# QConnectBase

# v. 1.1.0

Nguyen Huynh Tri Cuong

05.07.2022

# Contents

# Chapter 1

# Introduction

QConnectBaseLibrary is a connection testing library for RobotFramework. Library will be supported to downloaded from PyPI soon. It provides a mechanism to handle trace log continously receiving from a connection (such as Raw TCP, SSH, Serial, etc.) besides sending data back to the other side. It's especially efficient for monitoring the overflood response trace log from an asynchronous trace systems. It is supporting Python 3.7+ and RobotFramework 3.2+.

# Chapter 2

# Description

*QConnectBase*

## 2.1 Getting Started

We have a plan to publish all the sourcecode as OSS in the near future so that you can downloaded from PyPI. For the current period, you can checkout

QConnectBaseLibrary

After checking out the source completely, you can install by running below command inside **robotframework-qconnect-base** directory.

```
python setup.py install
```

## 2.2 Usage

QConnectBase Library support following keywords for testing connection in RobotFramework.

### 2.2.1 connect

**Use for establishing a connection.**

**Syntax**:

**connect** `[conn_name]` `[conn_type]` `[conn_mode]` `[conn_conf]` *(All parameters are required to be in order)* or

**connect** `conn_name=[conn_name]` `conn_type=[conn_type]` `conn_mode=[conn_mode]` `conn_conf=[conn_conf]` *(All parameters are assigned by name)*

**Arguments**:

**conn_name**: Name of the connection.

**conn_type**: Type of the connection. QConnectBaseLibrary has supported below connection types:

- **TCPIPClient**: Create a Raw TCPIP connection to TCP Server.
- **SSHClient**: Create a client connection to a SSH server.
- **SerialClient**: Create a client connection via Serial Port.

**conn_mode**: (unused) Mode of a connection type.

**conn_conf**: Configurations for making a connection. Depend on **conn_type** (Type of Connection), there is a suitable configuration in JSON format as below.

- **TCPIPClient**

```
{
    "address": [server host], # Optional. Default value is ↩
    ↪ "localhost".
    "port": [server port]     # Optional. Default value is 1234.
    "logfile": [Log file path. Possible values: 'nonlog', ↩
    ↪ 'console', [user define path] ]
 }
```

- **SSHClient**

```
{
    "address" : [server host],  # Optional. Default value is ↩
    ↪ "localhost".
    "port" : [server host],     # Optional. Default value is 22.
    "username" : [username],    # Optional. Default value is "root".
    "password" : [password],    # Optional. Default value is "".
    "authentication" : "password" | "keyfile" | ↩
    ↪ "passwordkeyfile",  # Optional. Default value is "".
    "key_filename" : [filename or list of filenames], # ↩
    ↪ Optional. Default value is null.
    "logfile": [Log file path. Possible values: 'nonlog', ↩
    ↪ 'console', [user define path] ]
 }
```

- **SerialClient**

```
{
    "port" : [comport or null],
    "baudrate" : [Baud rate such as 9600 or 115200 etc.],
    "bytesize" : [Number of data bits. Possible values: 5, 6, 7, 8],
    "stopbits" : [Number of stop bits. Possible values: 1, 1.5, 2],
    "parity" : [Enable parity checking. Possible values: 'N', ↩
    ↪ 'E', 'O', 'M', 'S'],
    "rtscts" : [Enable hardware (RTS/CTS) flow control.],
    "xonxoff" : [Enable software flow control.],
    "logfile": [Log file path. Possible values: 'nonlog', ↩
    ↪ 'console', [user define path] ]
 }
```

### 2.2.2 disconnect

**Use for disconnect a connection by name.**

**Syntax**:

**disconnect** `conn_name`

**Arguments**:

**conn_name**: Name of the connection.

### 2.2.3 send command

**Use for sending a command to the other side of connection.**

**Syntax**:

**send command** `[conn_name] [command]` *(All parameters are required to be in order)* or
**send command** `conn_name=[conn_name] command=[command]` *(All parameters are assigned by name)*

