Shutdown signal, emitted when the system is shutting down.
        self._manager.register_callback('Shutdown', self._handle_shutdown)

    def _handle_shutdown(self, event, manager):
        self._kill_flag = True

    def _handle_event(self, event, manager):
        print("Received event: %s" % event.name)

    def _handle_string_event(self, event, manager):
        print("Received string event: %s" % event.name)

    def _handle_class_event(self, event, manager):
        print("Received class event: %s" % event.name)

    def is_alive(self):
        return not self._kill_flag

    def kill(self):
        self._manager.close()


class Error(Exception):
    """
    The base class from which all exceptions native to this module inherit.
    """

class ConnectionError(Error):
    """
    Indicates that a problem occurred while connecting to the Asterisk server
    or that the connection was severed unexpectedly.
    """

if __name__ == '__main__':
    ami_core = AMICore()

    while ami_core.is_alive():
        #In a larger application, you'd probably do something useful in another non-daemon
        #thread or maybe run a parallel FastAGI server. The pystrix implementation has the AMI
        #threads run daemonically, however, so a block like this in the main thread is necessary
        time.sleep(1)
    ami_core.kill()
```

1.2 Asterisk Gateway Interface (AGI)

A simple AGI implementation is provided below, demonstrating how to handle requests from Asterisk, like, as illustrated, answering a call, playing a message, and hanging up:

```
#!/usr/bin/env python
import pystrix

if __name__ == '__main__':
    agi = pystrix.agi.AGI()

    agi.execute(pystrix.agi.core.Answer()) #Answer the call

    response = agi.execute(pystrix.agi.core.StreamFile('demo-thanks', escape_digits=(
↪'1', '2'))) #Play a file; allow DTMF '1' or '2' to interrupt
    if response: #Playback was interrupted; if you don't care, you don't need to␣
↪catch this
        (dtmf_character, offset) = response #The key pressed by the user and the␣
↪playback time

    agi.execute(pystrix.agi.core.Hangup()) #Hang up the call
```

1.3 Fast Asterisk Gateway Interface (FastAGI)

A simple FastAGI implementation is provided below, demonstrating how to listen for and handle requests from Asterisk, like, as illustrated, answering a call, playing a message, and hanging up:

```
import re
import threading
import time

import pystrix

class FastAGIServer(threading.Thread):
    """
    A simple thread that runs a FastAGI server forever.
    """
    _fagi_server = None #The FastAGI server controlled by this thread

    def __init__(self):
        threading.Thread.__init__(self)
        self.daemon = True

        self._fagi_server = pystrix.agi.FastAGIServer()

        self._fagi_server.register_script_handler(re.compile('demo'), self._demo_
↪handler)
        self._fagi_server.register_script_handler(None, self._noop_handler)

    def _demo_handler(self, agi, args, kwargs, match, path):
        """
        `agi` is the AGI instance used to process events related to the channel,␣
↪`args` is a
        collection of positional arguments provided with the script as a tuple,␣
↪`kwargs` is a
```

(continues on next page)

(continued from previous page)

```
        dictionary of keyword arguments supplied with the script (values are␣
↪enumerated in a list),
        `match` is the regex match object (None if the fallback handler), and `path`␣
↪is the string
        path supplied by Asterisk, in case special processing is needed.

        The directives issued in this function can all raise Hangup exceptions, which␣
↪should be
        caught if doing anything complex, but an uncaught exception will simply cause␣
↪a warning to
        be raised, making AGI scripts very easy to write.
        """
        agi.execute(pystrix.agi.core.Answer()) #Answer the call

        response = agi.execute(pystrix.agi.core.StreamFile('demo-thanks', escape_
↪digits=('1', '2'))) #Play a file; allow DTMF '1' or '2' to interrupt
        if response: #Playback was interrupted; if you don't care, you don't need to␣
↪catch this
            (dtmf_character, offset) = response #The key pressed by the user and the␣
↪playback time

        agi.execute(pystrix.agi.core.Hangup()) #Hang up the call

    def _noop_handler(self, agi, args, kwargs, match, path):
        """
        Does nothing, causing control to return to Asterisk's dialplan immediately;␣
↪provided just
        to demonstrate the fallback handler.
        """

    def kill(self):
        self._fagi_server.shutdown()

    def run(self):
        self._fagi_server.serve_forever()



if __name__ == '__main__':
    fastagi_core = FastAGIServer()
    fastagi_core.start()

