cnxman Documentation

Release 0.0.2

Pat Daburu

Jan 06, 2020

Contents:

CHAPTER 1

API Documentation

This is a simple framework for managing connections to things.

1.1 cnxman.basics

This module contains the base classes and basic utilities.

class cnxman.basics.**Connection**
Bases: object

Extend this class to define a logical connection to something. The expectations we have of a connection are these:

- It can attempt create a connection and report on whether or not the connection was successful.
- It can (at least by all appearances) gracefully disconnect.
- It can release all its resources upon request.

Seealso *Connection.try_connect()*

Seealso *Connection.disconnect()*

Seealso *Connection.teardown()*

class Signals
Bases: enum.Enum

These are the used by connection objects.

Seealso pydispatch.dispatcher()

RAISE_ALARM = 'raise-alarm'

__init__
Initialize self. See help(type(self)) for accurate signature.

disconnect()
Override this method to take the steps required to gracefully disconnect.

raise_alarm()
Raise the alarm to notify anyone who might be interested (like a *ConnectionManager*) that there is trouble with the connection.

teardown()
Override this method to release resources when requested.

try_connect() → bool
Override this method to define the logic by which a connection is make.

Returns `True` if and only if the connection attempt is successful, otherwise `False`.

Return type `bool`

exception cnxman.basics.**ConnectionException**(*message: str*, *inner: Exception*)
Bases: `Exception`

Raised when an error occurs within a connection.

__init__(*message: str*, *inner: Exception*)

Parameters

- **message** (`str`) – the original message
- **inner** (`Exception`) – the exception responsible for the raising of this exception.

args

static from_exception(*ex: Exception*)
This is a convenience method that can be used to create a connection exception from another exception, using default logic to populate the constructor arguments.

Parameters **ex** (`Exception`) – the original exception

Returns a new connection exception

Return type *ConnectionException*

inner
This is the original exception responsible for raising this connection exception.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class cnxman.basics.**ConnectionManager**(*connection: cnxman.basics.Connection*)
Bases: `object`

Extend this class to create your own object with the know-how to establish and maintain a connection to something.

__init__(*connection: cnxman.basics.Connection*)

connect
An input for a L{MethodicalMachine}.

connected = MethodicalState(method=<function ConnectionManager.connected>)

connecting = MethodicalState(method=<function ConnectionManager.connecting>)

disconnect
An input for a L{MethodicalMachine}.

disconnected = MethodicalState(method=<function ConnectionManager.disconnected>)

ready = MethodicalState(method=<function ConnectionManager.ready>)

recovering = MethodicalState(method=<function ConnectionManager.recovering>)

teardown
An input for a L{MethodicalMachine}.

torndown = MethodicalState(method=<function ConnectionManager.torndown>)

1.2 cnxman.serial

Let's manage serial port connections!

class cnxman.serial.**SerialConnection**(*port: str*, *baudrate: int = 9600*, *bytesize: int = 8*, *parity: str = 'N'*, *stopbits: int = 1*, *timeout=None*)
Bases: *cnxman.basics.Connection*

class Signals
Bases: enum.Enum

These are the used by serial listener objects.

Seealso pydispatch.dispatcher()

DATA_RECEIVED = 'data-received'

__init__(*port: str*, *baudrate: int = 9600*, *bytesize: int = 8*, *parity: str = 'N'*, *stopbits: int = 1*, *timeout=None*)

disconnect()
Disconnect from the serial port.

logger = <Logger cnxman.serial.SerialConnection (NOTSET)>

raise_alarm()
Raise the alarm to notify anyone who might be interested (like a ConnectionManager) that there is trouble with the connection.

teardown()
Release the serial port entirely.

try_connect() → bool
Attempt to connect to the serial port.

Returns True if and only if the connection attempt is successful, otherwise False.

Return type bool

class cnxman.serial.**SerialListener**(*serial: serial.serialposix.Serial*)
Bases: threading.Thread

This is a thread object that listens for incoming data from a serial connection.

class Signals
Bases: enum.Enum

These are the used by serial listener objects.

Seealso pydispatch.dispatcher()

DATA_RECEIVED = 'data-received'

READ_ERROR = 'read-error'

__init__(*serial: serial.serialposix.Serial*)

daemon
A boolean value indicating whether this thread is a daemon thread.

This must be set before start() is called, otherwise RuntimeError is raised. Its initial value is inherited from the creating thread; the main thread is not a daemon thread and therefore all threads created in the main thread default to daemon = False.

The entire Python program exits when no alive non-daemon threads are left.

getName()

ident
Thread identifier of this thread or None if it has not been started.

This is a nonzero integer. See the get_ident() function. Thread identifiers may be recycled when a thread exits and another thread is created. The identifier is available even after the thread has exited.

isAlive()
Return whether the thread is alive.

This method is deprecated, use is_alive() instead.

isDaemon()

is_alive()
Return whether the thread is alive.

This method returns True just before the run() method starts until just after the run() method terminates. The module function enumerate() returns a list of all alive threads.

join(*timeout=None*)
Wait until the thread terminates.

This blocks the calling thread until the thread whose join() method is called terminates – either normally or through an unhandled exception or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof). As join() always returns None, you must call is_alive() after join() to decide whether a timeout happened – if the thread is still alive, the join() call timed out.

When the timeout argument is not present or None, the operation will block until the thread terminates.

A thread can be join()ed many times.

join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to join() a thread before it has been started and attempts to do so raises the same exception.

name
A string used for identification purposes only.

It has no semantics. Multiple threads may be given the same name. The initial name is set by the constructor.

run()
Start listening for data on the serial connection.

serial
This is the serial object we're monitoring.

Return type `pyserial.Serial`

setDaemon(*daemonic*)

setName(*name*)

start()
Start the thread's activity.

It must be called at most once per thread object. It arranges for the object's run() method to be invoked in a separate thread of control.

This method will raise a RuntimeError if called more than once on the same thread object.

terminate()
Terminate the listener.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

T

W