**Arguments**:

**conn_name**: Name of the connection.
**command**: Command to be sent.

### 2.2.4 verify

**Use for verifying a response from the connection if it matched a pattern.**

**Syntax**:

**verify** `[conn_name] [search_pattern] [timeout] [fetch_block] [eob_pattern] [filter_pattern] [send_cmd]` *(All parameters are required to be in order)* or
**verify** `conn_name=[conn_name] search_pattern=[search_pattern] timeout=[timeout] fetch_block=[fetch_block] eob_pattern=[eob_pattern] filter_pattern=[filter_pattern] send_cmd=[send_cmd]` *(All parameters are assigned by name)*

**Arguments**:

**conn_name**: Name of the connection.
**search_pattern**: Regular expression for matching with the response.
**timeout**: Timeout for waiting response matching pattern.
**fetch_block**: If this value is true, every response line will be put into a block untill a line match **eob_pattern** pattern.
**eob_pattern**: Regular expression for matching the endline when using **fetch_block**.
**filter_pattern**: Regular expression for filtering every line of block when using **fetch_block**.
**send_cmd**: Command to be sent to the other side of connection and waiting for response.

**Return value**:

**A corresponding match object if it is found.**
**E.g.**

```
${result} = verify  conn_name=SSH_Connection
                    search_pattern=(?<=\s).*([0-9]..).*(command).$
                    send_cmd=*echo This is the 1st test command.*
```

- ${result}[0] will be **"This is the 1st test command."** which is the matched string.
- ${result}[1] will be **"1st"** which is the first captured string.
- ${result}[2] will be **"command"** which is the second captured string.

## 2.3 Example

```
*** Settings ***
Documentation    Suite description
Library     QConnectBase.ConnectionManager

*** Test Cases ***
Test SSH Connection
    # Create config for connection.
    ${config_string}=    catenate
    ...  {
    ...   "address": "127.0.0.1",
    ...   "port": 8022,
    ...   "username": "root",
    ...   "password": "",
    ...   "authentication": "password",
    ...   "key_filename": null
    ...  }
    log to console       \nConnecting with configurations:\n${config_string}
    ${config}=             evaluate        json.loads('''${config_string}''')    json

    # Connect to the target with above configurations.
    connect             conn_name=test_ssh
    ...                 conn_type=SSHClient
    ...                 conn_conf=${config}

    # Send command 'cd ..' and 'ls' then wait for the response '.*' pattern.
    send command                conn_name=test_ssh
```

```
...                         command=cd ..

${res}=     verify                  conn_name=test_ssh
...                                 search_pattern=.*
...                                 send_cmd=ls
log to console      ${res}

# Disconnect
disconnect  test_ssh
```

Listing 2.1: Robot code example

## 2.4 Contribution Guidelines

QConnectBaseLibrary is designed for ease of making an extension library. By that way you can take advantage of the QConnectBaseLibrary's infrastructure for handling your own connection protocal. For creating an extension library for QConnectBaseLibrary, please following below steps.

1. Create a library package which have the prefix name is **robotframework-qconnect-***[your specific name]*.
2. Your hadling connection class should be derived from **QConnectionLibrary.connection_base.ConnectionBase** class.
3. In your *Connection Class*, override below attributes and methods:

   - **_CONNECTION_TYPE**: name of your connection type. It will be the input of the conn_type argument when using **connect** keyword. Depend on the type name, the library will detemine the correct connection handling class.
   - **__init__(self, _mode, config)**: in this constructor method, you should:
     - Prepare resource for your connection.
     - Initialize receiver thread by calling **self._init_thread_receiver(cls._socket_instance, mode="")** method.
     - Configure and initialize the lowlevel receiver thread (if it's necessary) as below
       ```
       self._llrecv_thrd_obj = None
        self._llrecv_thrd_term = threading.Event()
        self._init_thrd_llrecv(cls._socket_instance)
       ```
     - Incase you use the lowlevel receiver thread. You should implement the **thrd_llrecv_from_connection_interface** method. This method is a mediate layer which will receive the data from connection at the very beginning, do some process then put them in a queue for the **receiver thread** above getting later.
     - Create the queue for this connection (use Queue.Queue).
   - **connect()**: implement the way you use to make your own connection protocol.
   - **_read()**: implement the way to receive data from connection.
   - **_write()**: implement the way to send data via connection.
   - **disconnect()**: implement the way you use to disconnect your own connection protocol.
   - **quit()**: implement the way you use to quit connection and clean resource.