    while fastagi_core.is_alive():
        #In a larger application, you'd probably do something useful in another non-
↪daemon
        #thread or maybe run a parallel AMI server
        time.sleep(1)
    fastagi_core.kill()
```

CHAPTER 2

Asterisk Gateway Interface (AGI)

The AGI interface consists of a number of action classes that are sent to Asterisk to effect actions on active channels. Two different means of getting access to a channel are defined: AGI and FastAGI, with the difference between them being that every AGI instance runs as a child process of Asterisk (full Python interpreter and everything), while FastAGI runs over a TCP/IP socket, allowing for faster startup times and lower overhead, with the cost of a little more development investment.

pystrix exposes the same feature-set and interaction model for both AGI and FastAGI, allowing any of the actions defined in the following sections to be instantiated and passed (any number of times) to `agi.AGI.execute()`.

2.1 Core

By default, Asterisk exposes a number of ways to interact with a channel, all of which are described below.

2.1.1 Members

All of the following objects should be accessed as part of the *agi.core* namespace, regardless of the modules in which they are defined.

Constants

CHANNEL_DOWN_AVAILABLE
: Channel is down and available

CHANNEL_DOWN_RESERVED
: Channel is down and reserved

CHANNEL_OFFHOOK
: Channel is off-hook

CHANNEL_DIALED
: A destination address has been specified

CHANNEL_ALERTING
The channel is locally ringing

CHANNEL_REMOTE_ALERTING
The channel is remotely ringing

CHANNEL_UP
The channel is connected

CHANNEL_BUSY
The channel is in a busy, non-conductive state

FORMAT_SLN
Selects the *sln* audio format

FORMAT_G723
Selects the *g723* audio format

FORMAT_G729
Selects the *g729* audio format

FORMAT_GSM
Selects the *gsm* audio format

FORMAT_ALAW
Selects the *alaw* audio format

FORMAT_ULAW
Selects the *ulaw* audio format

FORMAT_VOX
Selects the *vox* audio format

FORMAT_WAV
Selects the *wav* audio format

LOG_DEBUG
The Asterisk logging level equivalent to 'debug'

LOG_INFO
The Asterisk logging level equivalent to 'info'

LOG_WARN
The Asterisk logging level equivalent to 'warn'

LOG_ERROR
The Asterisk logging level equivalent to 'error'

LOG_CRITICAL
The Asterisk logging level equivalent to 'critical'

TDD_ON
Sets TDD to on

TDD_OFF
Sets TDD to off

TDD_MATE
Sets TDD to mate

Actions

class agi.core.**Answer**
Answers the call on the channel.

If the channel has already been answered, this is a no-op.

AGIAppError is raised on failure, most commonly because the connection could not be established.

class agi.core.**ChannelStatus**(*channel=None*)
Provides the current state of this channel or, if *channel* is set, that of the named channel.

Returns one of the channel-state constants listed below:

- CHANNEL_DOWN_AVAILABLE: Channel is down and available
- CHANNEL_DOWN_RESERVED: Channel is down and reserved
- CHANNEL_OFFHOOK: Channel is off-hook
- CHANNEL_DIALED: A destination address has been specified
- CHANNEL_ALERTING: The channel is locally ringing
- CHANNEL_REMOTE_ALERTING: The channel is remotely ringing
- CHANNEL_UP: The channel is connected
- CHANNEL_BUSY: The channel is in a busy, non-conductive state

The value returned is an integer in the range 0-7; values outside of that range were undefined at the time of writing, but will be returned verbatim. Applications unprepared to handle unknown states should raise an exception upon their receipt or otherwise handle the code gracefully.

AGIAppError is raised on failure, most commonly because the channel is in a hung-up state.

class agi.core.**ControlStreamFile**(*filename*, *escape_digits=”*, *sample_offset=0*, *forward=”*, *rewind=”*, *pause=”*)
See also *GetData*, *GetOption*, *StreamFile*.

Plays back the specified file, which is the *filename* of the file to be played, either in an Asterisk-searched directory or as an absolute path, without extension. (‘myfile.wav’ would be specified as ‘myfile’, to allow Asterisk to choose the most efficient encoding, based on extension, for the channel)

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received.

sample_offset may be used to jump an arbitrary number of milliseconds into the audio data.

If specified, *forward*, *rewind*, and *pause* are DTMF characters that will seek forwards and backwards in the audio stream or pause it temporarily; by default, these features are disabled.

If a DTMF key is received, it is returned as a string. If nothing is received or the file could not be played back (see Asterisk logs), None is returned.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

class agi.core.**DatabaseDel**(*family*, *key*)
Deletes the specified family/key entry from Asterisk’s database.

AGIAppError is raised on failure.

AGIDBError is raised if the key could not be removed, which usually indicates that it didn’t exist in the first place.

class agi.core.**DatabaseDeltree**(*family*, *keytree=None*)
Deletes the specificed family (and optionally keytree) from Asterisk's database.

AGIAppError is raised on failure.

AGIDBError is raised if the family (or keytree) could not be removed, which usually indicates that it didn't exist in the first place.

class agi.core.**DatabaseGet**(*family*, *key*)
Retrieves the value of the specified family/key entry from Asterisk's database.

AGIAppError is raised on failure.

AGIDBError is raised if the key could not be found or if some other database problem occurs.

class agi.core.**DatabasePut**(*family*, *key*, *value*)
Inserts or updates value of the specified family/key entry in Asterisk's database.

AGIAppError is raised on failure.

AGIDBError is raised if the key could not be inserted or if some other database problem occurs.

class agi.core.**Exec**(*application*, *options=()*)
Executes an arbitrary Asterisk *application* with the given *options*, returning that application's output.

options is an optional sequence of arguments, with any double-quote characters or commas explicitly escaped.

AGIAppError is raised if the application could not be executed.

class agi.core.**GetData**(*filename*, *timeout=2000*, *max_digits=255*)
See also *ControlStreamFile*, *GetOption*, *StreamFile*.

Plays back the specified file, which is the *filename* of the file to be played, either in an Asterisk-searched directory or as an absolute path, without extension. ('myfile.wav' would be specified as 'myfile', to allow Asterisk to choose the most efficient encoding, based on extension, for the channel)

timeout is the number of milliseconds to wait between DTMF presses or following the end of playback if no keys were pressed to interrupt playback prior to that point. It defaults to 2000.

max_digits is the number of DTMF keypresses that will be accepted. It defaults to 255.

The value returned is a tuple consisting of (dtmf_keys:str, timeout:bool). '#' is always interpreted as an end-of-event character and will never be present in the output.

AGIAppError is raised on failure, most commonly because no keys, aside from '#', were entered.

class agi.core.**GetFullVariable**(*variable*)
Returns a *variable* associated with this channel, with full expression-processing.

The value of the requested variable is returned as a string. If the variable is undefined, *None* is returned.

AGIAppError is raised on failure.

class agi.core.**GetOption**(*filename*, *escape_digits=''*, *timeout=2000*)
See also *ControlStreamFile*, *GetData*, *StreamFile*.

Plays back the specified file, which is the *filename* of the file to be played, either in an Asterisk-searched directory or as an absolute path, without extension. ('myfile.wav' would be specified as 'myfile', to allow Asterisk to choose the most efficient encoding, based on extension, for the channel)

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received.

timeout is the number of milliseconds to wait following the end of playback if no keys were pressed to interrupt playback prior to that point. It defaults to 2000.

The value returned is a tuple consisting of (dtmf_key:str, offset:int), where the offset is the number of milliseconds that elapsed since the start of playback, or None if playback completed successfully or the sample could not be opened.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

class agi.core.**GetVariable**(*variable*)
Returns a *variable* associated with this channel.

The value of the requested variable is returned as a string. If the variable is undefined, *None* is returned.

AGIAppError is raised on failure.

class agi.core.**Hangup**(*channel=None*)
Hangs up this channel or, if *channel* is set, the named channel.

AGIAppError is raised on failure.

class agi.core.**Noop**
Does nothing.

Good for testing the connection to the Asterisk server, like a ping, but not useful for much else. If you wish to log information through Asterisk, use the *verbose* method instead.

AGIAppError is raised on failure.

class agi.core.**ReceiveChar**(*timeout=0*)
Receives a single character of text from a supporting channel, discarding anything else in the character buffer.

timeout is the number of milliseconds to wait for a character to be received, defaulting to infinite.

The value returned is a tuple of the form (char:str, timeout:bool), with the timeout element indicating whether the function returned because of a timeout, which may result in an empty string. *None* is returned if the channel does not support text.

AGIAppError is raised on failure.

class agi.core.**ReceiveText**(*timeout=0*)
Receives a block of text from a supporting channel.

timeout is the number of milliseconds to wait for text to be received, defaulting to infinite. Presumably, the first block received is immediately returned in full.

The value returned is a string.

AGIAppError is raised on failure.

class agi.core.**RecordFile**(*filename*, *format='wav'*, *escape_digits=''*, *timeout=-1*, *sample_offset=0*, *beep=True*, *silence=None*)
Records audio to the specified file, which is the *filename* of the file to be written, defaulting to Asterisk's sounds path or an absolute path, without extension. ('myfile.wav' would be specified as 'myfile') *format* is one of the following, which sets the extension and encoding, with WAV being the default:

- FORMAT_SLN
- FORMAT_G723
- FORMAT_G729
- FORMAT_GSM
- FORMAT_ALAW
- FORMAT_ULAW
- FORMAT_VOX

- FORMAT_WAV: PCM16

The filename may also contain the special string '%d', which Asterisk will replace with an auto-incrementing number, with the resulting filename appearing in the 'RECORDED_FILE' channel variable.

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received.

timeout is the number of milliseconds to wait following the end of playback if no keys were pressed to end recording prior to that point. By default, it waits forever.

sample_offset may be used to jump an arbitrary number of milliseconds into the audio data.

beep, if *True*, the default, causes an audible beep to be heard when recording begins.

silence, if given, is the number of seconds of silence to allow before terminating recording early.

The value returned is a tuple consisting of (dtmf_key:str, offset:int, timeout:bool), where the offset is the number of milliseconds that elapsed since the start of playback dtmf_key may be the empty string if no key was pressed, and timeout is *False* if recording ended due to another condition (DTMF or silence).

The raising of *AGIResultHangup* is another condition that signals a successful recording, though it also means the user hung up.

AGIAppError is raised on failure, most commonly because the destination file isn't writable.

class agi.core.**SayAlpha**(*characters*, *escape_digits=''*)
Reads an alphabetic string of *characters*.

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received and it is returned. If nothing is recieved, *None* is returned.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

class agi.core.**SayDate**(*seconds=None*, *escape_digits=''*)
Reads the date associated with *seconds* since the UNIX Epoch. If not given, the local time is used.

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received and it is returned. If nothing is recieved, *None* is returned.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

class agi.core.**SayDatetime**(*seconds=None*, *escape_digits=''*, *format=None*, *timezone=None*)
Reads the datetime associated with *seconds* since the UNIX Epoch. If not given, the local time is used.

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received and it is returned. If nothing is recieved, *None* is returned.

format defaults to *"ABdY 'digits/at' IMp"*, but may be a string with any of the following meta-characters (or single-quote-escaped sound-file references):

- A: Day of the week
- B: Month (Full Text)
- m: Month (Numeric)
- d: Day of the month
- Y: Year
- I: Hour (12-hour format)

- H: Hour (24-hour format)
- M: Minutes
- p: AM/PM
- Q: Shorthand for Today, Yesterday or ABdY
- R: Shorthand for HM
- S: Seconds
- T: Timezone

timezone may be a string in standard UNIX form, like 'America/Edmonton'. If *format* is undefined, *timezone* is ignored and left to default to the system's local value.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

class agi.core.**SayDigits**(*digits*, *escape_digits=''*)
Reads a numeric string of *digits*.

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received and it is returned. If nothing is recieved, *None* is returned.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

class agi.core.**SayNumber**(*number*, *escape_digits=''*)
Reads a *number* naturally.

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received and it is returned. If nothing is recieved, *None* is returned.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

class agi.core.**SayPhonetic**(*characters*, *escape_digits=''*)
Reads a phonetic string of *characters*.

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received and it is returned. If nothing is received, *None* is returned.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

class agi.core.**SayTime**(*seconds=None*, *escape_digits=''*)
Reads the time associated with *seconds* since the UNIX Epoch. If not given, the local time is used.

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received and it is returned. If nothing is received, *None* is returned.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

class agi.core.**SendImage**(*filename*)
Sends the specified image, which is the *filename* of the file to be presented, either in an Asterisk-searched directory or as an absolute path, without extension. ('myfile.png' would be specified as 'myfile', to allow Asterisk to choose the most efficient encoding, based on extension, for the channel)

AGIAppError is raised on failure.

class agi.core.**SendText**(*text*)
Sends the specified *text* on a supporting channel.

AGIAppError is raised on failure.

class agi.core.**SetAutohangup**(*seconds=0*)
Instructs Asterisk to hang up the channel after the given number of *seconds* have elapsed.

Calling this function with *seconds* set to 0, the default, will disable auto-hangup.

AGIAppError is raised on failure.

class agi.core.**SetCallerid**(*number*, *name=None*)
Sets the called ID (*number* and, optionally, *name*) of Asterisk on this channel.

AGIAppError is raised on failure.

class agi.core.**SetContext**(*context*)
Sets the context for Asterisk to use upon completion of this AGI instance.

No context-validation is performed; specifying an invalid context will cause the call to terminate unexpectedly.

AGIAppError is raised on failure.

class agi.core.**SetExtension**(*extension*)
Sets the extension for Asterisk to use upon completion of this AGI instance.

No extension-validation is performed; specifying an invalid extension will cause the call to terminate unexpectedly.

AGIAppError is raised on failure.

class agi.core.**SetMusic**(*on*, *moh_class=None*)
Enables or disables music-on-hold for this channel, per the state of the *on* argument.

If specified, *moh_class* identifies the music-on-hold class to be used.

AGIAppError is raised on failure.

class agi.core.**SetPriority**(*priority*)
Sets the priority for Asterisk to use upon completion of this AGI instance.

No priority-validation is performed; specifying an invalid priority will cause the call to terminate unexpectedly.

AGIAppError is raised on failure.

class agi.core.**SetVariable**(*name*, *value*)
Sets the variable identified by *name* to *value* on the current channel.

AGIAppError is raised on failure.

class agi.core.**StreamFile**(*filename*, *escape_digits=''*, *sample_offset=0*)
See also *ControlStreamFile*, *GetData*, *GetOption*.

Plays back the specified file, which is the *filename* of the file to be played, either in an Asterisk-searched directory or as an absolute path, without extension. ('myfile.wav' would be specified as 'myfile', to allow Asterisk to choose the most efficient encoding, based on extension, for the channel)

escape_digits may optionally be a list of DTMF digits, specified as a string or a sequence of possibly mixed ints and strings. Playback ends immediately when one is received.

sample_offset may be used to jump an arbitrary number of milliseconds into the audio data.

The value returned is a tuple consisting of (dtmf_key:str, offset:int), where the offset is the number of milliseconds that elapsed since the start of playback, or None if playback completed successfully or the sample could not be opened.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

class agi.core.**TDDMode**(*mode*)
Sets the TDD transmission *mode* on supporting channels, one of the following:

- TDD_ON
- TDD_OFF
- TDD_MATE

True is returned if the mode is set, *False* if the channel isn't capable, and *AGIAppError* is raised if a problem occurs. According to documentation from 2006, all non-capable channels will cause an exception to occur.

class agi.core.**Verbose**(*message*, *level=1*)
Causes Asterisk to process *message*, logging it to console or disk, depending on whether *level* is greater-than-or-equal-to Asterisk's corresponding verbosity threshold.

level is one of the following, defaulting to LOG_INFO:

- LOG_DEBUG
- LOG_INFO
- LOG_WARN
- LOG_ERROR
- LOG_CRITICAL

AGIAppError is raised on failure.

class agi.core.**WaitForDigit**(*timeout=-1*)
Waits for up to *timeout* milliseconds for a DTMF keypress to be received, returning that value. By default, this function blocks indefinitely.

If no DTMF key is pressed, *None* is returned.

AGIAppError is raised on failure, most commonly because the channel was hung-up.

Exceptions

exception agi.core.**AGIDBError**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIAppError

Indicates that Asterisk encountered an error while interactive with its database.

2.2 Members

All of the following objects should be accessed as part of the *agi* namespace, regardless of the modules in which they are defined.

2.2.1 Classes

class agi.**AGI**(*debug=False*)
An interface to Asterisk, exposing request-response functions for synchronous management of the call associated with this channel.

execute(*action*)
Sends a request to Asterisk and waits for a response before returning control to the caller.

The given *_Action* object, *action*, carries the command, arguments, and result-processing logic used to communicate with Asterisk.

The state of the channel is verified with each call to this function, to ensure that it is still connected. An instance of *AGIHangup* is raised if it is not.

get_environment()
Returns Asterisk's initial environment variables as a dictionary.

Note that this function returns a copy of the values, so repeated calls are less favourable than storing the returned value locally and dissecting it there.

class agi.**FastAGIServer**(*interface='127.0.0.1'*, *port=4573*, *daemon_threads=True*, *debug=False*)
Provides a FastAGI TCP server to handle requests from Asterisk servers.

timeout
The number of seconds to wait for a request when using *handle_request()*. Has no effect on *serve_forever()*.

handle_request()
Handles at most one request in a separate thread or times out and returns control silently.

serve_forever()
Continues to serve requests as they are received, handling each in a new thread, until *shutdown()* is called.

shutdown()
Interrupts *serve_forever()* gracefully.

clear_script_handlers()
Empties the list of script handlers, allowing it to be repopulated. The default handler is not cleared by this action; to clear it, call *register_script_handler(None, None)*.

get_script_handler(*script_path*)
Provides the callable specified to handle requests received by this FastAGI server and the result of matching, as a tuple.

script_path is the path received from Asterisk.

register_script_handler(*regex*, *handler*)
Registers the given *handler*, which is a callable that accepts an AGI object used to communicate with the Asterisk channel, a tuple containing any positional arguments, a dictionary containing any keyword arguments (values are enumerated in a list), the match object (may be *None*), and the original script address as a string.

Handlers are resolved by *regex*, which may be a regular expression object or a string, match in the order in which they were supplied, so provide more specific qualifiers first.

The special *regex* value *None* sets the default handler, invoked when every comparison fails; this is preferable to adding a catch-all handler in case the list is changed at runtime. Setting the default handler to *None* disables the catch-all, which will typically make Asterisk just drop the call.

unregister_script_handler(*regex*)
Removes a specific script-handler from the list, given the same *regex* object used to register it initially.

This function should only be used when a specific handler is no longer useful; if you want to re-introduce handlers, consider using *clear_script_handlers()* and re-adding all handlers in the desired order.

2.2.2 Exceptions

exception agi.**AGIException**(*message*, *items=None*)
Bases: exceptions.Exception

The base exception from which all exceptions native to this module inherit.

items
A dictionary containing any key-value items received from Asterisk to explain the exception.

exception agi.**AGIError**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIException

The base error from which all errors native to this module inherit.

exception agi.**AGIUnknownError**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIError

An error raised when an unknown response is received from Asterisk.

exception agi.**AGIAppError**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIError

An error raised when an attempt to make use of an Asterisk application fails.

exception agi.**AGIHangup**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIException

The base exception used to indicate that the call has been completed or abandoned.

exception agi.**AGISIGPIPEHangup**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIHangup

Indicates that the communications pipe to Asterisk has been severed.

exception agi.**AGISIGHUPHangup**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIHangup

Indicates that the script's process received the SIGHUP signal, implying Asterisk has hung up the call. Specific to script-based instances.

exception agi.**AGIResultHangup**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIHangup

Indicates that Asterisk received a clean hangup event.

exception agi.**AGIDeadChannelError**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIError

Indicates that a command was issued on a channel that can no longer process it.

exception agi.**AGIUsageError**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIError

Indicates that a request made to Asterisk was sent with invalid syntax.

exception agi.**AGIInvalidCommandError**(*message*, *items=None*)
Bases: pystrix.agi.agi_core.AGIError

Indicates that a request made to Asterisk was not understood.

CHAPTER 3

Asterisk Management Interface (AMI)

The AMI interface consists primarily of a number of action classes that are sent to Asterisk to ellicit responses. Additionally, a number of event classes are defined to provide convenience processing on the various messages Asterisk generates.

3.1 Actions

The AMI reacts primarily to requests submitted to it in the form of actions, which are described in this section. Their usage is consistently a matter of instantiating a class, then passing it (as many times as you'd like) to *ami.Manager.send_action()*

3.1.1 Core

Asterisk provides a rich collection of features by default, the standard set of which are described here.

Members

All of the following objects should be accessed as part of the *ami.core* namespace, regardless of the modules in which they are defined.

Constants

AUTHTYPE_MD5
Uses MD5 authentication when logging into AMI

EVENTMASK_ALL
Turns on all events with the *ami.core.Events* action

EVENTMASK_NONE
Turns off all events with the *ami.core.Events* action

EVENTMASK_CALL
Turns on call events with the *ami.core.Events* action

EVENTMASK_LOG
Turns on log events with the *ami.core.Events* action

EVENTMASK_SYSTEM
Turns on system events with the *ami.core.Events* action

FORMAT_SLN
Selects the *sln* audio format

FORMAT_G723
Selects the *g723* audio format

FORMAT_G729
Selects the *g729* audio format

FORMAT_GSM
Selects the *gsm* audio format

FORMAT_ALAW
Selects the *alaw* audio format

FORMAT_ULAW
Selects the *ulaw* audio format

FORMAT_VOX
Selects the *vox* audio format

FORMAT_WAV
Selects the *wav* audio format

ORIGINATE_RESULT_REJECT
Remote extension rejected (hung up) without answering

ORIGINATE_RESULT_RING_LOCAL
Local extension rang, but didn't answer

ORIGINATE_RESULT_RING_REMOTE
Remote extension rang, but didn't answer

ORIGINATE_RESULT_ANSWERED
Remote extension answered

ORIGINATE_RESULT_BUSY
Remote extension was busy

ORIGINATE_RESULT_CONGESTION
Remote extension was unreachable

ORIGINATE_RESULT_INCOMPLETE
Remote extension could not be identified

Actions

class ami.core.**AbsoluteTimeout**(*channel*, *seconds=0*)
Bases: pystrix.ami.ami._Request

Causes Asterisk to hang up a channel after a given number of seconds.

Requires call

__init__(*channel*, *seconds=0*)
Causes the call on *channel* to be hung up after *seconds* have elapsed, defaulting to disabling auto-hangup.

class ami.core.**AGI**(*channel*, *command*, *command_id=None*)
Bases: pystrix.ami.ami._Request

Causes Asterisk to execute an arbitrary AGI application in a call.

Upon successful execution, an 'AsyncAGI' event is generated.

Requires call

__init__(*channel*, *command*, *command_id=None*)
channel is the call in which to execute *command*, the value passed to the AGI dialplan application. *command_id* is an optional value that will be present in the resulting event, and can reasonably be set to a sequential digit or UUID in your application for tracking purposes.

class ami.core.**Bridge**(*channel_1*, *channel_2*, *tone=False*)
Bases: pystrix.ami.ami._Request

Bridges two channels already connected to Asterisk.

Requires call

__init__(*channel_1*, *channel_2*, *tone=False*)
channel_1 is the channel to which *channel_2* will be connected. *tone*, if *True*, will cause a sound to be played on *channel_2*.

class ami.core.**Challenge**(*authtype='MD5'*)
Bases: pystrix.ami.ami._Request

Asks the AMI server for a challenge token to be used to hash the login secret.

The value provided under the returned response's 'Challenge' key must be passed as the 'challenge' parameter of the *Login* object's constructor:

```
login = Login(username='me', secret='password', challenge=response.get('Challenge
↪'))
```

__init__(*authtype='MD5'*)
authtype is used to specify the authentication type to be used.

class ami.core.**ChangeMonitor**(*channel*, *filename*)
Bases: pystrix.ami.ami._Request

Changes the filename associated with the recording of a monitored channel. The channel must have previously been selected by the *Monitor* action.

Requires call

__init__(*channel*, *filename*)
channel is the channel to be affected and *filename* is the new target filename, without extension, as either an auto-resolved or absolute path.

class ami.core.**Command**(*command*)
Bases: pystrix.ami.ami._Request

Sends an arbitrary shell command to Asterisk, returning its response as a series of lines in the 'data' attribute.

Requires command

__init__(*command*)
command is the command to be executed.

class ami.core.**CoreShowChannels**
Bases: pystrix.ami.ami._Request

Asks Asterisk to list all active channels.

Any number of 'CoreShowChannel' events may be generated in response to this request, followed by one 'CoreShowChannelsComplete'.

Requires system

__init__()
action is the type of action being requested of the Asterisk server.

class ami.core.**CreateConfig**(*filename*)
Bases: pystrix.ami.ami._Request

Creates an empty configuration file, intended for use before *UpdateConfig()*.

Requires config

__init__(*filename*)
filename is the name of the file, with extension, to be created.

class ami.core.**DBDel**(*family*, *key*)
Bases: pystrix.ami.ami._Request

Deletes a database value from Asterisk.

Requires system

__init__(*family*, *key*)
family and *key* are specifiers to select the value to remove.

class ami.core.**DBDelTree**(*family*, *key=None*)
Bases: pystrix.ami.ami._Request

Deletes a database tree from Asterisk.

Requires system

__init__(*family*, *key=None*)
family and *key* (optional) are specifiers to select the values to remove.

class ami.core.**DBGet**(*family*, *key*)
Bases: pystrix.ami.ami._Request

Requests a database value from Asterisk.

A 'DBGetResponse' event will be generated upon success.

Requires system

__init__(*family*, *key*)
family and *key* are specifiers to select the value to retrieve.

class ami.core.**DBPut**(*family*, *key*, *value*)
Bases: pystrix.ami.ami._Request

Stores a database value in Asterisk.

Requires system

__init__(*family*, *key*, *value*)
family and *key* are specifiers for where to place *value*.

class ami.core.**Events**(*mask*)
Bases: pystrix.ami.ami._Request

Changes the types of unsolicited events Asterisk sends to this manager connection.

__init__(*mask*)
Mask is one of the following...

- EVENTMASK_ALL
- EVENTMASK_NONE

...or an iterable, like a tuple, with any combination of the following...

- EVENTMASK_CALL
- EVENTMASK_LOG
- EVENTMASK_SYSTEM

If an empty value is provided, EVENTMASK_NONE is assumed.

class ami.core.**ExtensionState**(*extension*, *context*)
Bases: pystrix.ami.ami._Request

Provides the state of an extension.

If successful, a 'Status' key will be present, with one of the following values as a string:

- -2: Extension removed
- -1: Extension hint not found
- 0: Idle
- 1: In use
- 2: Busy

If non-negative, a 'Hint' key will be present, too, containing string data that can be helpful in discerning the current activity of the device.

Requires call

__init__(*extension*, *context*)
extension is the extension to be checked and *context* is the container in which it resides.

class ami.core.**GetConfig**(*filename*)
Bases: pystrix.ami.ami._Request

Gets the contents of an Asterisk configuration file.

The result is recturned as a series of 'Line-XXXXXX-XXXXXX' keys that increment from 0 sequentially, starting with 'Line-000000-000000'.

A sequential generator is provided by the 'get_lines()' function on the response.

Requires config

get_lines()
Provides a generator that yields every line in order.

__init__(*filename*)
filename is the name of the config file to be read, including extension.

class ami.core.**Getvar**(*variable*, *channel=None*)
Bases: pystrix.ami.ami._Request

Gets the value of a channel or global variable from Asterisk, returning the result under the 'Value' key.

Requires call

__init__(*variable*, *channel=None*)
variable is the name of the variable to retrieve. *channel* is optional; if not specified, a global variable is retrieved.

class ami.core.**Hangup**(*channel*)
Bases: pystrix.ami.ami._Request

Hangs up a channel.

On success, a 'Hangup' event is generated.

Requires call

__init__(*channel*)
channel is the ID of the channel to be hung up.

class ami.core.**ListCommands**
Bases: pystrix.ami.ami._Request

Provides a list of every command exposed by the Asterisk Management Interface, with synopsis, as a series of lines in the response's 'data' attribute.

__init__()
action is the type of action being requested of the Asterisk server.

class ami.core.**ListCategories**(*filename*)
Bases: pystrix.ami.ami._Request

Provides a list of every category in an Asterisk configuration file, as a series of lines in the response's 'data' attribute.

Requires config

__init__(*filename*)
filename is the name of the file, with extension, to be read.

class ami.core.**LocalOptimizeAway**(*channel*)
Bases: pystrix.ami.ami._Request

Allows a bridged channel to be optimised in Asterisk's processing logic. This function should only be invoked after explicitly bridging.

Requires call

__init__(*channel*)
channel is the channel to be optimised.

class ami.core.**Login**(*username*, *secret*, *events=True*, *challenge=None*, *authtype='MD5'*)
Bases: pystrix.ami.ami._Request

Authenticates to the AMI server.

__init__(*username*, *secret*, *events=True*, *challenge=None*, *authtype='MD5'*)
username and *secret* are the credentials used to authenticate.

events may be set to *False* to prevent unsolicited events from being received. This is normally not desireable, so leaving it *True* is usually a good idea.

If given, *challenge* is a challenge string provided by Asterisk after sending a *Challenge* action, used with *authtype* to determine how to authenticate. *authtype* is ignored if the *challenge* parameter is unset.

class ami.core.**Logoff**
Bases: pystrix.ami.ami._Request

Logs out of the current manager session, permitting reauthentication.

__init__()
action is the type of action being requested of the Asterisk server.

class ami.core.**ModuleLoad**(*load_type*, *module=None*)
Bases: pystrix.ami.ami._Request

Loads, unloads, or reloads modules.

Requires system

__init__(*load_type*, *module=None*)
load_type is one of the following:

- 'load'
- 'unload'
- 'reload': if *module* is undefined, all modules are reloaded

module is optionally the name of the module, with extension, or one of the following for a built-in subsystem:

- 'cdr'
- 'dnsmgr'
- 'enum'
- 'extconfig'
- 'http'
- 'manager'
- 'rtp'

class ami.core.**Monitor**(*channel*, *filename*, *format='wav'*, *mix=True*)
Bases: pystrix.ami.ami._Request

Starts monitoring (recording) a channel.

Requires call

__init__(*channel*, *filename*, *format='wav'*, *mix=True*)
channel is the channel to be affected and *filename* is the new target filename, without extension, as either an auto-resolved or absolute path.

format may be any format Asterisk understands, defaulting to FORMAT_WAV:

- FORMAT_SLN
- FORMAT_G723
- FORMAT_G729
- FORMAT_GSM
- FORMAT_ALAW
- FORMAT_ULAW
- FORMAT_VOX
- FORMAT_WAV: PCM16

mix, defaulting to *True*, muxes both audio streams associated with the channel after recording is complete, with the alternative leaving the two streams separate.

class ami.core.**MuteAudio**(*channel*, *input=False*, *output=False*, *muted=False*)

Bases: pystrix.ami.ami._Request

Starts or stops muting audio on a channel.

Either (or both) directions can be silenced.

Requires system

__init__(*channel*, *input=False*, *output=False*, *muted=False*)

channel is the channel to be affected and *muted* indicates whether audio is being turned on or off. *input* (from the channel) and *output* (to the channel) indicate the subchannels to be adjusted.

class ami.core.**Originate_Application**(*channel*, *application*, *data=()*, *timeout=None*, *callerid=None*, *variables={}*, *account=None*, *async_=True*)

Bases: ami.core._Originate

Initiates a call that answers, executes an arbitrary dialplan application, and hangs up.

Requires call

__init__(*channel*, *application*, *data=()*, *timeout=None*, *callerid=None*, *variables={}*, *account=None*, *async_=True*)

channel is the destination to be called, expressed as a fully qualified Asterisk channel, like "SIP/test-account@example.org".

application is the name of the application to be executed, and *data* is optionally any parameters to pass to the application, as an ordered sequence (list or tuple) of strings, escaped as necessary (the ',' character is special).

timeout, if given, is the number of milliseconds to wait before dropping an unanwsered call. If set, the request's timeout value will be set to this number + 2 seconds, removing the need to set both variables. If not set, the request's timeout value will be set to ten minutes.

callerid is an optinal string of the form "name"<number>, where 'name' is the name to be displayed (on supporting channels) and 'number' is the source identifier, typically a string of digits on most channels that may interact with the PSTN.

variables is an oprional dictionary of key-value variable pairs to be set as part of the channel's namespace.

account is an optional account code to be associated with the channel, useful for tracking billing information.

*async_* should always be *True*. If not, only one unanswered call can be active at a time.

class ami.core.**Originate_Context**(*channel*, *context*, *extension*, *priority*, *timeout=None*, *callerid=None*, *variables={}*, *account=None*, *async_=True*)

Bases: ami.core._Originate

Initiates a call with instructions derived from an arbitrary context/extension/priority.

Requires call

__init__(*channel*, *context*, *extension*, *priority*, *timeout=None*, *callerid=None*, *variables={}*, *account=None*, *async_=True*)

channel is the destination to be called, expressed as a fully qualified Asterisk channel, like "SIP/test-account@example.org".

context, *extension*, and *priority*, must match a triple known to Asterisk internally. No validation is performed, so specifying an invalid target will terminate the call immediately.

timeout, if given, is the number of milliseconds to wait before dropping an unanwsered call. If set, the request's timeout value will be set to this number + 2 seconds, removing the need to set both variables. If not set, the request's timeout value will be set to ten minutes.

callerid is an optinal string of the form "name"<number>, where 'name' is the name to be displayed (on supporting channels) and 'number' is the source identifier, typically a string of digits on most channels that may interact with the PSTN.

variables is an oprional dictionary of key-value variable pairs to be set as part of the channel's namespace.

account is an optional account code to be associated with the channel, useful for tracking billing information.

*async_* should always be *True*. If not, only one unanswered call can be active at a time.

class ami.core.**Park**(*channel*, *channel_callback*, *timeout=None*)
Bases: pystrix.ami.ami._Request

Parks a call for later retrieval.

Requires call

__init__(*channel*, *channel_callback*, *timeout=None*)
channel is the channel to be parked and *channel_callback* is the channel to which parking information is announced.

If *timeout*, a number of milliseconds, is given, then *channel_callback* is given *channel* if the call was not previously retrieved.

class ami.core.**ParkedCalls**
Bases: pystrix.ami.ami._Request

Lists all parked calls.

Any number of 'ParkedCall' events may be generated in response to this request, followed by one 'ParkedCallsComplete'.

__init__()
action is the type of action being requested of the Asterisk server.

class ami.core.**PauseMonitor**(*channel*)
Bases: pystrix.ami.ami._Request

Pauses the recording of a monitored channel. The channel must have previously been selected by the *Monitor* action.

Requires call

__init__(*channel*)
channel is the channel to be affected.

class ami.core.**Ping**
Bases: pystrix.ami.ami._Request

Pings the AMI server. The response value has a 'RTT' attribute, which is the number of seconds the trip took, as a floating-point number, or -1 in case of failure.

__init__()
action is the type of action being requested of the Asterisk server.

class ami.core.**PlayDTMF**(*channel*, *digit*)
Bases: pystrix.ami.ami._Request

Plays a DTMF tone on a channel.

Requires call

__init__(*channel*, *digit*)
channel is the channel to be affected, and *digit* is the tone to play.

class ami.core.**QueueAdd**(*interface*, *queue*, *membername=None*, *penalty=0*, *paused=False*)
Bases: pystrix.ami.ami._Request

Adds a member to a queue.

Upon success, a 'QueueMemberAdded' event will be generated.

Requires agent

__init__(*interface*, *queue*, *membername=None*, *penalty=0*, *paused=False*)
Adds the device identified by *interface* to the given *queue*.

membername optionally provides a friendly name for logging purposes, *penalty* establishes a priority structure (lower priorities first, defaulintg to 0) for call escalation, and *paused* optinally allows the interface to start in a disabled state.

class ami.core.**QueueLog**(*queue*, *event*, *interface=None*, *uniqueid=None*, *message=None*)
Bases: pystrix.ami.ami._Request

Adds an arbitrary record to the queue log.

Requires agent

__init__(*queue*, *event*, *interface=None*, *uniqueid=None*, *message=None*)
queue is the queue to which the *event* is to be attached.

interface optionally allows the event to be associated with a specific queue member.

uniqueid's purpose is presently unknown.

message's purpose is presently unknown.

class ami.core.**QueuePause**(*interface*, *paused*, *queue=None*)
Bases: pystrix.ami.ami._Request

Pauses or unpauses a member in one or all queues.

Upon success, a 'QueueMemberPaused' event will be generated for all affected queues.

Requires agent

__init__(*interface*, *paused*, *queue=None*)
interface is the device to be affected, and *queue* optionally limits the scope to a single queue. *paused* must be *True* or *False*, to control the action being taken.

class ami.core.**QueuePenalty**(*interface*, *penalty*, *queue=None*)
Bases: pystrix.ami.ami._Request

Changes the penalty value associated with a queue member, in one or all queues.

Requires agent

__init__(*interface*, *penalty*, *queue=None*)
Changes the *penalty* value associated with *interface* in all queues, unless *queue* is defined, limiting it to one.

class ami.core.**QueueReload**(*queue=None*, *members='yes'*, *rules='yes'*, *parameters='yes'*)
Bases: pystrix.ami.ami._Request

Reloads properties from config files for one or all queues.

Requires agent

__init__(*queue=None*, *members='yes'*, *rules='yes'*, *parameters='yes'*)
Reloads parameters for all queues, unless *queue* is defined, limiting it to one.

members is 'yes' (default) or 'no', indicating whether the member-list should be reloaded.

rules is 'yes' (default) or 'no', indicating whether the rule-list should be reloaded.

parameters is 'yes' (default) or 'no', indicating whether the parameter-list should be reloaded.

class ami.core.**QueueRemove**(*interface*, *queue*)
Bases: pystrix.ami.ami._Request

Removes a member from a queue.

Upon success, a 'QueueMemberRemoved' event will be generated.

Requires agent

__init__(*interface*, *queue*)
Removes the device identified by *interface* from the given *queue*.

class ami.core.**QueueStatus**(*queue=None*)
Bases: pystrix.ami.ami._Request

Describes the status of one (or all) queues.

Upon success, 'QueueParams', 'QueueMember', and 'QueueEntry' events will be generated, ending with 'QueueStatusComplete'.

__init__(*queue=None*)
Describes all queues in the system, unless *queue* is given, which limits the scope to one.

