cthread

Release 1.2.1

Kieran Sonter

Mar 31, 2020

DOCUMENTATION:

cthread (*ControllableThread*) is a Python library that is built upon the `threading.Thread` class. *cthread* provides additional functionality to the standard `threading.Thread` library by allowing the threads to be started, paused, resumed, reset, and killed.

CHAPTER ONE

INSTALLATION

For Python 2/3:

```
>>> pip install cthread
```

CHAPTER

TWO

USAGE

See the Quickstart and Alternative States examples listed in *Examples*.

2.1 Module (`cthread`)

The documentation of the `cthread` package is outlined below.

2.1.1 Exceptions

The custom exceptions that `cthread` can raise are documented below.

exception `cthread.`**`CThreadException`**(*message=None*, **args*, ***kwargs*)

Bases: `Exception`

Base class of any ControllableThread exception.

All exceptions that are thrown by the `cthread` module inherit from this exception. This specific exception instance is however never raised.

Parameters **`message`** (*`str, optional`*) – Information about the exception that was raised.

exception `cthread.`**`CallbackNotImplemented`**(*message=None*, **args*, ***kwargs*)

Bases: `cthread.cthread.CThreadException`

A state callback function has not been implemented by the child class.

This exception is raised if a `cthread.ControllableThread` is created without overwriting a particular state callback function.

Parameters **`message`** (*`str, optional`*) – Information about the exception that was raised.

exception `cthread.`**`InvalidArgument`**(*message=None*, **args*, ***kwargs*)

Bases: `cthread.cthread.CThreadException`

Base class of any invalid argument exception.

This exception is raised if an invalid argument is input to any publicly accessible `cthread` method. This specific exception instance is however never raised.

Parameters **`message`** (*`str, optional`*) – Information about the exception that was raised.

exception `cthread.`**`InvalidState`**(*message=None*, **args*, ***kwargs*)
Bases: `cthread.cthread.InvalidArgument`

Base class of an unrecognised thread state.

This exception is raised if a thread state is not recognised. While this exception is a base class of an unrecognised thread state, this specific instance could also be raised.

Parameters **`message`** (`str`, *`optional`*) – Information about the exception that was raised.

exception `cthread.`**`InvalidMaxState`**(*message=None*, **args*, ***kwargs*)
Bases: `cthread.cthread.InvalidState`

Maximum thread state is not recognised.

This exception is raised if the maximum thread state is not recognised.

Parameters **`message`** (`str`, *`optional`*) – Information about the exception that was raised.

exception `cthread.`**`InvalidAlternativeState`**(*message=None*, **args*, ***kwargs*)
Bases: `cthread.cthread.InvalidState`

Alternative thread state is not recognised.

This exception is raised if the required updated state of the thread is not a registered alternative state.

Parameters **`message`** (`str`, *`optional`*) – Information about the exception that was raised.

exception `cthread.`**`InvalidCallback`**(*message=None*, **args*, ***kwargs*)
Bases: `cthread.cthread.InvalidArgument`

No alternative state callback functions supplied.

This exception is raised if the thread is initialised with alternative states but not alternative state callback functions.

Parameters **`message`** (`str`, *`optional`*) – Information about the exception that was raised.

exception `cthread.`**`InvalidName`**(*message=None*, **args*, ***kwargs*)
Bases: `cthread.cthread.InvalidArgument`

Desired name of the thread is not a string.

This exception is raised if the name of the thread is not a string.

Parameters **`message`** (`str`, *`optional`*) – Information about the exception that was raised.

exception `cthread.`**`InvalidQueue`**(*message=None*, **args*, ***kwargs*)
Bases: `cthread.cthread.InvalidArgument`

No communication method to the initialising thread.

This exception is raised if an instance of a *`cthread.ControllableThread`* is initialised without a means of communicating with the thread that initialises it.

Parameters **`message`** (`str`, *`optional`*) – Information about the exception that was raised.

2.1.2 Classes

The classes contained within the *`cthread`* package are documented below.

class `cthread.`**`ControllableThread`**(*name*, *queue*, ***kwargs*)
Bases: `threading.Thread`

Parent class for any *`cthread.ControllableThread`*.

Allows threads to be killed, paused and resumed, and allows for direct communication to the main initialising thread.

Parameters

- **`name`** (*`str`*) – Name of the thread.
- **`queue`** (`queue.Queue`) – Priority queue for communication to the main thread.

Raises

- ***`cthread.InvalidName`*** – If name is not a string.
- ***`cthread.InvalidQueue`*** – If queue is not a Queue.
- ***`cthread.InvalidCallback`*** – If kwargs are not a dictionary of functions.

`alt`(*name*)

Updates the state of the thread to a registered alternative state.

Note that the state must be a registered alternative state in order for the thread to be updated to the desired state.

Parameters **`name`** (*`str`*) – Name of the state to which the thread will be updated.

Raises ***`cthread.InvalidAlternativeState`*** – If the required updated state of the thread is not a registered alternative state.