## 2.5 Configure Git and correct EOL handling

Here you can find the references for Dealing with line endings.

Every time you press return on your keyboard you're actually inserting an invisible character called a line ending. Historically, different operating systems have handled line endings differently. When you view changes in a file, Git handles line endings in its own way. Since you're collaborating on projects with Git and GitHub, Git might produce unexpected results if, for example, you're working on a Windows machine, and your collaborator has made a change in OS X.

To avoid problems in your diffs, you can configure Git to properly handle line endings. If you are storing the .gitattributes file directly inside of your repository, than you can asure that all EOL are manged by git correctly as defined.

## 2.6 Sourcecode Documentation

For investigating sourcecode, please refer to QConnectBase library documentation

## 2.7 Feedback

If you have any problem when using the library or think there is a better solution for any part of the library, I'd love to know it, as this will all help me to improve the library. Please don't hesitate to contact me.

Do share your valuable opinion, I appreciate your honest feedback!

## 2.8 About

### 2.8.1 Maintainers

Nguyen Huynh Tri Cuong

### 2.8.2 Contributors

Nguyen Huynh Tri Cuong

Thomas Pollerspöck

## 2.9 License

# Chapter 3

# __init__.py

## 3.1 Class: ConnectionManager

```
QConnectBase.__init__
```

Class to manage all connections.

# Chapter 4

# connection_base.py

## 4.1 Class: BrokenConnError

```
QConnectBase.connection_base
```

## 4.2 Class: ConnectionBase

```
QConnectBase.connection_base
```

Base class for all connection classes.

### 4.2.1 Method: is_supported_platform

Check if current platform is supported.

**Returns:**

- `is_supported`
  / *Type*: bool /
  True if platform is supported.
  False if platform is not supported.

### 4.2.2 Method: is_precondition_pass

Check for precondition.

**Returns:**

True if passing the precondition.
False if failing the precondition.

### 4.2.3 Method: error_instruction

Get the error instruction.

**Returns:**

Error instruction string.

### 4.2.4 Method: quit

>> This method MUST be overridden in derived class <<
Abstract method for quiting the connection.
**Arguments:**

- `is_disconnect_all`
  / *Condition*: optional / *Type*: bool /
  Determine if it's necessary to disconnect all connections.

**Returns:**

None.

### 4.2.5 Method: connect

>> This method MUST be overridden in derived class <<
Abstract method for quiting the connection.
**Arguments:**

- `device`
  / *Condition*: required / *Type*: str /
  Device name.
- `device`
  / *Condition*: optional / *Type*: list /
  Trace file list if using dlt connection.
- `test_connection`
  / *Condition*: optional / *Type*: bool /
  Determnine if it's necessary for testing the connection.

**Returns:**

None.

### 4.2.6 Method: disconnect

>> This method MUST be overridden in derived class <<
Abstract method for disconnecting connection.
**Arguments:**

- `n_thrd_id`
  / *Condition*: required / *Type*: int /
  Thread id.

**Returns:**

None.

### 4.2.7 Method: send_obj

Wrapper method to send message to a tcp connection.

**Arguments:**

- `obj`
  / *Condition*: required / *Type*: str /
  Data to be sent.
- `cr`
  / *Condition*: optional / *Type*: str /
  Determine if it's necessary to add newline character at the end of command.

**Returns:**

None

### 4.2.8 Method: read_obj

Wrapper method to get the response from connection.

**Returns:**

Responded message.

### 4.2.9 Method: wait_4_trace

Suspend the control flow until a Trace message is received which matches to a specified regular expression.