class ami.core.**QueueSummary**(*queue=None*)
Bases: pystrix.ami.ami._Request

Describes the Summary of one (or all) queues.

Upon success, 'QueueSummary' event will be generated, ending with 'QueueSummaryComplete'.

__init__(*queue=None*)
Describes all queues in the system, unless *queue* is given, which limits the scope to one.

class ami.core.**Redirect**(*channel*, *context*, *extension*, *priority*)
Bases: pystrix.ami.ami._Request

Redirects a call to an arbitrary context/extension/priority.

Requires call

__init__(*channel*, *context*, *extension*, *priority*)
channel is the destination to be redirected.

context, *extension*, and *priority*, must match a triple known to Asterisk internally. No validation is performed, so specifying an invalid target will terminate the call immediately.

class ami.core.**Reload**(*module=None*)
Bases: pystrix.ami.ami._Request

Reloads Asterisk's configuration globally or for a specific module.

Requires call

__init__(*module=None*)
If given, *module* limits the scope of the reload to a specific module, named without extension.

class ami.core.**SendText**(*channel*, *message*)
Bases: pystrix.ami.ami._Request

Sends text along a supporting channel.

Requires call

__init__(*channel*, *message*)
channel is the channel along which to send *message*.

class ami.core.**SetCDRUserField**(*channel*, *user_field*)
Bases: pystrix.ami.ami._Request

Sets the user-field attribute for the CDR associated with a channel.

Requires call

__init__(*channel*, *user_field*)
channel is the channel to be affected, and *user_field* is the value to set.

class ami.core.**Setvar**(*variable*, *value*, *channel=None*)
Bases: pystrix.ami.ami._Request

Sets a channel-level or global variable.

Requires call

__init__(*variable*, *value*, *channel=None*)
value is the value to be set under *variable*.

channel is the channel to be affected, or *None*, the default, if the variable is global.

class ami.core.**SIPnotify**(*channel*, *headers={}*)
Bases: pystrix.ami.ami._Request

Sends a SIP NOTIFY to the remote party on a channel.

Requires call

__init__(*channel*, *headers={}*)
channel is the channel along which to send the NOTIFY.

headers is a dictionary of key-value pairs to be inserted as SIP headers.

class ami.core.**SIPpeers**
Bases: pystrix.ami.ami._Request

Lists all SIP peers.

Any number of 'PeerEntry' events may be generated in response to this request, followed by one 'PeerlistComplete'.

Requires system

__init__()
action is the type of action being requested of the Asterisk server.

class ami.core.**SIPqualify**(*peer*)
Bases: pystrix.ami.ami._Request

Sends a SIP OPTIONS to the specified peer, mostly to ensure its presence.

Some events are likely raised by this, but they're unknown at the moment.

Requires system

__init__(*peer*)
peer is the peer to ping.

class ami.core.**SIPshowpeer**(*peer*)
Bases: pystrix.ami.ami._Request

Provides detailed information about a SIP peer.

The response has the following key-value pairs:

- 'ACL': True or False
- 'Address-IP': The IP of the peer
- 'Address-Port': The port of the peer, as an integer
- 'AMAflags': "Unknown"
- 'Callgroup': ?
- 'Callerid': "Linksys #2" <555>
- 'Call-limit': ?
- 'Channeltype': "SIP"
- 'ChanObjectType': "peer"
- 'CID-CallingPres': ?
- 'Context': The context associated with the peer
- 'CodecOrder': The order in which codecs are tried
- 'Codecs': A list of supported codecs
- 'Default-addr-IP': ?
- 'Default-addr-port': ?
- 'Default-Username': ?
- 'Dynamic': True or False, depending on whether the peer is resolved by static IP or authentication
- 'Language': The language preference (may be empty) of this peer
- 'LastMsgsSent': ?
- 'MaxCallBR': The maximum bitrate in kbps supported by the peer, as an integer
- 'MD5SecretExist': True or False, depending on whether an MD5 secret is defined
- 'ObjectName': The internal name of the peer
- 'Pickupgroup': ?
- 'Reg-Contact': The registration contact address for this peer
- 'RegExpire': Time in seconds until SIP registration expires, as an integer
- 'RegExtension': ?
- 'SecretExist': True or False, depending on whether a secret is defined.
- 'SIP-AuthInsecure': True or False
- 'SIP-CanReinvite': True or False, depending on whether the peer supports REINVITE
- 'SIP-DTMFmode': The DTMF transport mode to use with this peer, "rfc2833" or ?

- 'SIP-NatSupport': The NATting workarounds supported by this peer, "RFC3581" or ?
- 'SIP-PromiscRedir': True or False, depending on whether this peer is allowed to arbitrarily redirect calls
- 'SIP-Useragent': The User-Agent of the peer
- 'SIP-UserPhone': True or False, (presumably) depending on whether this peer is a terminal device
- 'SIP-VideoSupport': True or False
- 'SIPLastMsg': ?
- 'Status': 'Unmonitored', 'OK (d+ ms)'
- 'ToHost': ?
- 'TransferMode': "open"
- 'VoiceMailbox': The mailbox associated with the peer; may be null

Requires system

__init__(*peer*)
peer is the identifier of the peer for which information is to be retrieved.

class ami.core.**SIPshowregistry**
Bases: pystrix.ami.ami._Request

Lists all SIP registrations.

Any number of 'RegistryEntry' events may be generated in response to this request, followed by one 'RegistrationsComplete'.

Requires system

__init__()
action is the type of action being requested of the Asterisk server.

class ami.core.**Status**(*channel*)
Bases: pystrix.ami.ami._Request

Lists the status of an active channel.

Zero or one 'Status' events are generated, followed by a 'StatusComplete' event.

Requires call

__init__(*channel*)
channel is the channel for which status information is to be retrieved.

class ami.core.**StopMonitor**(*channel*)
Bases: pystrix.ami.ami._Request

Stops recording a monitored channel. The channel must have previously been selected by the *Monitor* action.

Requires call

__init__(*channel*)
channel is the channel to be affected.

class ami.core.**UnpauseMonitor**(*channel*)
Bases: pystrix.ami.ami._Request

Unpauses recording on a monitored channel. The channel must have previously been selected by the *Monitor* action.

Requires call

__init__(*channel*)
channel is the channel to be affected.

class ami.core.**UpdateConfig**(*src_filename*, *dst_filename*, *changes*, *reload=True*)
Bases: pystrix.ami.ami._Request

Updates any number of values in an Asterisk configuration file.

Requires config

__init__(*src_filename*, *dst_filename*, *changes*, *reload=True*)
Reads from *src_filename*, performing all *changes*, and writing to *dst_filename*.

If *reload* is *True*, the changes take effect immediately. If *reload* is the name of a module, that module is reloaded.

changes may be any iterable object countaining quintuples with the following items:

1. One of the following:

- 'NewCat': creates a new category
- 'RenameCat': renames a category
- 'DelCat': deletes a category
- 'Update': changes a value
- 'Delete': removes a value
- 'Append': adds a value

2. The name of the category to operate on
3. *None* or the name of the variable to operate on
4. *None* or the value to be set/added (has no effect with 'Delete')
5. *None* or a string that needs to be matched in the line to serve as a qualifier

class ami.core.**UserEvent**(***kwargs*)
Bases: pystrix.ami.ami._Request

Causes a 'UserEvent' event to be generated.

Requires user

__init__(***kwargs*)
Any keyword-arguments passed will be present in the generated event, making this usable as a crude form of message-passing between AMI clients.

class ami.core.**VoicemailUsersList**
Bases: pystrix.ami.ami._Request

Lists all voicemail information.

Any number of 'VoicemailUserEntry' events may be generated in response to this request, followed by one 'VoicemailUserEntryComplete'.

Requires system (probably)

__init__()
action is the type of action being requested of the Asterisk server.

Exceptions

exception ami.core.**ManagerAuthError**
Bases: pystrix.ami.ami.ManagerError

Indicates that a problem occurred while authenticating

3.1.2 DAHDI

DAHDI is an interface layer for integrating traditional telephony technologies with digital formats.

Members

All of the following objects should be accessed as part of the *ami.dahdi* namespace, regardless of the modules in which they are defined.

Actions

class ami.dahdi.**DAHDIDNDoff**(*dahdi_channel*)
Bases: pystrix.ami.ami._Request

Sets a DAHDI channel's DND status to off.

__init__(*dahdi_channel*)
dahdi_channel is the channel to modify.

class ami.dahdi.**DAHDIDNDon**(*dahdi_channel*)
Bases: pystrix.ami.ami._Request

Sets a DAHDI channel's DND status to on.

__init__(*dahdi_channel*)
dahdi_channel is the channel to modify.

class ami.dahdi.**DAHDIDialOffhook**(*dahdi_channel*, *number*)
Bases: pystrix.ami.ami._Request

Dials a number on an off-hook DAHDI channel.

__init__(*dahdi_channel*, *number*)
dahdi_channel is the channel to use and *number* is the number to dial.

class ami.dahdi.**DAHDIHangup**(*dahdi_channel*)
Bases: pystrix.ami.ami._Request

Hangs up a DAHDI channel.

__init__(*dahdi_channel*)
dahdi_channel is the channel to hang up.

class ami.dahdi.**DAHDIRestart**
Bases: pystrix.ami.ami._Request

Fully restarts all DAHDI channels.

__init__()
action is the type of action being requested of the Asterisk server.

class ami.dahdi.**DAHDIShowChannels**(*dahdi_channel=None*)
Bases: pystrix.ami.ami._Request

Provides the current status of all (or one) DAHDI channels through a series of 'DAHDIShowChannels' events, ending with a 'DAHDIShowChannelsComplete' event.

__init__(*dahdi_channel=None*)
action is the type of action being requested of the Asterisk server.

3.1.3 (Application) Confbridge

Confbridge is Asterisk's new conferencing subsystem, providing far greater functionality than Meetme, with better performance and structural design. While technically a part of Asterisk's core, it's specialised enough that pystrix treats it as a module.

Members

All of the following objects should be accessed as part of the *ami.app_confbirdge* namespace, regardless of the modules in which they are defined.

Actions

class ami.app_confbridge.**ConfbridgeKick**(*conference*, *channel*)
Bases: pystrix.ami.ami._Request

Kicks a participant from a ConfBridge room.

__init__(*conference*, *channel*)
channel is the channel to be kicked from *conference*.

class ami.app_confbridge.**ConfbridgeList**(*conference*)
Bases: pystrix.ami.ami._Request

Lists all participants in a ConfBridge room.

A series of 'ConfbridgeList' events follow, with one 'ConfbridgeListComplete' event at the end.

__init__(*conference*)
conference is the identifier of the bridge.

class ami.app_confbridge.**ConfbridgeListRooms**
Bases: pystrix.ami.ami._Request

Lists all ConfBridge rooms.

A series of 'ConfbridgeListRooms' events follow, with one 'ConfbridgeListRoomsComplete' event at the end.

__init__()
action is the type of action being requested of the Asterisk server.

class ami.app_confbridge.**ConfbridgeLock**(*conference*)
Bases: pystrix.ami.ami._Request

Locks a ConfBridge room, disallowing access to non-administrators.

__init__(*conference*)
conference is the identifier of the bridge.

class ami.app_confbridge.**ConfbridgeUnlock**(*conference*)
Bases: pystrix.ami.ami._Request

Unlocks a ConfBridge room, allowing access to non-administrators.

__init__(*conference*)
conference is the identifier of the bridge.

class ami.app_confbridge.**ConfbridgeMoHOn**(*conference*, *channel*)
Bases: pystrix.ami.ami._Request

Forces MoH to a participant in a ConfBridge room.

This action does not mute audio coming from the participant.

Depends on <path>.

__init__(*conference*, *channel*)
channel is the channel to which MoH should be started in *conference*.

class ami.app_confbridge.**ConfbridgeMoHOff**(*conference*, *channel*)
Bases: pystrix.ami.ami._Request

Stops forcing MoH to a participant in a ConfBridge room.

This action does not unmute audio coming from the participant.

Depends on <path>.

__init__(*conference*, *channel*)
channel is the channel to which MoH should be stopped in *conference*.

class ami.app_confbridge.**ConfbridgeMute**(*conference*, *channel*)
Bases: pystrix.ami.ami._Request

Mutes a participant in a ConfBridge room.

__init__(*conference*, *channel*)
channel is the channel to be muted in *conference*.

class ami.app_confbridge.**ConfbridgeUnmute**(*conference*, *channel*)
Bases: pystrix.ami.ami._Request

Unmutes a participant in a ConfBridge room.

__init__(*conference*, *channel*)
channel is the channel to be unmuted in *conference*.

class ami.app_confbridge.**ConfbridgePlayFile**(*file*, *conference*, *channel=None*)
Bases: pystrix.ami.ami._Request

Plays a file to individuals or an entire conference.

Note: This implementation is built upon the not-yet-accepted patch under https://issues.asterisk.org/jira/browse/ASTERISK-19571

__init__(*file*, *conference*, *channel=None*)
file, resolved like other Asterisk media, is played to *conference* or, if specified, a specific *channel* therein.

class ami.app_confbridge.**ConfbridgeStartRecord**(*conference*, *filename=None*)
Bases: pystrix.ami.ami._Request

Starts recording a ConfBridge conference.

A 'VarSet' event will be generated to indicate the absolute path of the recording. To identify it, match its 'Channel' key against "ConfBridgeRecorder/conf-?-...", where "..." is Asterisk-generated identification data

that can be discarded and "?" is the room ID. The 'Variable' key must be "MIXMONITOR_FILENAME", with the 'Value' key holding the file's path.

__init__(*conference*, *filename=None*)
conference is the room to be recorded, and *filename*, optional, is the path, Asterisk-resolved or absolute, of the file to write.

class ami.app_confbridge.**ConfbridgeStopRecord**(*conference*)
Bases: pystrix.ami.ami._Request

Stops recording a ConfBridge conference.

A 'Hangup' event will be generated when the recorder detaches from the conference. To identify it, match its 'Channel' key against "ConfBridgeRecorder/conf-?-...", where "..." is Asterisk-generated identification data that can be discarded and "?" is the room ID.

__init__(*conference*)
conference is the room being recorded.

class ami.app_confbridge.**ConfbridgeSetSingleVideoSrc**(*conference*, *channel*)
Bases: pystrix.ami.ami._Request

Sets the video source for the conference to a single channel's stream.

__init__(*conference*, *channel*)
channel is the video source in *conference*.

3.1.4 (Application) Meetme

Meetme is Asterisk's long-standing, now-being-phased-out conferencing subsystem. While technically a part of Asterisk's core, it's specialised enough that pystrix treats it as a module.

Members

All of the following objects should be accessed as part of the *ami.app_meetme* namespace, regardless of the modules in which they are defined.

Actions

class ami.app_meetme.**MeetmeList**(*conference=None*)
Bases: pystrix.ami.ami._Request

Lists all participants in all (or one) conferences.

A series of 'MeetmeList' events follow, with one 'MeetmeListComplete' event at the end.

Note that if no conferences are active, the response will indicate that it was not successful, per https://issues.asterisk.org/jira/browse/ASTERISK-16812

__init__(*conference=None*)
conference is the optional identifier of the bridge.

class ami.app_meetme.**MeetmeListRooms**
Bases: pystrix.ami.ami._Request

Lists all conferences.

A series of 'MeetmeListRooms' events follow, with one 'MeetmeListRoomsComplete' event at the end.

__init__()
action is the type of action being requested of the Asterisk server.

class ami.app_meetme.**MeetmeMute**(*meetme*, *usernum*)
Bases: pystrix.ami.ami._Request

Mutes a participant in a Meetme bridge.

Requires call

__init__(*meetme*, *usernum*)
meetme is the identifier of the bridge and *usernum* is the participant ID of the user to be muted, which is associated with a channel by the 'MeetmeJoin' event. If successful, this request will trigger a 'MeetmeMute' event.

class ami.app_meetme.**MeetmeUnmute**(*meetme*, *usernum*)
Bases: pystrix.ami.ami._Request

Unmutes a participant in a Meetme bridge.

Requires call

__init__(*meetme*, *usernum*)
meetme is the identifier of the bridge and *usernum* is the participant ID of the user to be unmuted, which is associated with a channel by the 'MeetmeJoin' event. If successful, this request will trigger a 'MeetmeMute' event.