`kill`()

Updates the state of the thread to *`ThreadState.KILLED`*.

`pause`()

Updates the state of the thread to *`ThreadState.PAUSED`*.

`reset`()

Updates the state of the thread to *`ThreadState.STARTED`*.

`resume`()

Updates the state of the thread to *`ThreadState.RESUMED`*.

`run`()

Entry point for the thread.

Contains the cyclic executive of the thread. There are at least six possible states for the thread:

1. STARTED
2. ACTIVE
3. IDLE
4. PAUSED
5. RESUMED
6. KILLED
7. Any alternative individual thread-specific states.

Each of these states has a callback function that must be implemented in any of the child threads. Of course, the alterative state callback should not be implemented if there are no alternative states.

This cyclic executive will execute as long as the thread is not killed.

Raises ***`cthread.CallbackNotImplemented`*** – If any of the following callback functions were not overwritten: `cthread.ControllableThread._started_callback()`, `cthread.ControllableThread._active_callback()`, `cthread.ControllableThread._paused_callback()`, `cthread.ControllableThread._resumed_callback()`, or `cthread.ControllableThread._killed_callback()`.

class `cthread.`**`ThreadState`**

Bases: `object`

Represents the state of the thread.

`STARTED`

Numerical encoding of the 'started' thread state.

Type int

`ACTIVE`

Numerical encoding of the 'active' thread state.

Type int

`IDLE`

Numerical encoding of the 'idle' thread state.

Type int

`PAUSED`

Numerical encoding of the 'paused' thread state.

Type int

`RESUMED`

Numerical encoding of the 'resumed' thread state.

Type int

`KILLED`

Numerical encoding of the 'killed' thread state.

Type int

`get_state``()`

Gets the state of the thread.

Returns

State of the thread. There are at least six possible return values:

- *`ThreadState.STARTED`*: if the thread is being initialised,
- *`ThreadState.ACTIVE`*: if the thread is currently running,
- *`ThreadState.IDLE`*: if the thread is currently not running,
- *`ThreadState.PAUSED`*: if the thread is transitioning from running to not running.
- *`ThreadState.RESUMED`*: if the thread is transitioning from not running to running.
- *`ThreadState.KILLED`*: if the thread is in the process of terminating, and
- Any other user-defined thread states.

Return type int

`update_max_state``(`*maxState*`)`

Updates the maximum state of the thread.

The *maxState* value is used to determine the validity of a state to supplied to the `cthread.ThreadState.update_state()` function. The state of the thread must be within a predefined range, or else it is an invalid state. The *maxState* is also not a fixed constant because the user can define their own states, and hence the validity check of a state must accomodate these user-defined states.

Parameters **`maxState`** (*`int`*) – Maximum state that the thread can take.

Raises ***`cthread.InvalidState`*** – If the maximum state is <= *`ThreadState.KILLED`*.

`update_state`(*state*)

Updates the state of the thread.

Parameters **`state`** (*`int`*) – New state of the thread.

Raises ***`cthread.InvalidState`*** – If *state* < *`ThreadState.STARTED`* or *state* > `ThreadState._maxState`

2.2 Examples

Note: In order for the *`cthread`* package to be fully functional, the developer must follow two coding practices:

1. Any code within the `cthread.ControllableThread._active_callback()` callback function and any registered alternative state callback function must be:
   - Non-blocking (at the very least it must only block for a short period of time), and
   - Contain no indefinite `while` or `for` loops.

   In short, the `cthread.ControllableThread._active_callback()` callback function and any registered alternative state callback functions must be written to minimize its execution time. This allows the thread to be controllable. This is because it is only at the end of each callback function that a check is conducted to determine whether the state of the thread must be updated.

2. The code within the `cthread.ControllableThread._started_callback()` `cthread.ControllableThread._paused_callback()`, `cthread.ControllableThread._resumed_callback()`, and `cthread.ControllableThread._killed_callback()` callback functions will only execute once. There is a wrapper function for each callback function within the *`cthread.ControllableThread`* class. This wrapper function automatically updates the thread state to *`cthread.ThreadState.ACTIVE`* at the completion of the `cthread.ControllableThread._started_callback()` and `cthread.ControllableThread._resumed_callback()` callback functions, and *`cthread.ThreadState.IDLE`* at the completion of the `cthread.ControllableThread._paused_callback()` callback function. These callbacks should be written in such a way that they pause or resume/start thread functionality.