**Arguments:**

- `search_obj`
  / *Condition*: optional / *Type*: str /
  Regular expression all received trace messages are compare to. Can be passed either as a string or a regular expression object. Refer to Python documentation for module 're'.
- `use_fetch_block`
  / *Condition*: optional / *Type*: bool /
  Determine if 'fetch block' feature is used.
- `end_of_block_pattern`
  / *Condition*: optional / *Type*: str /
  The end of block pattern.
- `filter_pattern`
  / *Condition*: optional / *Type*: str /
  Pattern to filter message line by line.
- `timeout`
  / *Condition*: optional / *Type*: re.Pattern /
  Timeout parameter specified as a floating point number in the unit 'seconds'.
- `fct_args`
  / *Condition*: optional / *Type*: Tuple /
  List of function arguments passed to be sent.

**Returns:**

- `match`

  / *Type*: re.Match /

  If no trace message matched to the specified regular expression and a timeout occurred.

  If a trace message has matched to the specified regular expression, a match object is returned as the result.The complete trace message can be accessed by the 'string' attribute of the match object. For access to groups within the regular expression, use the group() method. For more information, refer to Python documentation for module 're'.

### 4.2.10 Method: wait_4_trace_continuously

Getting trace log continuously without creating a new trace queue.

**Arguments:**

- `trace_queue`

  / *Condition*: optional / *Type*: Queue /

  Queue to store the traces.

- `timeout`

  / *Condition*: optional / *Type*: int /

  Timeout for waiting a matched log.

- `fct_args`

  / *Condition*: optional / *Type*: Tuple /

  Arguments to be sent to connection.

**Returns:**

- `None`

  / *Type*: None /

  If no trace message matched to the specified regular expression and a timeout occurred.

- `match`

  / *Type*: re.Match /

  If no trace message matched to the specified regular expression and a timeout occurred.

  If a trace message has matched to the specified regular expression, a match object is returned as the result.The complete trace message can be accessed by the 'string' attribute of the match object. For access to groups within the regular expression, use the group() method. For more information, refer to Python documentation for module 're'.

### 4.2.11 Method: create_and_activate_trace_queue

Create Queue and assign it to _trace_queue object and activate the queue with the search element.

**Arguments:**

- `search_element`

  / *Condition*: optional / *Type*: str /

  Regular expression all received trace messages are compare to.

  Can be passed either as a string or a regular expression object. Refer to Python documentation for module 're'.#

- `use_fetch_block`

  / *Condition*: optional / *Type*: bool /

  Determine if 'fetch block' feature is used.

- `end_of_block_pattern`
  / *Condition*: optional / *Type*: str /
  The end of block pattern.
- `regex_line_filter_pattern`
  / *Condition*: optional / *Type*: str /
  Regular expression object to filter message line by line.

**Returns:**

- `trq_handle, trace_queue`
  / *Type*: tuple /
  The handle and search object

### 4.2.12 Method: deactivate_and_delete_trace_queue

Deactivate trace queue and delete.

**Arguments:**

- `trq_handle`
  / *Condition*: optional / *Type*: int /
  Trace queue handle.
- `trace_queue`
  / *Condition*: optional / *Type*: Queue /
  Trace queue object.

**Returns:**

None.

### 4.2.13 Method: activate_trace_queue

Activates a trace message filter specified as a regular expression. All matching trace messages are put in the specified queue object.

**Arguments:**

- `search_obj`
  / *Condition*: optional / *Type*: str /
  Regular expression all received trace messages are compare to. Can be passed either as a string or a regular expression object. Refer to Python documentation for module 're'.
- `trace_queue`
  / *Condition*: optional / *Type*: Queue /
  A queue object all trace message which matches the regular expression are put in. The using application must assure, that the queue is emptied or deleted.
- `use_fetch_block`
  / *Condition*: optional / *Type*: bool /
  Determine if 'fetch block' feature is used.
- `end_of_block_pattern`
  / *Condition*: optional / *Type*: str /
  The end of block pattern.

- `line_filter_pattern`
  / *Condition*: optional / *Type*: str /
  Regular expression object to filter message line by line.

**Returns:**

- `handle_id`
  / *Type*: int /
  Handle to deactivate the message filter.