3.2 Events

The AMI generates events in response to changes in its environment (unsolicited) or as a response to certain request actions. All known events are described in this section. Their usage is consistently a matter of registering a callback handler for an event with *ami.Manager.register_callback()* and then waiting for the event to occur.

3.2.1 Core

Asterisk provides a rich assortment of information-carrying events by default, the standard set of which are described here.

Members

All of the following objects should be accessed as part of the *ami.core_events* namespace, regardless of the modules in which they are defined.

Events

class ami.core_events.**AGIExec**(*response*)
Bases: pystrix.ami.ami._Event

Generated when an AGI script executes an arbitrary application.

- 'Channel': The channel in use
- 'Command': The command that was issued
- 'CommandId': The command's identifier, used to track events from start to finish

- 'SubEvent': "Start", "End"
- 'Result': Only present when 'SubEvent' is "End": "Success" (and "Failure"?)
- 'ResultCode': Only present when 'SubEvent' is "End": the result-code from Asterisk

process()
Translates the 'Result' header's value into a bool.

Translates the 'ResultCode' header's value into an int, setting it to *-1* if coercion fails.

class ami.core_events.**AsyncAGI**(*response*)
Bases: pystrix.ami.ami._Event

Generated when an AGI request is processed.

- All fields currently unknown

class ami.core_events.**ChannelUpdate**(*response*)
Bases: pystrix.ami.ami._Event

Describes a change in a channel.

Some fields are type-dependent and will appear as children of that type in the list.

- 'Channel': The channel being described
- 'Channeltype': One of the following types
  - 'SIP': SIP channels have the following fields
  - 'SIPcallid': 'DB45B1B5-1EAD11E1-B979D0B6-32548E42@10.13.38.201', the CallID negotiated with the endpoint; this should be present in any CDRs generated
  - 'SIPfullcontact': 'sip:flan@uguu.ca', the address of the SIP contact field, if any (observed during a REFER)
- 'UniqueID': An Asterisk-unique value

class ami.core_events.**CoreShowChannel**(*response*)
Bases: pystrix.ami.ami._Event

Provides the definition of an active Asterisk channel.

- 'AccountCode': The account code associated with the channel
- 'Application': The application currently being executed by the channel
- 'ApplicationData': The arguments provided to the application
- 'BridgedChannel': The channel to which this channel is connected, if any
- 'BridgedUniqueID': ?
- 'CallerIDnum': The (often) numeric address of the caller
- 'CallerIDname': The (optional, media-specific) name of the caller
- 'Channel': The channel being described
- 'ChannelState': One of the following numeric values, as a string:
  - '0': Not connected
  - '4': Alerting

- '6': Connected
- 'ChannelStateDesc': A lexical description of the channel's current state
- 'ConnectedLineNum': The (often) numeric address of the called party (may be nil)
- 'ConnectedLineName': The (optional, media-specific) name of the called party (may be nil)
- 'Context': The dialplan context in which the channel is executing
- 'Duration': The client's connection time in "hh:mm:ss" form
- 'Extension': The dialplan context in which the channel is executing
- 'Priority': The dialplan priority in which the channel is executing
- 'UniqueID': An Asterisk-unique value (the timestamp at which the channel was connected?)

process()
Translates the 'ChannelState' header's value into an int, setting it to *None* if coercion fails.

Replaces the 'Duration' header's value with the number of seconds, as an int, or -1 if conversion fails.

class ami.core_events.**CoreShowChannelsComplete**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that all Asterisk channels have been listed.

- 'ListItems' : The number of items returned prior to this event

process()
Translates the 'ListItems' header's value into an int, or -1 on failure.

class ami.core_events.**DBGetResponse**(*response*)
Bases: pystrix.ami.ami._Event

Provides the value requested from the database.

- 'Family': The family of the value being provided
- 'Key': The key of the value being provided
- 'Val': The value being provided, represented as a string

class ami.core_events.**DTMF**(*response*)
Bases: pystrix.ami.ami._Event

- 'Begin': 'Yes' or 'No', indicating whether this started or ended the DTMF press
- 'Channel': The channel being described
- 'Digit': The DTMF digit that was pressed
- 'Direction': 'Received' or 'Sent'
- 'End': 'Yes' or 'No', indicating whether this started or ended the DTMF press (inverse of *Begin*, though both may be *Yes* if the event has no duration)
- 'UniqueID': An Asterisk-unique value

process()
Translates 'Begin' and 'End' into booleans, and adds a 'Received':bool header.

class ami.core_events.**FullyBooted**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that Asterisk is online.

- 'Status': "Fully Booted"

class ami.core_events.**Hangup**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a channel has been hung up.

- 'Cause': One of the following numeric values, as a string:
- '0': Hung up
- '16': Normal clearing
- 'Cause-txt': Additional information related to the hangup
- 'Channel': The channel hung-up
- 'Uniqueid': An Asterisk unique value

process()
Translates the 'Cause' header's value into an int, setting it to *None* if coercion fails.

class ami.core_events.**HangupRequest**(*response*)
Bases: pystrix.ami.ami._Event

Emitted when a request to terminate the call is received.

- 'Channel': The channel identifier used by Asterisk
- 'Uniqueid': An Asterisk unique value

class ami.core_events.**MonitorStart**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that monitoring has begun.

- 'Channel': The channel being monitored
- 'Uniqueid': An Asterisk unique value

class ami.core_events.**MonitorStop**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that monitoring has ceased.

- 'Channel': The channel that was monitored
- 'Uniqueid': An Asterisk unique value

class ami.core_events.**NewAccountCode**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that the account-code associated with a channel has changed.

- 'AccountCode': The new account code
- 'Channel': The channel that was affected.
- 'OldAccountCode': The old account code

class ami.core_events.**Newchannel**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a new channel has been created.

- 'AccountCode': The billing account associated with the channel; may be empty

- 'CallerIDNum': The (often) numeric identifier of the caller
- 'CallerIDName': The caller's name, on supporting channels
- 'Channel': The channel identifier used by Asterisk
- 'ChannelState': One of the following numeric values, as a string:
- '0': Not connected
- '4': Alerting
- '6': Connected
- 'ChannelStateDesc': A lexical description of the channel's current state
- 'Context': The context that the channel is currently operating in
- 'Exten': The extension the channel is currently operating in
- 'Uniqueid': An Asterisk unique value

process()
Translates the 'ChannelState' header's value into an int, setting it to *None* if coercion fails.

class ami.core_events.**Newexten**(*response*)
Bases: pystrix.ami.ami._Event

Emitted when a channel switches executing extensions.

- 'AppData': The argument passed to the application
- 'Application': The application being invoked
- 'Channel': The channel identifier used by Asterisk
- 'Context': The context the channel is currently operating in
- 'Extension': The extension the channel is currently operating in
- 'Priority': The priority the channel is currently operating in
- 'Uniqueid': An Asterisk unique value

class ami.core_events.**Newstate**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a channel's state has changed.

- 'CallerIDNum': The (often) numeric identifier of the caller
- 'CallerIDName': The caller's name, on supporting channels
- 'Channel': The channel identifier used by Asterisk
- 'ChannelStateDesc': A lexical description of the channel's current state
- 'ChannelState': One of the following numeric values, as a string:
- '0': Not connected
- '4': Alerting
- '6': Connected
- 'ConnectedLineNum': ?

- 'ConnectedLineName': ?
- 'Uniqueid': An Asterisk unique value

process()
Translates the 'ChannelState' header's value into an int, setting it to *None* if coercion fails.

class ami.core_events.**OriginateResponse**(*response*)
Bases: pystrix.ami.ami._Event

Describes the result of an Originate request.

- 'CallerIDName': The supplied source name
- 'CallerIDNum': The supplied source address
- 'Channel': The Asterisk channel used for the call
- 'Context': The dialplan context into which the call was placed, as a string; unused for applications
- 'Exten': The dialplan extension into which the call was placed, as a string; unused for applications
- 'Reason': An integer as a string, ostensibly one of the *ORIGINATE_RESULT* constants; undefined integers may exist

process()
Sets the 'Reason' values to an int, one of the *ORIGINATE_RESULT* constants, with -1 indicating failure.

class ami.core_events.**ParkedCall**(*response*)
Bases: pystrix.ami.ami._Event

Describes a parked call.

- 'CallerID': The ID of the caller, ".+?" <.+?>
- 'CallerIDName' (optional): The name of the caller, on supporting channels
- 'Channel': The channel of the parked call
- 'Exten': The extension associated with the parked call
- 'From': The callback channel associated with the call
- 'Timeout' (optional): The time remaining before the call is reconnected with the callback channel

process()
Translates the 'Timeout' header's value into an int, setting it to *None* if coercion fails, and leaving it absent if it wasn't present in the original response.

class ami.core_events.**ParkedCallsComplete**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that all parked calls have been listed.

- 'Total' : The number of items returned prior to this event

process()
Translates the 'Total' header's value into an int, or -1 on failure.

class ami.core_events.**PeerEntry**(*response*)
Bases: pystrix.ami.ami._Event

Describes a peer.

- 'ChannelType': The type of channel being described.
- 'SIP'

- 'ObjectName': The internal name by which this peer is known
- 'ChanObjectType': The type of object
- 'peer'
- 'IPaddress' (optional): The IP of the peer
- 'IPport' (optional): The port of the peer
- 'Dynamic': 'yes' or 'no', depending on whether the peer is resolved by IP or authentication
- 'Natsupport': 'yes' or 'no', depending on whether the peer's messages' content should be trusted for routing purposes. If not, packets are sent back to the last hop
- 'VideoSupport': 'yes' or 'no'
- 'ACL': 'yes' or 'no'
- 'Status': 'Unmonitored', 'OK (d+ ms)'
- 'RealtimeDevice': 'yes' or 'no'

process()
Translates the 'Port' header's value into an int, setting it to *None* if coercion fails, and leaving it absent if it wasn't present in the original response.

Translates the 'Dynamic', 'Natsupport', 'VideoSupport', 'ACL', and 'RealtimeDevice' headers' values into bools.

Translates 'Status' into the number of milliseconds since the peer was last seen or -2 if unmonitored. -1 if parsing failed.

class ami.core_events.**PeerlistComplete**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that all peers have been listed.

- 'ListItems' : The number of items returned prior to this event

process()
Translates the 'ListItems' header's value into an int, or -1 on failure.

class ami.core_events.**QueueEntry**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a call is waiting to be answered.

- 'Channel': The channel of the inbound call
- 'CallerID': The (often) numeric ID of the caller
- 'CallerIDName' (optional): The friendly name of the caller on supporting channels
- 'Position': The numeric position of the caller in the queue
- 'Queue': The queue in which the caller is waiting
- 'Wait': The number of seconds the caller has been waiting

process()
Translates the 'Position' and 'Wait' headers' values into ints, setting them to -1 on error.

class ami.core_events.**QueueMember**(*response*)
Bases: pystrix.ami.ami._Event

Describes a member of a queue.

- 'CallsTaken': The number of calls received by this member
- 'LastCall': The UNIX timestamp of the last call taken, or 0 if none
- 'Location': The interface in the queue
- 'MemberName' (optional): The friendly name of the member
- 'Membership': "dynamic" ("static", too?)
- 'Paused': '1' or '0' for 'true' and 'false', respectively
- 'Penalty': The selection penalty to apply to this member (higher numbers mean later selection)
- 'Queue': The queue to which the member belongs
- 'Status': One of the following, as a string:
- '0': Idle
- '1': In use
- '2': Busy

process()
Translates the 'CallsTaken', 'LastCall', 'Penalty', and 'Status' headers' values into ints, setting them to -1 on error.

'Paused' is set to a bool.

class ami.core_events.**QueueMemberAdded**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a member was added to a queue.

- 'CallsTaken': The number of calls received by this member
- 'LastCall': The UNIX timestamp of the last call taken, or 0 if none
- 'Location': The interface added to the queue
- 'MemberName' (optional): The friendly name of the member
- 'Membership': "dynamic" ("static", too?)
- 'Paused': '1' or '0' for 'true' and 'false', respectively
- 'Penalty': The selection penalty to apply to this member (higher numbers mean later selection)
- 'Queue': The queue to which the member was added
- 'Status': One of the following, as a string:
- '0': Idle
- '1': In use
- '2': Busy

process()
Translates the 'CallsTaken', 'LastCall', 'Penalty', and 'Status' headers' values into ints, setting them to -1 on error.

'Paused' is set to a bool.

class ami.core_events.**QueueMemberPaused**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that the pause-state of a queue member was changed.

- 'Location': The interface added to the queue
- 'MemberName' (optional): The friendly name of the member
- 'Paused': '1' or '0' for 'true' and 'false', respectively
- 'Queue': The queue in which the member was paused

process()
'Paused' is set to a bool.

class ami.core_events.**QueueMemberRemoved**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a member was removed from a queue.

- 'Location': The interface removed from the queue
- 'MemberName' (optional): The friendly name of the member
- 'Queue': The queue from which the member was removed

class ami.core_events.**QueueParams**(*response*)
Bases: pystrix.ami.ami._Event

Describes the attributes of a queue.

- 'Abandoned': The number of calls that have gone unanswered
- 'Calls': The number of current calls in the queue
- 'Completed': The number of completed calls
- 'Holdtime': ?
- 'Max': ?
- 'Queue': The queue being described
- 'ServiceLevel': ?
- 'ServiceLevelPerf': ?
- 'Weight': ?

process()
Translates the 'Abandoned', 'Calls', 'Completed', 'Holdtime', and 'Max' headers' values into ints, setting them to -1 on error.

Translates the 'ServiceLevel', 'ServiceLevelPerf', and 'Weight' values into floats, setting them to -1 on error.

class ami.core_events.**QueueStatusComplete**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a QueueStatus request has completed.

class ami.core_events.**QueueSummary**(*response*)
Bases: pystrix.ami.ami._Event

Describes a Summary of a queue. Example:

- Event: QueueSummary
- Queue: default
- LoggedIn: 0
- Available: 0
- Callers: 0
- HoldTime: 0
- TalkTime: 0
- LongestHoldTime: 0
- Event: QueueSummaryComplete
- EventList: Complete
- ListItems: 2

process()
Translates the 'LoggedIn', 'Available', 'Callers', 'Holdtime', 'TalkTime' and 'LongestHoldTime' headers' values into ints, setting them to -1 on error.

class ami.core_events.**QueueSummaryComplete**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a QueueSummary request has completed.

class ami.core_events.**RegistryEntry**(*response*)
Bases: pystrix.ami.ami._Event

Describes a SIP registration.

- 'Domain': The domain in which the registration took place