2.2.1 Quickstart

This code example can be found in the file `examples/quickstart.py`. It is an example of how to create a class that inherits from *`ControllableThread`*. No functionality has been added to any of the states. The lines:

```
        # Add any state specific code here (and remove the 'pass') #
        pass
```

can be replaced in each of the state callback functions with functional state code that is specific to the user's needs.

```
import cthread
import logging
import queue as q
import time

# Configure the logger #
logging.basicConfig(level=logging.INFO,
        format='%(asctime)s %(levelname)s %(name)s %(message)s',
        datefmt='%Y-%m-%d %H:%M:%S')

class Quickstart(cthread.ControllableThread):

    def _started_callback(self):
        # Add any state specific code here (and remove the 'pass') #
        pass

    def _active_callback(self):
        # Add any state specific code here (and remove the 'pass') #
        pass

    def _paused_callback(self):
        # Add any state specific code here (and remove the 'pass') #
        pass

    def _resumed_callback(self):
        # Add any state specific code here (and remove the 'pass') #
        pass

    def _killed_callback(self):
        # Add any state specific code here (and remove the 'pass') #
        pass

if __name__ == "__main__":

    queue = q.PriorityQueue()
    quickstart = Quickstart(name="QuickstartThread", queue=queue)

    quickstart.start() # Start the thread.  MUST be called first #
    time.sleep(1)

    quickstart.pause() # Pause the thread. #
    time.sleep(1)

    quickstart.resume() # Resume the thread #
    time.sleep(1)

    quickstart.reset() # Reset the thread #
    time.sleep(1)

    quickstart.kill() # Kill the thread #
```

This code gives the following output:

```
>> Starting thread: QuickstartThread...
>> (Re)initialising thread...
>> Thread activated!
>> Pausing thread...
>> Thread paused!
>> Resuming thread...
>> Thread activated!
>> (Re)initialising thread...
>> Thread activated!
>> Killing thread...
>> Stopped thread: QuickstartThread!
```

2.2.2 Alternative Thread states

This code example can be found in the file `examples/alternative_state.py`. It is an example of how to create a class that inherits from *ControllableThread* with additional allowable thread states. No functionality has been added to any of the additional states. The lines:

```
        # 2. Add any alternative state specific code here #
        pass
```

can be replaced in each of the additional state callback functions with functional state code that is specific to the user's needs.

Note: There are two steps that must be followed when creating a *cthread.ControllableThread* instance with alternative states:

1. Initialise the *cthread.ControllableThread* parent class with additional kwargs of the form **name: callback**. Each state name and callback function must be unique.
2. Supply a callback function for each alternative state with the same name as that which was passed to the parent class in step 1 above.

These two steps are highlighted in the proceeding code.

Then, in order to transition the thread into one of the alternative states, the public method *cthread.ControllableThread.run()* must be called with the `name` parameter identical to the name specified in the kwargs from step 1 above.

```
import cthread
import logging
import queue as q
import time

# Configure the logger #
logging.basicConfig(level=logging.INFO,
        format='%(asctime)s %(levelname)s %(name)s %(message)s',
        datefmt='%Y-%m-%d %H:%M:%S')

class AlternativeState(cthread.ControllableThread):

    def __init__(self, queue):
        # 1. Initialise ControllableThread with additional kwargs #
        cthread.ControllableThread.__init__(self,
```

(continues on next page)

(continued from previous page)

```python
                name="AlternativeStateThread",
                queue=queue,
                ALT1=self._alt1_callback,
                ALT2=self._alt2_callback
        )

    def _started_callback(self):
        pass

    def _active_callback(self):
        pass

    def _paused_callback(self):
        pass

    def _resumed_callback(self):
        pass

    def _killed_callback(self):
        pass

    def _alt1_callback(self):
        # 2. Add any alternative state specific code here #
        pass

    def _alt2_callback(self):
        # 2. Add any alternative state specific code here #
        pass

if __name__ == "__main__":

    queue = q.PriorityQueue()
    quickstart = AlternativeState(queue)

    quickstart.start() # Start the thread.  MUST be called first #
    time.sleep(1)

    quickstart.alt(name="ALT1") # ALT1 state #
    time.sleep(1)

    quickstart.alt(name="ALT2") # ALT2 state #
    time.sleep(1)

    quickstart.kill() # Kill the thread #
```

This code gives the following output:

```
>> Starting thread: Starting thread: AlternativeStateThread...
>> (Re)initialising thread...
>> Thread activated!
>> Transitioning thread to ALT1 state...
>> Transitioning thread to ALT2 state....
>> Killing thread...
>> Stopped thread: AlternativeStateThread!
```

2.3 MIT License

2.4 Change Log

2.4.1 1.2.1 (2019-05-08)

Features

- Added the additional classes to the package (imported into __init__.py) that were released as part of version 1.2.

2.4.2 1.2 (2019-05-07)

Features

- Added examples
- Added wrapper functions to the callback functions for logging purposes and automatic state transition purposes
- Added alternative state registration process
- Additional exceptions

2.4.3 1.1 (2019-05-07)

Features

- Updated Google docstrings to be compatible with Sphinx RST formatting.
- Finalised documentation for the package classes and exceptions.

2.4.4 1.0 (2019-05-05)

Features

- Removed time delay in the cyclic executive of the thread.

2.4.5 0.0.1 (2019-05-01)

Features

- Initial release

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

INDEX