### 4.2.14 Method: deactivate_trace_queue

Deactivates a trace message filter previously activated by ActivateTraceQ() method.
**Arguments:**

- `handle`
  / *Condition*: optional / *Type*: int /
  Integer object returned by ActivateTraceQ() method.

**Returns:**
* `is_success`

  / *Type*: bool / . False : No trace message filter active with the specified handle (i.e. handle is not in use).
  True : Trace message filter successfully deleted.

### 4.2.15 Method: check_timeout

>> This method will be override in derived class <<
Check if responded message come in cls._RESPOND_TIMEOUT or we will raise a timeout event.
**Arguments:**

- `timeout`
  / *Condition*: optional / *Type*: int /
  Timeout in seconds.

**Returns:** None.

### 4.2.16 Method: pre_msg_check

>> This method will be override in derived class <<
Pre-checking message when receiving it from connection.
**Arguments:**

- `msg`
  / *Condition*: optional / *Type*: str /
  Received message to be checked.

**Returns:** None.

### 4.2.17 Method: post_msg_check

>> This method will be override in derived class <<

Post-checking message when receiving it from connection.

**Arguments:**

- `msg`
  / *Condition*: optional / *Type*: str /
  Received message to be checked.

**Returns:** None.

# Chapter 5

# connection_manager.py

## 5.1 Class: InputParam

```
QConnectBase.connection_manager
```

### 5.1.1 Method: get_attr_list

## 5.2 Class: ConnectParam

```
QConnectBase.connection_manager
```

Class for storing parameters for connect action.

## 5.3 Class: SendCommandParam

```
QConnectBase.connection_manager
```

Class for storing parameters for send command action.

## 5.4 Class: VerifyParam

```
QConnectBase.connection_manager
```

Class for storing parameters for verify action.

## 5.5 Class: ConnectionManager

```
QConnectBase.connection_manager
```

Class to manage all connections.

### 5.5.1 Method: quit

Quit connection manager.

**Returns:**

(*no returns*)

### 5.5.2 Method: add_connection

Add a connection to managed dictionary.

**Arguments:**

- `name`
  / *Condition*: required / *Type*: str /
  Connection's name.
- `conn`
  / *Condition*: required / *Type*: socket.socket /
  Connection object.

**Returns:**

(*no returns*)

### 5.5.3 Method: remove_connection

Remove a connection by name.

**Arguments:**

- `connection_name`
  / *Condition*: required / *Type*: str /
  Connection's name.

**Returns:**

(*no returns*)

### 5.5.4 Method: get_connection_by_name

Get an exist connection by name.

**Arguments:**

- `connection_name`
  / *Condition*: required / *Type*: str /
  Connection's name.

**Returns:**

- `conn`
  / *Type*: socket.socket /
  Connection object.

### 5.5.5 Method: disconnect

Keyword for disconnecting a connection by name.

**Arguments:**

- `connection_name`
  / *Condition*: required / *Type*: str /
  Connection's name.

**Returns:**

(*no returns*)

### 5.5.6 Method: connect

Keyword for making a connection.

**Arguments:**

- `args`
  / *Condition*: required / *Type*: tuple /
  Non-Keyword Arguments.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  Keyword Arguments.

**Returns:**

(*no returns*)

### 5.5.7 Method: connect_named_args

Making a connection with name arguments.

**Arguments:**

- `kwargs`
  / *Condition*: required / *Type*: dict /
  Keyword Arguments.

**Returns:**

(*no returns*)

### 5.5.8 Method: connect_unnamed_args

Making a connection.

**Arguments:**

- `connection_name`
  / *Condition*: required / *Type*: str /
  Name of connection.
- `connection_type`
  / *Condition*: required / *Type*: str /
  Type of connection.

- `mode`
  / *Condition*: required / *Type*: str /
  Connection mode.
- `config`
  / *Condition*: required / *Type*: json /
  Configuration for connection.

**Returns:**
(*no returns*)

### 5.5.9 Method: send_command

Keyword for sending command to a connection.
**Arguments:**

- `args`
  / *Condition*: require / *Type*: tuple /
  Non-Keyword Arguments.
- `kwargs`
  / *Condition*: require / *Type*: dict /
  Keyword Arguments.