- 'DomainPort': The port in use in the registration domain
- 'Host': The address of the host
- 'Port': The port in use on the host
- 'Refresh': The amount of time remaining until the registration will be renewed
- 'RegistrationTime': The time at which the registration was made, as a UNIX timestamp
- 'State': The current status of the registration
- 'Username': The username used for the registration

process()
Translates the 'DomainPort', 'Port', 'Refresh', and 'RegistrationTime' values into ints, setting them to -1 on error.

class ami.core_events.**RegistrationsComplete**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that all registrations have been listed.

- 'ListItems' : The number of items returned prior to this event

process()
Translates the 'ListItems' header's value into an int, or -1 on failure.

class ami.core_events.**Reload**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that Asterisk's configuration was reloaded.

- 'Message': A human-readable summary
- 'Module': The affected module
- 'Status': 'Enabled'

class ami.core_events.**RTCPReceived**(*response*)
Bases: pystrix.ami.ami._Event

A Real Time Control Protocol message emitted by Asterisk when using an RTP-based channel, providing statistics on the quality of a connection, for the receiving leg.

- 'DLSR': ? (float as a string, followed by '(sec)')
- 'FractionLost': The percentage of lost packets, a float as a string
- 'From': The IP and port of the source, separated by a colon
- 'HighestSequence': ? (int as string)
- 'IAJitter': ? (float as a string)
- 'LastSR': ? (int as string)
- 'PacketsLost': The number of lost packets, as a string
- 'PT': ?
- 'ReceptionReports': The number of reports used to compile this event, as a string
- 'SenderSSRC': Session source
- 'SequenceNumberCycles': ?

process()
Translates the 'HighestSequence', 'LastSR', 'PacketsLost', 'ReceptionReports, and 'SequenceNumbercycles' values into ints, setting them to -1 on error.

Translates the 'DLSR', 'FractionLost', 'IAJitter', and 'SentNTP' values into floats, setting them to -1 on error.

Splits 'From' into a tuple of IP:str and port:int, or sets it to *None* if the format is unknown.

class ami.core_events.**RTCPSent**(*response*)
Bases: pystrix.ami.ami._Event

A Real Time Control Protocol message emitted by Asterisk when using an an RTP-based channel, providing statistics on the quality of a connection, for the sending leg.

- 'CumulativeLoss': The number of lost packets, as a string
- 'DLSR': ? (float as a string, followed by '(sec)')
- 'FractionLost': The percentage of lost packets, a float as a string
- 'IAJitter': ? (float as a string)
- 'OurSSRC': Session source
- 'ReportBlock': ?

- 'SentNTP': The NTP value, a float as a string
- 'SentOctets': The number of bytes sent, as a string
- 'SentPackets': The number of packets sent, as a string
- 'SentRTP': The number of RTP events sent, as a string
- 'TheirLastSR': ? (int as string)
- 'To': The IP and port of the recipient, separated by a colon

process()
Translates the 'CumulativeLoss', 'SentOctets', 'SentPackets', 'SentRTP', and 'TheirLastSR' values into ints, setting them to -1 on error.

Translates the 'DLSR', 'FractionLost', 'IAJitter', and 'SentNTP' values into floats, setting them to -1 on error.

Splits 'To' into a tuple of IP:str and port:int, or sets it to *None* if the format is unknown.

class ami.core_events.**Shutdown**(*response*)
Bases: pystrix.ami.ami._Event

Emitted when Asterisk shuts down.

- 'Restart': "True" or "False"
- 'Shutdown': "Cleanly"

process()
'Restart' is set to a bool.

class ami.core_events.**SoftHangupRequest**(*response*)
Bases: pystrix.ami.ami._Event

Emitted when a request to terminate the call is exchanged.

- 'Channel': The channel identifier used by Asterisk
- 'Cause': The reason for the disconnect, a numeric value as a string:
- '16': ?
- '32': ?
- 'Uniqueid': An Asterisk unique value

class ami.core_events.**Status**(*response*)
Bases: pystrix.ami.ami._Event

Describes the current status of a channel.

- 'Account': The billing account associated with the channel; may be empty
- 'Channel': The channel being described
- 'CallerID': The ID of the caller, ".+?" <.+?>
- 'CallerIDNum': The (often) numeric component of the CallerID
- 'CallerIDName' (optional): The, on suporting channels, name of the caller, enclosed in quotes
- 'Context': The context of the directive the channel is executing
- 'Extension': The extension of the directive the channel is executing

- 'Link': ?
- 'Priority': The priority of the directive the channel is executing
- 'Seconds': The number of seconds the channel has been active
- 'State': "Up"
- 'Uniqueid': An Asterisk unique value

process()
Translates the 'Seconds' header's value into an int, setting it to -1 on error.

class ami.core_events.**StatusComplete**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that all requested channel information has been provided.

- 'Items': The number of items emitted prior to this event

process()
Translates the 'Items' header's value into an int, or -1 on failure.

class ami.core_events.**UserEvent**(*response*)
Bases: pystrix.ami.ami._Event

Generated in response to the UserEvent request.

- *: Any key-value pairs supplied with the request, as strings

class ami.core_events.**VarSet**(*response*)
Bases: pystrix.ami.ami._Event

Emitted when a variable is set, either globally or on a channel.

- 'Channel' (optional): The channel on which the variable was set, if not global
- 'Uniqueid': An Asterisk unique value
- 'Value': The value of the variable, as a string
- 'Variable': The name of the variable that was set

class ami.core_events.**VoicemailUserEntry**(*response*)
Bases: pystrix.ami.ami._Event

Describes a voicemail user.

- **'AttachMessage': "Yes", "No"**
  - 'AttachmentFormat': unknown
  - 'CallOperator': "Yes", "No"
  - 'CanReview': "Yes", "No"
- 'Callback': unknown
- 'DeleteMessage': "Yes", "No"
- 'Dialout': unknown
- 'Email': unknown
- 'ExitContext': The context to use when leaving the mailbox
- 'Fullname': unknown
- **'IMAPFlags': Any associated IMAP flags (IMAP only)**

  - 'IMAPPort': The associated IMAP port (IMAP only)
  - 'IMAPServer': The associated IMAP server (IMAP only)
  - 'IMAPUser': The associated IMAP username (IMAP only)
  - 'Language': The language to use for voicemail prompts
  - 'MailCommand': unknown
  - 'MaxMessageCount': The maximum number of messages that can be stored
  - 'MaxMessageLength': The maximum length of any particular message
  - 'NewMessageCount': The number of unheard messages
  - 'OldMessageCount': The number of previously heard messages (IMAP only)
  - 'Pager': unknown
- 'SayCID': "Yes", "No"
- 'SayDurationMinimum': The minumum amount of time a message may be
- 'SayEnvelope': "Yes", "No"
- 'ServerEmail': unknown
- 'TimeZone': The timezone of the mailbox
- **'UniqueID': unknown**
  - 'VMContext': The associated Asterisk context
  - 'VoiceMailbox': The associated mailbox
  - 'VolumeGain': A floating-point value

process()

Translates the 'MaxMessageCount', 'MaxMessageLength', 'NewMessageCount', 'OldMessageCount', and 'SayDurationMinimum' values into ints, setting them to -1 on error.

Translates the 'VolumeGain' value into a float, setting it to None on error.

Translates the 'AttachMessage', 'CallOperator', 'CanReview', 'DeleteMessage', 'SayCID', and 'SayEnvelope' values into booleans.

class ami.core_events.**VoicemailUserEntryComplete**(*response*)

Bases: pystrix.ami.ami._Event

Indicates that all requested voicemail user definitions have been provided.

No, its name is not a typo; it's really "Entry" in Asterisk's code.

Aggregate Events

class ami.core_events.**CoreShowChannels_Aggregate**(*action_id*)

Bases: pystrix.ami.ami._Aggregate

Emitted after all channels have been received in response to a CoreShowChannels request.

Its members consist of CoreShowChannel events.

It is finalised by CoreShowChannelsComplete.

class ami.core_events.**ParkedCalls_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all parked calls have been received in response to a ParkedCalls request.

Its members consist of ParkedCall events.

It is finalised by ParkedCallsComplete.

class ami.core_events.**QueueStatus_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all queue properties have been received in response to a QueueStatus request.

Its members consist of QueueParams, QueueMember, and QueueEntry events.

It is finalised by QueueStatusComplete.

class ami.core_events.**QueueSummary_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all queue properties have been received in response to a QueueSummary request.

Its members consist of QueueSummary events.

It is finalised by QueueSummaryComplete.

class ami.core_events.**SIPpeers_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all queue properties have been received in response to a SIPpeers request.

Its members consist of 'PeerEntry' events.

It is finalised by PeerlistComplete.

class ami.core_events.**SIPshowregistry_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all SIP registrants have been received in response to a SIPshowregistry request.

Its members consist of RegistryEntry events.

It is finalised by RegistrationsComplete.

class ami.core_events.**Status_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all statuses have been received in response to a Status request.

Its members consist of Status events.

It is finalised by StatusComplete.

class ami.core_events.**VoicemailUsersList_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all voicemail users have been received in response to a VoicemailUsersList request.

Its members consist of VoicemailUserEntry events.

It is finalised by VoicemailUserEntryComplete.

3.2.2 DAHDI

DAHDI is an interface layer for integrating traditional telephony technologies with digital formats.

Members

All of the following objects should be accessed as part of the *ami.dahdi_events* namespace, regardless of the modules in which they are defined.

Events

class ami.dahdi_events.**DAHDIShowChannels**(*response*)
Bases: pystrix.ami.ami._Event

Describes the current state of a DAHDI channel.

Yes, the event's name is pluralised.

- 'AccountCode': unknown (not present if the DAHDI channel is down)
- 'Alarm': unknown
- 'Channel': The channel being described (not present if the DAHDI channel is down)
- 'Context': The Asterisk context associated with the channel
- 'DAHDIChannel': The ID of the DAHDI channel
- 'Description': unknown
- 'DND': 'Disabled' or 'Enabled'
- 'Signalling': A lexical description of the current signalling state
- 'SignallingCode': A numeric description of the current signalling state
- 'Uniqueid': unknown (not present if the DAHDI channel is down)

process()
Translates the 'DND' header's value into a bool.

Translates the 'DAHDIChannel' and 'SignallingCode' headers' values into ints, or -1 on failure.

class ami.dahdi_events.**DAHDIShowChannelsComplete**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that all DAHDI channels have been described.

- 'Items': The number of items returned prior to this event

process()
Translates the 'Items' header's value into an int, or -1 on failure.

Aggregate Events

class ami.dahdi_events.**DAHDIShowChannels_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all DAHDI channels have been enumerated in response to a DAHDIShowChannels request.

Its members consist of DAHDIShowChannels events.

It is finalised by DAHDIShowChannelsComplete.

3.2.3 (Application) Confbridge

Confbridge is Asterisk's new conferencing subsystem, providing far greater functionality than Meetme, with better performance and structural design. While technically a part of Asterisk's core, it's specialised enough that pystrix treats it as a module.

Members

All of the following objects should be accessed as part of the *ami.app_confbridge_events* namespace, regardless of the modules in which they are defined.

Events

class ami.app_confbridge_events.**ConfbridgeEnd**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a ConfBridge has ended.

- 'Conference' : The room's identifier

class ami.app_confbridge_events.**ConfbridgeJoin**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a participant has joined a ConfBridge room.

NameRecordingPath blocks on <path>

- 'CallerIDname' (optional) : The name, on supporting channels, of the participant
- 'CallerIDnum' : The (often) numeric address of the participant
- 'Channel' : The channel that joined
- 'Conference' : The identifier of the room that was joined
- 'NameRecordingPath' (optional) : The path at which the user's name-recording is kept
- 'Uniqueid' : An Asterisk unique value

class ami.app_confbridge_events.**ConfbridgeLeave**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a participant has left a ConfBridge room.

- 'CallerIDname' (optional) : The name, on supporting channels, of the participant
- 'CallerIDnum' : The (often) numeric address of the participant
- 'Channel' : The channel that left
- 'Conference' : The identifier of the room that was left
- 'Uniqueid' : An Asterisk unique value

class ami.app_confbridge_events.**ConfbridgeList**(*response*)
Bases: pystrix.ami.ami._Event

Describes a participant in a ConfBridge room.

- 'Admin' : 'Yes' or 'No'
- 'CallerIDNum' : The (often) numeric address of the participant
- 'CallerIDName' (optional) : The name of the participant on supporting channels

- 'Channel' : The Asterisk channel in use by the participant
- 'Conference' : The room's identifier
- 'MarkedUser' : 'Yes' or 'No'
- 'NameRecordingPath' (optional) : The path at which the user's name-recording is kept

process()
Translates the 'Admin' and 'MarkedUser' headers' values into bools.

class ami.app_confbridge_events.**ConfbridgeListComplete**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that all participants in a ConfBridge room have been enumerated.

- 'ListItems' : The number of items returned prior to this event

process()
Translates the 'ListItems' header's value into an int, or -1 on failure.

class ami.app_confbridge_events.**ConfbridgeListRooms**(*response*)
Bases: pystrix.ami.ami._Event

Describes a ConfBridge room.

And, yes, it's plural in Asterisk, too.

- 'Conference' : The room's identifier
- 'Locked' : 'Yes' or 'No'
- 'Marked' : The number of marked users
- 'Parties' : The number of participants

process()
Translates the 'Marked' and 'Parties' headers' values into ints, or -1 on failure.

Translates the 'Locked' header's value into a bool.

class ami.app_confbridge_events.**ConfbridgeListRoomsComplete**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that all ConfBridge rooms have been enumerated.

- 'ListItems' : The number of items returned prior to this event

process()
Translates the 'ListItems' header's value into an int, or -1 on failure.

class ami.app_confbridge_events.**ConfbridgeStart**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a ConfBridge has started.

- 'Conference' : The room's identifier

class ami.app_confbridge_events.**ConfbridgeTalking**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a participant has started or stopped talking.

- 'Channel' : The Asterisk channel in use by the participant
- 'Conference' : The room's identifier
- 'TalkingStatus' : 'on' or 'off'

- 'Uniqueid' : An Asterisk unique value

process()
Translates the 'TalkingStatus' header's value into a bool.

Aggregate Events

class ami.app_confbridge_events.**ConfbridgeList_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all conference participants have been received in response to a ConfbridgeList request.

Its members consist of ConfbridgeList events.

It is finalised by ConfbridgeListComplete.

class ami.app_confbridge_events.**ConfbridgeListRooms_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all conference rooms have been received in response to a ConfbridgeListRooms request.

Its members consist of ConfbridgeListRooms events.

It is finalised by ConfbridgeListRoomsComplete.

3.2.4 (Application) Meetme

Meetme is Asterisk's long-standing, now-being-phased-out conferencing subsystem. While technically a part of Asterisk's core, it's specialised enough that pystrix treats it as a module.