**Returns:**
(*no returns*)

### 5.5.10 Method: send_command_named_args

Send command to a connection with name arguments.
**Arguments:**

- `kwargs`
  / *Condition*: required / *Type*: dict /
  Keyword Arguments.

**Returns:**
(*no returns*)

### 5.5.11 Method: send_command_unnamed_args

Send command to a connection.
**Arguments:**

- `connection_name`
  / *Condition*: required / *Type*: str /
  Name of connection.
- `command`
  / *Condition*: required / *Type*: str /
  Command to be sent.

**Returns:**
(*no returns*)

### 5.5.12 Method: verify

Keyword uses to verify a pattern from connection response after sending a command.

**Arguments:**

- `args`
  / *Condition*: required / *Type*: tuple /
  Non-Keyword Arguments.
- `kwargs`
  / *Condition*: required / *Type*: dict /
  Keyword Arguments.

**Returns:**

- `match_res`
  / *Type*: str /
  Matched string.

### 5.5.13 Method: verify_named_args

Verify a pattern from connection response after sending a command with named arguments.

**Arguments:**

- `kwargs`
  / *Condition*: required / *Type*: dict /
  Keyword Arguments.

**Returns:**

- `match_res`
  / *Type*: str /
  Matched string.

### 5.5.14 Method: verify_unnamed_args

Verify a pattern from connection response after sending a command.

**Arguments:**

- `connection_name`
  / *Condition*: required / *Type*: str /
  Name of connection.
- `search_obj`
  / *Condition*: required / *Type*: str /
  Regular expression all received trace messages are compare to. Can be passed either as a string or a regular expression object. Refer to Python documentation for module 're'.
- `fetch_block`
  / *Condition*: optional / *Type*: bool / *Default*: False /
  Determine if 'fetch block' feature is used.
- `eob_pattern`
  / *Condition*: optional / *Type*: str / *Default*: '.*' /
  The end of block pattern.

- `filter_pattern`
  / *Condition*: optional / *Type*: str / *Default*: '.*' /
  Pattern to filter message line by line.
- `timeout`
  / *Condition*: optional / *Type*: float / *Default*: 0 /
  Timeout parameter specified as a floating point number in the unit 'seconds'.
- `fct_args`
  / *Condition*: optional / *Type*: Tuple / *Default*: None /
  List of function arguments passed to be sent.

**Returns:**

- `match_res`
  / *Type*: str /
  Matched string.

## 5.6 Class: TestOption

```
QConnectBase.connection_manager
```

# Chapter 6

# constants.py

## 6.1 Class: SocketType

```
QConnectBase.constants
```

## 6.2 Class: String

```
QConnectBase.constants
```

# Chapter 7

# qlogger.py

## 7.1 Class: ColorFormatter

```
QConnectBase.qlogger
```

Custom formatter class for setting log color.

### 7.1.1 Method: format

Set the color format for the log.
**Arguments:**

- `record`
  / *Condition*: required / *Type*: str /
  Log record.

**Returns:** Log with color formatter.

## 7.2 Class: QFileHandler

```
QConnectBase.qlogger
```

Handler class for user defined file in config.

### 7.2.1 Method: get_log_path

Get the log file path for this handler.
**Arguments:**

- `config`
  / *Condition*: required / *Type*: DictToClass /
  Connection configurations.

**Returns:**
*Log file path.*

### 7.2.2 Method: get_config_supported

Check if the connection config is supported by this handler.

**Arguments:**

- `config`
  / *Condition*: required / *Type*: DictToClass /
  Connection configurations.

**Returns:**

True if the config is supported.
False if the config is not supported.

## 7.3 Class: QDefaultFileHandler

```
QConnectBase.qlogger
```

Handler class for default log file path.

### 7.3.1 Method: get_log_path

Get the log file path for this handler.

**Arguments:**

- `logger_name`
  / *Condition*: required / *Type*: str /
  Name of the logger.

**Returns:** Log file path.

### 7.3.2 Method: get_config_supported

Check if the connection config is supported by this handler.

**Arguments:**

- `config`
  / *Condition*: required / *Type*: DictToClass /
  Connection configurations.

**Returns:**

True if the config is supported.
False if the config is not supported.

## 7.4 Class: QConsoleHandler

```
QConnectBase.qlogger
```

Handler class for console log.

### 7.4.1 Method: get_config_supported

Check if the connection config is supported by this handler.

**Arguments:**

- `config`
  / *Condition*: required / *Type*: DictToClass /
  Connection configurations.

**Returns:**

True if the config is supported.
False if the config is not supported.

## 7.5 Class: QLogger

```
QConnectBase.qlogger
```

Logger class for QConnect Libraries.

### 7.5.1 Method: get_logger

Get the logger object.

**Arguments:**

- `logger_name`
  / *Condition*: required / *Type*: str /
  Name of the logger.

**Returns:**

- `logger`
  / *Type*: Logger /
  Logger object. .

### 7.5.2 Method: set_handler

Set handler for logger.

**Arguments:**

- `config`
  / *Condition*: required / *Type*: DictToClass /
  Connection configurations.

**Returns:**

None if no handler is set.
Handler object.

# Chapter 8

# serial_base.py

## 8.1 Class: SerialConfig

```
QConnectBase.serialclient.serial_base
```

Class to store the configuration for Serial connection.

## 8.2 Class: SerialSocket

```
QConnectBase.serialclient.serial_base
```

Class for handling serial connection.

### 8.2.1 Method: connect

Connect to serial port.

**Returns:**

(*no returns*)

### 8.2.2 Method: disconnect

Disconnect serial port.

**Arguments:**

- `_device`
  / *Condition*: required / *Type*: str /
  Unused

**Returns:**

(*no returns*)

### 8.2.3 Method: quit

Quit serial connection.

**Returns:**

(*no returns*)

## 8.3 Class: SerialClient

```
QConnectBase.serialclient.serial_base
```

Serial client class.

### 8.3.1 Method: connect

Connect to the Serial port.

**Returns:**

(*no returns*)

# Chapter 9

# raw_tcp.py

## 9.1 Class: RawTCPBase

```
QConnectBase.tcp/raw.raw_tcp
```

Base class for a raw tcp connection. Class: RawTCPServer ==========

```
QConnectBase.tcp/raw.raw_tcp
```

Class for a raw tcp connection server.

## 9.2 Class: RawTCPClient

```
QConnectBase.tcp/raw.raw_tcp
```

Class for a raw tcp connection client.

# Chapter 10

# ssh_client.py

## 10.1 Class: AuthenticationType

```
QConnectBase.tcp/ssh.ssh_client
```

## 10.2 Class: SSHConfig

```
QConnectBase.tcp/ssh.ssh_client
```

Class to store the configuration for SSH connection. Class: SSHClient ==========

```
QConnectBase.tcp/ssh.ssh_client
```

SSH client connection class. Method: connect ----------
Implementation for creating a SSH connection.
**Returns:**
(*no returns*)

### 10.2.1 Method: close

Close SSH connection.
**Returns:**
(*no returns*)

### 10.2.2 Method: quit

Quit and stop receiver thread.
**Returns:**
(*no returns*)

# Chapter 11

# tcp_base.py

## 11.1 Class: TCPConfig

```
QConnectBase.tcp.tcp_base
```

Class to store configurations for TCP connection.

## 11.2 Class: TCPBase

```
QConnectBase.tcp.tcp_base
```

Base class for a tcp connection.

### 11.2.1 Method: close

Close connection.
**Returns:**
(*no returns*)

### 11.2.2 Method: quit

Quit connection.
**Arguments:**

- `is_disconnect_all`
  / *Condition*: required / *Type*: bool /
  Determine if it's necessary for disconnect all connection.

**Returns:**
(*no returns*)

### 11.2.3 Method: connect

>> Should be override in derived class.
Establish the connection.
**Returns:**
(*no returns*)

### 11.2.4 Method: disconnect

>> Should be override in derived class.

Disconnect the connection.