Members

All of the following objects should be accessed as part of the *ami.app_meetme_events* namespace, regardless of the modules in which they are defined.

Events

class ami.app_meetme_events.**MeetmeJoin**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a user has joined a Meetme bridge.

- 'Channel' : The channel that was bridged
- 'Meetme' : The ID of the Meetme bridge, typically a number formatted as a string
- 'Uniqueid' : An Asterisk unique value
- 'Usernum' : The bridge-specific participant ID assigned to the channel

class ami.app_meetme_events.**MeetmeList**(*response*)
Bases: pystrix.ami.ami._Event

Describes a participant in a Meetme room.

- 'Admin' : 'Yes' or 'No'
- 'CallerIDNum' : The (often) numeric address of the participant

- 'CallerIDName' (optional) : The name of the participant on supporting channels
- 'Channel' : The Asterisk channel in use by the participant
- 'Conference' : The room's identifier
- 'ConnectedLineNum' : unknown
- 'ConnectedLineName' : unknown
- 'MarkedUser' : 'Yes' or 'No'
- 'Muted' : "By admin", "By self", "No"
- 'Role' : "Listen only", "Talk only", "Talk and listen"
- 'Talking' : 'Yes', 'No', or 'Not monitored'
- 'UserNumber' : The ID of the participant in the conference

process()
Translates the 'Admin' and 'MarkedUser' headers' values into bools.

Translates the 'Talking' header's value into a bool, or *None* if not monitored.

Translates the 'UserNumber' header's value into an int, or -1 on failure.

class ami.app_meetme_events.**MeetmeListRooms**(*response*)
Bases: pystrix.ami.ami._Event

Describes a Meetme room.

And, yes, it's plural in Asterisk, too.

- 'Activity' : The duration of the conference
- 'Conference' : The room's identifier
- 'Creation' : 'Dynamic' or 'Static'
- 'Locked' : 'Yes' or 'No'
- 'Marked' : The number of marked users, but not as an integer: 'N/A' or %.4d
- 'Parties' : The number of participants

process()
Translates the 'Parties' header's value into an int, or -1 on failure.

Translates the 'Locked' header's value into a bool.

class ami.app_meetme_events.**MeetmeMute**(*response*)
Bases: pystrix.ami.ami._Event

Indicates that a user has been muted in a Meetme bridge.

- 'Channel' : The channel that was muted
- 'Meetme' : The ID of the Meetme bridge, typically a number formatted as a string
- 'Status' : 'on' or 'off', depending on whether the user was muted or unmuted
- 'Uniqueid' : An Asterisk unique value
- 'Usernum' : The participant ID of the user that was affected

process()
Translates the 'Status' header's value into a bool.

Aggregate Events

class ami.app_meetme_events.**MeetmeList_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all participants have been received in response to a MeetmeList request.

Its members consist of MeetmeList events.

It is finalised by MeetmeListComplete.

class ami.app_meetme_events.**MeetmeListRooms_Aggregate**(*action_id*)
Bases: pystrix.ami.ami._Aggregate

Emitted after all participants have been received in response to a MeetmeListRooms request.

Its members consist of MeetmeListRooms events.

It is finalised by MeetmeListRoomsComplete.

All of these concepts are bound together by the *ami.Manager* class, which provides facilities for sending actions and serving callback handlers when events are received.

3.3 Members

All of the following objects should be accessed as part of the *ami* namespace, regardless of the modules in which they are defined.

3.3.1 Constants

Aside, perhaps, from the "GENERIC" values, to be matched against ami.ami._Message.name responses, these constants are largely unnecessary outside of internal module usage, but they're exposed for convenience's sake.

RESPONSE_GENERIC
A header-value provided as a surrogate for unidentifiable responses

EVENT_GENERIC
A header-value provided as a surrogate for unidentifiable unsolicited events

KEY_ACTION
The header key used to identify an action being requested of Asterisk

KEY_ACTIONID
The header key used to hold the ActionID of a request, for matching with responses

KEY_EVENT
The header key used to hold the event-name of a response

KEY_RESPONSE
The header key used to hold the event-name of a request

3.3.2 Classes

class ami.**Manager**(*debug=False*, *logger=None*, *aggregate_timeout=5*, *orphaned_response_timeout=5*)

close()
Release all resources associated with this manager and ensure that all threads have stopped.

This function is automatically invoked when garbage-collected.

connect(*host*, *port=5038*, *timeout=5*)
Establishes a connection to the specified Asterisk manager, closing any existing connection first.

timeout specifies the number of seconds to allow Asterisk to go between producing lines of a response; it differs from the timeout that may be set on individual requests and exists primarily to avoid having a thread stay active forever, to allow for clean shutdowns.

If the connection fails, *ManagerSocketError* is raised.

disconnect()
Gracefully closes a connection to the Asterisk manager.

If not connected, this is a no-op.

get_asterisk_info()
Provides the name and version of Asterisk as a tuple of strings.

If not connected, *None* is returned.

get_connection()
Returns the current *_SynchronisedSocket* in use by the active connection, or *None* if no manager is attached.

This function is exposed for debugging purposes and should not be used by normal applications that do not have very special reasons for interacting with Asterisk directly.

is_connected()
Indicates whether the manager is connected.

monitor_connection(*interval=2.5*)
Spawns a thread that watches the AMI connection to indicate a disruption when the connection goes down.

interval is the number of seconds to wait between automated Pings to see if Asterisk is still alive; defaults to 2.5.

register_callback(*event*, *function*)
Registers a callback for an Asterisk event identified by *event*, which may be a string for exact matches or a reference to the specific event class.

function is the callable to be invoked whenever a matching *_Event* is emitted; it must accept the positional arguments "event" and "manager", with "event" being the *_Event* object and "manager" being a reference to generating instance of this class.

Registering the same function twice for the same event will unset the first callback, placing the new one at the end of the list.

Registering against the special event *None* will cause the given function to receive all responses not associated with a request, which normally shouldn't exist, but may be observed in practice. Events will not be included.

Registering against the empty string will cause the given function to receive every event, suitable for logging purposes.

Callbacks are executed in the order in which they were registered.

send_action(*request*, *action_id=None*, ***kwargs*)
Sends a command, contained in *request*, a *_Request*, to the Asterisk manager, referred to interchangeably as "actions". Any additional keyword arguments are added directly into the request command as though they were native headers, though the original object is unaffected.

action_id is an optional Asterisk ActionID to use; if unspecified, whatever is in the request, keyed at 'ActionID', is used with the output of *id_generator* being a fallback.

Asterisk's response is returned as a named tuple of the following form, or *None* if the request timed out:

- result: The processed response from Asterisk, nominally the same as *response*; see the specific *_Request* subclass for details in case it provides additional processing
- response: The formatted, but unprocessed, response from Asterisk
- request: The *_Request* object supplied when the request was placed; not a copy of the original
- action_id: The 'ActionID' sent with this request
- success: A boolean value indicating whether the request was met with success
- time: The number of seconds, as a float, that the request took to be serviced
- events: A dictionary containing related events if the request is synchronous or None otherwise
- events_timeout: Whether the request timed out while waiting for events

For forward-compatibility reasons, elements of the tuple should be accessed by name, rather than by index.

Raises *ManagerError* if the manager is not connected.

Raises *ManagerSocketError* if the socket is broken during transmission.

This function is thread-safe.

unregister_callback(*event*, *function*)
Unregisters a previously bound callback.

A boolean value is returned, indicating whether anything was removed.

Internal classes

The following classes are not meant to be worked with directly, but are important for other parts of the system, with members that are worth knowing about.

class ami.ami.**_MessageTemplate**
An abstract base-class for all message-types, including aggregates.

action_id
The Asterisk Action-ID associated with this message, or *None* if undefined, as is the case with unsolicited events.

class ami.ami.**_Aggregate**(*action_id*)
Bases: *ami.ami._MessageTemplate*, dict

Provides, as a dictionary, access to all events that make up the aggregation, keyed by event-class. Repeatable event-types are exposed as lists, while others are direct references to the event itself.

valid
Indicates whether the aggregate is consistent with Asterisk's protocol.

error_message
If *valid* is *False*, this will offer a string explaining why validation failed.

class ami.ami.**_Event**(*response*)
Bases: *ami.ami._Message*

The base-class of any event received from Asterisk, either unsolicited or as part of an extended response-chain.

process()
Provides a tuple containing a copy of all headers as a dictionary and a copy of all response lines. The value of this data is negligible, but subclasses may apply further processing, replacing the values of headers with Python types or making the data easier to work with.

class ami.ami.**_Message**(*response*)
Bases: *ami.ami._MessageTemplate*, dict

The common base-class for both replies and events, this is any structured response received from Asterisk.

All message headers are exposed as dictionary items on this object directly.

data
A series of lines containing the message's payload from Asterisk.

headers
A reference to a dictionary containing all headers associated with this message. Simply treating the message itself as a dictionary for headers is preferred, however; the two methods are equivalent.

raw
The raw response from Asterisk as a series of lines, provided for applications that need access to the original data.

class ami.ami.**_Request**(*action*)
Provides a generic container for assembling AMI requests, the basis of all actions.

Subclasses may override *__init__* and define any additional behaviours they may need, as well as override *process_response()* to specially format the data to be returned after a request has been served.

aggregate
If *True* (*False* by default), an aggregate-event will be generated after the list of independent events generated by this request.

This only has an effect with requests that generate lists of events to begin with and will be ignored in other cases.

synchronous
If *True* (*False* by default), any events generated by this request will be collected and returned in the response.

Synchronous requests will suppress generation of associated asynchronous events and aggregates.

timeout
The number of seconds to wait before considering this request timed out, defaulting to *5*; may be a float.

Indefinite waiting is not supported, but arbitrarily large values may be provided.

A request that has timed out may still be serviced by Asterisk, with the notification being treated as an orphaned event.

Changing the timeout value of the request object has no effect on any previously-sent instances of the request object, since the value is copied at dispatch-time.

class ami.ami.**_Response**

action_id
The action-ID associated with the request that led to the creation of this response.

events
If the corresponding request was *synchronous*, this is a dictionary containing any events emitted in response. If not, this is *None*.

The dictionary will contain references to either the events themselves or lists of events, depending on what is appropriate, keyed by the event's class-object and its friendly name as a string, like *pystrix.ami.core_events.CoreShowChannels* and "CoreShowChannels".

events_timeout
A boolean value indicating whether any events were still unreceived when the response was returned. This is meaningful only if the reqyest had *synchronous* set.

response
The response from Asterisk, without any post-processing applied. You will generally want to use *result* instead, unless you need to see exactly what Asterisk returned.

request
The request object that led to this response. This will be *None* if the response is an orhpan, which may happen when a request times out, but a response is generated anyway, if multiple AMI clients are working with the same Asterisk instance (they won't know each other's action-IDs), or when working with buggy or experimental versions of Asterisk.

result
The response from Asterisk, with post-processing applied to make it easier to work with in Python. This is the attribute about which you will likely care most.

success
A boolean value that indicates whether the response appears to indicate that the request succeeded.

time
The amount of time, as a UNIX timestamp, that elapsed while waiting for a response.

3.3.3 Exceptions

exception ami.**Error**
Bases: exceptions.Exception

The base exception from which all errors native to this module inherit.

exception ami.**ManagerError**
Bases: pystrix.ami.ami.Error

Represents a generic error involving the Asterisk manager.

exception ami.**ManagerSocketError**
Bases: pystrix.ami.ami.Error

Represents a generic error involving the Asterisk connection.

Index

Symbols

_Aggregate (*class in ami.ami*), 62
_Event (*class in ami.ami*), 62
_Message (*class in ami.ami*), 63
_MessageTemplate (*class in ami.ami*), 62
_Request (*class in ami.ami*), 63
__init__() (*ami.app_confbridge.ConfbridgeKick method*), 37
__init__() (*ami.app_confbridge.ConfbridgeList method*), 37
__init__() (*ami.app_confbridge.ConfbridgeListRooms method*), 37
__init__() (*ami.app_confbridge.ConfbridgeLock method*), 37
__init__() (*ami.app_confbridge.ConfbridgeMoHOff method*), 38
__init__() (*ami.app_confbridge.ConfbridgeMoHOn method*), 38
__init__() (*ami.app_confbridge.ConfbridgeMute method*), 38
__init__() (*ami.app_confbridge.ConfbridgePlayFile method*), 38
__init__() (*ami.app_confbridge.ConfbridgeSetSingleVideoSrc method*), 39
__init__() (*ami.app_confbridge.ConfbridgeStartRecord method*), 39
__init__() (*ami.app_confbridge.ConfbridgeStopRecord method*), 39
__init__() (*ami.app_confbridge.ConfbridgeUnlock method*), 38
__init__() (*ami.app_confbridge.ConfbridgeUnmute method*), 38
__init__() (*ami.app_meetme.MeetmeList method*), 39
__init__() (*ami.app_meetme.MeetmeListRooms method*), 39
__init__() (*ami.app_meetme.MeetmeMute method*), 40
__init__() (*ami.app_meetme.MeetmeUnmute method*), 40
__init__() (*ami.core.AGI method*), 23
__init__() (*ami.core.AbsoluteTimeout method*), 22
__init__() (*ami.core.Bridge method*), 23
__init__() (*ami.core.Challenge method*), 23
__init__() (*ami.core.ChangeMonitor method*), 23
__init__() (*ami.core.Command method*), 23
__init__() (*ami.core.CoreShowChannels method*), 24
__init__() (*ami.core.CreateConfig method*), 24
__init__() (*ami.core.DBDel method*), 24
__init__() (*ami.core.DBDelTree method*), 24
__init__() (*ami.core.DBGet method*), 24
__init__() (*ami.core.DBPut method*), 24
__init__() (*ami.core.Events method*), 25
__init__() (*ami.core.ExtensionState method*), 25
__init__() (*ami.core.GetConfig method*), 25
__init__() (*ami.core.Getvar method*), 26
__init__() (*ami.core.Hangup method*), 26
__init__() (*ami.core.ListCategories method*), 26
__init__() (*ami.core.ListCommands method*), 26
__init__() (*ami.core.LocalOptimizeAway method*), 26
__init__() (*ami.core.Login method*), 26
__init__() (*ami.core.Logoff method*), 27
__init__() (*ami.core.ModuleLoad method*), 27
__init__() (*ami.core.Monitor method*), 27
__init__() (*ami.core.MuteAudio method*), 28
__init__() (*ami.core.Originate_Application method*), 28
__init__() (*ami.core.Originate_Context method*), 28
__init__() (*ami.core.Park method*), 29
__init__() (*ami.core.ParkedCalls method*), 29
__init__() (*ami.core.PauseMonitor method*), 29
__init__() (*ami.core.Ping method*), 29
__init__() (*ami.core.PlayDTMF method*), 30
__init__() (*ami.core.QueueAdd method*), 30
__init__() (*ami.core.QueueLog method*), 30
__init__() (*ami.core.QueuePause method*), 30
__init__() (*ami.core.QueuePenalty method*), 30

__init__() (*ami.core.QueueReload method*), 30
__init__() (*ami.core.QueueRemove method*), 31
__init__() (*ami.core.QueueStatus method*), 31
__init__() (*ami.core.QueueSummary method*), 31
__init__() (*ami.core.Redirect method*), 31
__init__() (*ami.core.Reload method*), 31
__init__() (*ami.core.SIPnotify method*), 32
__init__() (*ami.core.SIPpeers method*), 32
__init__() (*ami.core.SIPqualify method*), 32
__init__() (*ami.core.SIPshowpeer method*), 34
__init__() (*ami.core.SIPshowregistry method*), 34
__init__() (*ami.core.SendText method*), 32
__init__() (*ami.core.SetCDRUserField method*), 32
__init__() (*ami.core.Setvar method*), 32
__init__() (*ami.core.Status method*), 34
__init__() (*ami.core.StopMonitor method*), 34
__init__() (*ami.core.UnpauseMonitor method*), 34
__init__() (*ami.core.UpdateConfig method*), 35
__init__() (*ami.core.UserEvent method*), 35
__init__() (*ami.core.VoicemailUsersList method*), 35
__init__() (*ami.dahdi.DAHDIDNDoff method*), 36
__init__() (*ami.dahdi.DAHDIDNDon method*), 36
__init__() (*ami.dahdi.DAHDIDialOffhook method*), 36
__init__() (*ami.dahdi.DAHDIHangup method*), 36
__init__() (*ami.dahdi.DAHDIRestart method*), 36
__init__() (*ami.dahdi.DAHDIShowChannels method*), 37

A