**Returns:**

(*no returns*)

## 11.3 Class: TCPBaseServer

```
QConnectBase.tcp.tcp_base
```

Base class for TCP server.

### 11.3.1 Method: accept_connection

Wrapper method for handling accept action of TCP Server.

**Returns:**

(*no returns*)

### 11.3.2 Method: connect

### 11.3.3 Method: disconnect

## 11.4 Class: TCPBaseClient

```
QConnectBase.tcp.tcp_base
```

Base class for TCP client.

### 11.4.1 Method: connect

### 11.4.2 Method: disconnect

# Chapter 12

# utils.py

## 12.1 Class: Singleton

```
QConnectBase.utils
```

Class to implement Singleton Design Pattern. This class is used to derive the TTFisClientReal as only a single instance of this class is allowed.

Disabled pyLint Messages: R0903: Too few public methods (%s/%s) Used when class has too few public methods, so be sure it's really worth it.

> This base class implements the Singleton Design Pattern required for the TTFisClientReal. Adding further methods does not make sense.

## 12.2 Class: DictToClass

```
QConnectBase.utils
```

Class for converting dictionary to class object. Method: validate ----------

## 12.3 Class: Utils

```
QConnectBase.utils
```

Class to implement utilities for supporting development. Method: get_all_descendant_classes ----------

Get all descendant classes of a class

**Arguments:** cls: Input class for finding descendants.

**Returns:** Array of descendant classes.

### 12.3.1 Method: get_all_sub_classes

Get all children classes of a class

**Arguments:**

- `cls`
  / *Condition*: required / *Type*: class /
  Input class for finding children.

**Returns:** Array of children classes.

### 12.3.2 Method: is_valid_host

### 12.3.3 Method: execute_command

### 12.3.4 Method: kill_process

### 12.3.5 Method: caller_name

Get a name of a caller in the format module.class.method

**Arguments:**

- `skip`
  / *Condition*: required / *Type*: int /

  **Specifies how many levels of stack to skip while getting caller** name. skip=1 means "who calls me", skip=2 "who calls my caller" etc.

**Returns:** An empty string is returned if skipped levels exceed stack height

### 12.3.6 Method: load_library

Load native library depend on the calling convention.

**Arguments:** path: library path. is_stdcall: determine if the library's calling convention is stdcall or cdecl.

**Returns:** Loaded library object.

### 12.3.7 Method: is_ascii_or_unicode

Check if the string is ascii or unicode

**Arguments:** str_check: string for checking codecs: encoding type list

**Returns:** True : if checked string is ascii or unicode False : if checked string is not ascii or unicode

## 12.4 Class: Job

```
QConnectBase.utils
```

### 12.4.1 Method: stop

### 12.4.2 Method: run

## 12.5 Class: ResultType

```
QConnectBase.utils
```

Result Types. Class: ResponseMessage ========================

```
QConnectBase.utils
```

Response message class

### 12.5.1 Method: get_json

Convert response message to json

**Returns:** Response message in json format

### 12.5.2 Method: get_data

Get string data result

**Returns:** String result

### 12.5.3 Method: create_from_string

# Chapter 13

# Appendix

**About this package:**

Table 13.1: Package setup

| Setup parameter | Value |
|---|---|
| Name | QConnectBase |
| Version | 1.1.0 |
| Date | 05.07.2022 |
| Description | Robot Framework test library for TCP, SSH, serial connection |
| Package URL | robotframework-qconnect-base |
| Author | Nguyen Huynh Tri Cuong |
| Email | cuong.nguyenhuynhtri@vn.bosch.com |
| Language | Programming Language :: Python :: 3 |
| License | License :: OSI Approved :: Apache Software License |
| OS | Operating System :: OS Independent |
| Python required | >=3.0 |
| Development status | Development Status :: 4 - Beta |
| Intended audience | Intended Audience :: Developers |
| Topic | Topic :: Software Development |

# Chapter 14

# History

| | |
|---|---|
| **1.1.0** | 07/2022 |
| *Initial version* | |

**QConnectBase.pdf**
*Created at 29.08.2022 - 15:10:25*
*by GenPackageDoc v. 0.28.0*