AbsoluteTimeout (*class in ami.core*), 22
action_id (*ami.ami._MessageTemplate attribute*), 62
action_id (*ami.ami._Response attribute*), 63
aggregate (*ami.ami._Request attribute*), 63
AGI (*class in agi*), 17
AGI (*class in ami.core*), 23
AGIAppError, 19
AGIDBError, 17
AGIDeadChannelError, 19
AGIError, 19
AGIException, 18
AGIExec (*class in ami.core_events*), 40
AGIHangup, 19
AGIInvalidCommandError, 19
AGIResultHangup, 19
AGISIGHUPHangup, 19
AGISIGPIPEHangup, 19
AGIUnknownError, 19
AGIUsageError, 19
ami.ami._Response (*built-in class*), 63
Answer (*class in agi.core*), 11
AsyncAGI (*class in ami.core_events*), 41
AUTHTYPE_MD5 (*built-in variable*), 21

B

Bridge (*class in ami.core*), 23

C

Challenge (*class in ami.core*), 23
ChangeMonitor (*class in ami.core*), 23
CHANNEL_ALERTING (*built-in variable*), 9
CHANNEL_BUSY (*built-in variable*), 10
CHANNEL_DIALED (*built-in variable*), 9
CHANNEL_DOWN_AVAILABLE (*built-in variable*), 9
CHANNEL_DOWN_RESERVED (*built-in variable*), 9
CHANNEL_OFFHOOK (*built-in variable*), 9
CHANNEL_REMOTE_ALERTING (*built-in variable*), 10
CHANNEL_UP (*built-in variable*), 10
ChannelStatus (*class in agi.core*), 11
ChannelUpdate (*class in ami.core_events*), 41
clear_script_handlers() (*agi.FastAGIServer method*), 18
close() (*ami.Manager method*), 60
Command (*class in ami.core*), 23
ConfbridgeEnd (*class in ami.app_confbridge_events*), 56
ConfbridgeJoin (*class in ami.app_confbridge_events*), 56
ConfbridgeKick (*class in ami.app_confbridge*), 37
ConfbridgeLeave (*class in ami.app_confbridge_events*), 56
ConfbridgeList (*class in ami.app_confbridge*), 37
ConfbridgeList (*class in ami.app_confbridge_events*), 56
ConfbridgeList_Aggregate (*class in ami.app_confbridge_events*), 58
ConfbridgeListComplete (*class in ami.app_confbridge_events*), 57
ConfbridgeListRooms (*class in ami.app_confbridge*), 37
ConfbridgeListRooms (*class in ami.app_confbridge_events*), 57
ConfbridgeListRooms_Aggregate (*class in ami.app_confbridge_events*), 58
ConfbridgeListRoomsComplete (*class in ami.app_confbridge_events*), 57
ConfbridgeLock (*class in ami.app_confbridge*), 37
ConfbridgeMoHOff (*class in ami.app_confbridge*), 38
ConfbridgeMoHOn (*class in ami.app_confbridge*), 38
ConfbridgeMute (*class in ami.app_confbridge*), 38
ConfbridgePlayFile (*class in ami.app_confbridge*), 38
ConfbridgeSetSingleVideoSrc (*class in ami.app_confbridge*), 39
ConfbridgeStart (*class in ami.app_confbridge_events*), 57

ConfbridgeStartRecord (*class in ami.app_confbridge*), 38
ConfbridgeStopRecord (*class in ami.app_confbridge*), 39
ConfbridgeTalking (*class in ami.app_confbridge_events*), 57
ConfbridgeUnlock (*class in ami.app_confbridge*), 37
ConfbridgeUnmute (*class in ami.app_confbridge*), 38
connect() (*ami.Manager method*), 61
ControlStreamFile (*class in agi.core*), 11
CoreShowChannel (*class in ami.core_events*), 41
CoreShowChannels (*class in ami.core*), 23
CoreShowChannels_Aggregate (*class in ami.core_events*), 53
CoreShowChannelsComplete (*class in ami.core_events*), 42
CreateConfig (*class in ami.core*), 24

D

DAHDIDialOffhook (*class in ami.dahdi*), 36
DAHDIDNDoff (*class in ami.dahdi*), 36
DAHDIDNDon (*class in ami.dahdi*), 36
DAHDIHangup (*class in ami.dahdi*), 36
DAHDIRestart (*class in ami.dahdi*), 36
DAHDIShowChannels (*class in ami.dahdi*), 36
DAHDIShowChannels (*class in ami.dahdi_events*), 55
DAHDIShowChannels_Aggregate (*class in ami.dahdi_events*), 55
DAHDIShowChannelsComplete (*class in ami.dahdi_events*), 55
data (*ami.ami._Message attribute*), 63
DatabaseDel (*class in agi.core*), 11
DatabaseDeltree (*class in agi.core*), 11
DatabaseGet (*class in agi.core*), 12
DatabasePut (*class in agi.core*), 12
DBDel (*class in ami.core*), 24
DBDelTree (*class in ami.core*), 24
DBGet (*class in ami.core*), 24
DBGetResponse (*class in ami.core_events*), 42
DBPut (*class in ami.core*), 24
disconnect() (*ami.Manager method*), 61
DTMF (*class in ami.core_events*), 42

E

Error, 64
error_message (*ami.ami._Aggregate attribute*), 62
EVENT_GENERIC (*built-in variable*), 60
EVENTMASK_ALL (*built-in variable*), 21
EVENTMASK_CALL (*built-in variable*), 21
EVENTMASK_LOG (*built-in variable*), 22
EVENTMASK_NONE (*built-in variable*), 21
EVENTMASK_SYSTEM (*built-in variable*), 22
events (*ami.ami._Response attribute*), 63
Events (*class in ami.core*), 24
events_timeout (*ami.ami._Response attribute*), 64
Exec (*class in agi.core*), 12
execute() (*agi.AGI method*), 17
ExtensionState (*class in ami.core*), 25

F

FastAGIServer (*class in agi*), 18
FORMAT_ALAW (*built-in variable*), 22
FORMAT_G723 (*built-in variable*), 22
FORMAT_G729 (*built-in variable*), 22
FORMAT_GSM (*built-in variable*), 22
FORMAT_SLN (*built-in variable*), 22
FORMAT_ULAW (*built-in variable*), 22
FORMAT_VOX (*built-in variable*), 22
FORMAT_WAV (*built-in variable*), 22
FullyBooted (*class in ami.core_events*), 42

G

get_asterisk_info() (*ami.Manager method*), 61
get_connection() (*ami.Manager method*), 61
get_environment() (*agi.AGI method*), 18
get_lines() (*ami.core.GetConfig method*), 25
get_script_handler() (*agi.FastAGIServer method*), 18
GetConfig (*class in ami.core*), 25
GetData (*class in agi.core*), 12
GetFullVariable (*class in agi.core*), 12
GetOption (*class in agi.core*), 12
Getvar (*class in ami.core*), 25
GetVariable (*class in agi.core*), 13

H

handle_request() (*agi.FastAGIServer method*), 18
Hangup (*class in agi.core*), 13
Hangup (*class in ami.core*), 26
Hangup (*class in ami.core_events*), 43
HangupRequest (*class in ami.core_events*), 43
headers (*ami.ami._Message attribute*), 63

I

is_connected() (*ami.Manager method*), 61
items (*agi.AGIException attribute*), 18

K

KEY_ACTION (*built-in variable*), 60
KEY_ACTIONID (*built-in variable*), 60
KEY_EVENT (*built-in variable*), 60
KEY_RESPONSE (*built-in variable*), 60

L

ListCategories (*class in ami.core*), 26

ListCommands (*class in ami.core*), 26
LocalOptimizeAway (*class in ami.core*), 26
LOG_CRITICAL (*built-in variable*), 10
LOG_DEBUG (*built-in variable*), 10
LOG_ERROR (*built-in variable*), 10
LOG_INFO (*built-in variable*), 10
LOG_WARN (*built-in variable*), 10
Login (*class in ami.core*), 26
Logoff (*class in ami.core*), 26

M

Manager (*class in ami*), 60
ManagerAuthError, 36
ManagerError, 64
ManagerSocketError, 64
MeetmeJoin (*class in ami.app_meetme_events*), 58
MeetmeList (*class in ami.app_meetme*), 39
MeetmeList (*class in ami.app_meetme_events*), 58
MeetmeList_Aggregate (*class in ami.app_meetme_events*), 60
MeetmeListRooms (*class in ami.app_meetme*), 39
MeetmeListRooms (*class in ami.app_meetme_events*), 59
MeetmeListRooms_Aggregate (*class in ami.app_meetme_events*), 60
MeetmeMute (*class in ami.app_meetme*), 40
MeetmeMute (*class in ami.app_meetme_events*), 59
MeetmeUnmute (*class in ami.app_meetme*), 40
ModuleLoad (*class in ami.core*), 27
Monitor (*class in ami.core*), 27
monitor_connection() (*ami.Manager method*), 61
MonitorStart (*class in ami.core_events*), 43
MonitorStop (*class in ami.core_events*), 43
MuteAudio (*class in ami.core*), 28

N

NewAccountCode (*class in ami.core_events*), 43
Newchannel (*class in ami.core_events*), 43
Newexten (*class in ami.core_events*), 44
Newstate (*class in ami.core_events*), 44
Noop (*class in agi.core*), 13

O

Originate_Application (*class in ami.core*), 28
Originate_Context (*class in ami.core*), 28
ORIGINATE_RESULT_ANSWERED (*built-in variable*), 22
ORIGINATE_RESULT_BUSY (*built-in variable*), 22
ORIGINATE_RESULT_CONGESTION (*built-in variable*), 22
ORIGINATE_RESULT_INCOMPLETE (*built-in variable*), 22
ORIGINATE_RESULT_REJECT (*built-in variable*), 22
ORIGINATE_RESULT_RING_LOCAL (*built-in variable*), 22
ORIGINATE_RESULT_RING_REMOTE (*built-in variable*), 22
OriginateResponse (*class in ami.core_events*), 45

P

Park (*class in ami.core*), 29
ParkedCall (*class in ami.core_events*), 45
ParkedCalls (*class in ami.core*), 29
ParkedCalls_Aggregate (*class in ami.core_events*), 53
ParkedCallsComplete (*class in ami.core_events*), 45
PauseMonitor (*class in ami.core*), 29
PeerEntry (*class in ami.core_events*), 45
PeerlistComplete (*class in ami.core_events*), 46
Ping (*class in ami.core*), 29
PlayDTMF (*class in ami.core*), 29
process() (*ami.ami._Event method*), 62
process() (*ami.app_confbridge_events.ConfbridgeList method*), 57
process() (*ami.app_confbridge_events.ConfbridgeListComplete method*), 57
process() (*ami.app_confbridge_events.ConfbridgeListRooms method*), 57
process() (*ami.app_confbridge_events.ConfbridgeListRoomsComplete method*), 57
process() (*ami.app_confbridge_events.ConfbridgeTalking method*), 58
process() (*ami.app_meetme_events.MeetmeList method*), 59
process() (*ami.app_meetme_events.MeetmeListRooms method*), 59
process() (*ami.app_meetme_events.MeetmeMute method*), 59
process() (*ami.core_events.AGIExec method*), 41
process() (*ami.core_events.CoreShowChannel method*), 42
process() (*ami.core_events.CoreShowChannelsComplete method*), 42
process() (*ami.core_events.DTMF method*), 42
process() (*ami.core_events.Hangup method*), 43
process() (*ami.core_events.Newchannel method*), 44
process() (*ami.core_events.Newstate method*), 45
process() (*ami.core_events.OriginateResponse method*), 45
process() (*ami.core_events.ParkedCall method*), 45
process() (*ami.core_events.ParkedCallsComplete method*), 45
process() (*ami.core_events.PeerEntry method*), 46
process() (*ami.core_events.PeerlistComplete method*), 46
process() (*ami.core_events.QueueEntry method*), 46

process() (*ami.core_events.QueueMember method*), 47
process() (*ami.core_events.QueueMemberAdded method*), 47
process() (*ami.core_events.QueueMemberPaused method*), 48
process() (*ami.core_events.QueueParams method*), 48
process() (*ami.core_events.QueueSummary method*), 49
process() (*ami.core_events.RegistrationsComplete method*), 49
process() (*ami.core_events.RegistryEntry method*), 49
process() (*ami.core_events.RTCPReceived method*), 50
process() (*ami.core_events.RTCPSent method*), 51
process() (*ami.core_events.Shutdown method*), 51
process() (*ami.core_events.Status method*), 52
process() (*ami.core_events.StatusComplete method*), 52
process() (*ami.core_events.VoicemailUserEntry method*), 53
process() (*ami.dahdi_events.DAHDIShowChannels method*), 55
process() (*ami.dahdi_events.DAHDIShowChannelsComplete method*), 55

Q

QueueAdd (*class in ami.core*), 30
QueueEntry (*class in ami.core_events*), 46
QueueLog (*class in ami.core*), 30
QueueMember (*class in ami.core_events*), 46
QueueMemberAdded (*class in ami.core_events*), 47
QueueMemberPaused (*class in ami.core_events*), 48
QueueMemberRemoved (*class in ami.core_events*), 48
QueueParams (*class in ami.core_events*), 48
QueuePause (*class in ami.core*), 30
QueuePenalty (*class in ami.core*), 30
QueueReload (*class in ami.core*), 30
QueueRemove (*class in ami.core*), 31
QueueStatus (*class in ami.core*), 31
QueueStatus_Aggregate (*class in ami.core_events*), 54
QueueStatusComplete (*class in ami.core_events*), 48
QueueSummary (*class in ami.core*), 31
QueueSummary (*class in ami.core_events*), 48
QueueSummary_Aggregate (*class in ami.core_events*), 54
QueueSummaryComplete (*class in ami.core_events*), 49

R

raw (*ami.ami._Message attribute*), 63
ReceiveChar (*class in agi.core*), 13
ReceiveText (*class in agi.core*), 13
RecordFile (*class in agi.core*), 13
Redirect (*class in ami.core*), 31
register_callback() (*ami.Manager method*), 61
register_script_handler() (*agi.FastAGIServer method*), 18
RegistrationsComplete (*class in ami.core_events*), 49
RegistryEntry (*class in ami.core_events*), 49
Reload (*class in ami.core*), 31
Reload (*class in ami.core_events*), 50
request (*ami.ami._Response attribute*), 64
response (*ami.ami._Response attribute*), 64
RESPONSE_GENERIC (*built-in variable*), 60
result (*ami.ami._Response attribute*), 64
RTCPReceived (*class in ami.core_events*), 50
RTCPSent (*class in ami.core_events*), 50

S

SayAlpha (*class in agi.core*), 14
SayDate (*class in agi.core*), 14
SayDatetime (*class in agi.core*), 14
SayDigits (*class in agi.core*), 15
SayNumber (*class in agi.core*), 15
SayPhonetic (*class in agi.core*), 15
SayTime (*class in agi.core*), 15
send_action() (*ami.Manager method*), 61
SendImage (*class in agi.core*), 15
SendText (*class in agi.core*), 15
SendText (*class in ami.core*), 31
serve_forever() (*agi.FastAGIServer method*), 18
SetAutohangup (*class in agi.core*), 15
SetCallerid (*class in agi.core*), 16
SetCDRUserField (*class in ami.core*), 32
SetContext (*class in agi.core*), 16
SetExtension (*class in agi.core*), 16
SetMusic (*class in agi.core*), 16
SetPriority (*class in agi.core*), 16
Setvar (*class in ami.core*), 32
SetVariable (*class in agi.core*), 16
Shutdown (*class in ami.core_events*), 51
shutdown() (*agi.FastAGIServer method*), 18
SIPnotify (*class in ami.core*), 32
SIPpeers (*class in ami.core*), 32
SIPpeers_Aggregate (*class in ami.core_events*), 54
SIPqualify (*class in ami.core*), 32
SIPshowpeer (*class in ami.core*), 33
SIPshowregistry (*class in ami.core*), 34
SIPshowregistry_Aggregate (*class in ami.core_events*), 54
SoftHangupRequest (*class in ami.core_events*), 51

Status (*class in ami.core*), 34
Status (*class in ami.core_events*), 51
Status_Aggregate (*class in ami.core_events*), 54
StatusComplete (*class in ami.core_events*), 52
StopMonitor (*class in ami.core*), 34
StreamFile (*class in agi.core*), 16
success (*ami.ami._Response attribute*), 64
synchronous (*ami.ami._Request attribute*), 63

T

TDD_MATE (*built-in variable*), 10
TDD_OFF (*built-in variable*), 10
TDD_ON (*built-in variable*), 10
TDDMode (*class in agi.core*), 16
time (*ami.ami._Response attribute*), 64
timeout (*agi.FastAGIServer attribute*), 18
timeout (*ami.ami._Request attribute*), 63

U

UnpauseMonitor (*class in ami.core*), 34
unregister_callback() (*ami.Manager method*), 62
unregister_script_handler() (*agi.FastAGIServer method*), 18
UpdateConfig (*class in ami.core*), 35
UserEvent (*class in ami.core*), 35
UserEvent (*class in ami.core_events*), 52

V

valid (*ami.ami._Aggregate attribute*), 62
VarSet (*class in ami.core_events*), 52
Verbose (*class in agi.core*), 17
VoicemailUserEntry (*class in ami.core_events*), 52
VoicemailUserEntryComplete (*class in ami.core_events*), 53
VoicemailUsersList (*class in ami.core*), 35
VoicemailUsersList_Aggregate (*class in ami.core_events*), 54

W

WaitForDigit (*class in agi.core*), 17