

**QConnectBase**

**v. 1.1.4**

Nguyen Huynh Tri Cuong

27.05.2025

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Description</b>	<b>2</b>
2.1	Getting Started . . . . .	2
2.2	Usage . . . . .	2
2.2.1	<code>connect</code> . . . . .	2
2.2.2	<code>disconnect</code> . . . . .	3
2.2.3	<code>send command</code> . . . . .	3
2.2.4	<code>transfer file</code> . . . . .	3
2.2.5	<code>verify</code> . . . . .	4
2.3	Example . . . . .	4
2.4	Auxiliary Utilities . . . . .	5
2.4.1	Introduction to RMQSignal . . . . .	5
2.4.2	Prerequisites . . . . .	5
2.4.3	Library Keywords and Usage . . . . .	6
2.4.4	Example of Inter-Process Communication . . . . .	7
2.5	Contribution Guidelines . . . . .	7
2.6	Configure Git and correct EOL handling . . . . .	8
2.7	Sourcecode Documentation . . . . .	8
2.8	Feedback . . . . .	8
2.9	About . . . . .	9
2.9.1	Maintainers . . . . .	9
2.9.2	Contributors . . . . .	9
2.10	License . . . . .	9
<b>3</b>	<b><code>__init__.py</code></b>	<b>10</b>
3.1	Class: <code>ConnectionManager</code> . . . . .	10
<b>4</b>	<b><code>connection_base.py</code></b>	<b>11</b>
4.1	Class: <code>BrokenConnError</code> . . . . .	11
4.2	Class: <code>ConnectionBase</code> . . . . .	11
4.2.1	Method: <code>is_supported_platform</code> . . . . .	11
4.2.2	Method: <code>is_precondition_pass</code> . . . . .	11
4.2.3	Method: <code>get_connection_type</code> . . . . .	11
4.2.4	Method: <code>error_instruction</code> . . . . .	12
4.2.5	Method: <code>quit</code> . . . . .	12
4.2.6	Method: <code>connect</code> . . . . .	12
4.2.7	Method: <code>disconnect</code> . . . . .	12

4.2.8	Method: <code>send_obj</code>	13
4.2.9	Method: <code>read_obj</code>	13
4.2.10	Method: <code>wait_4_trace</code>	13
4.2.11	Method: <code>wait_4_trace_continuously</code>	14
4.2.12	Method: <code>create_and_activate_trace_queue</code>	14
4.2.13	Method: <code>deactivate_and_delete_trace_queue</code>	15
4.2.14	Method: <code>activate_trace_queue</code>	15
4.2.15	Method: <code>deactivate_trace_queue</code>	16
4.2.16	Method: <code>check_timeout</code>	16
4.2.17	Method: <code>pre_msg_check</code>	16
4.2.18	Method: <code>post_msg_check</code>	16
<b>5</b>	<b><code>connection_manager.py</code></b>	<b>17</b>
5.1	Class: <code>InputParam</code>	17
5.1.1	Method: <code>get_attr_list</code>	17
5.2	Class: <code>ConnectParam</code>	17
5.3	Class: <code>SendCommandParam</code>	17
5.4	Class: <code>VerifyParam</code>	17
5.5	Class: <code>ConnectionManager</code>	17
5.5.1	Method: <code>quit</code>	18
5.5.2	Method: <code>add_connection</code>	18
5.5.3	Method: <code>remove_connection</code>	18
5.5.4	Method: <code>get_connection_by_name</code>	18
5.5.5	Keyword: <code>disconnect</code>	19
5.5.6	Keyword: <code>connect</code>	19
5.5.7	Keyword: <code>send_command</code>	19
5.5.8	Keyword: <code>transfer_file</code>	20
5.5.9	Keyword: <code>execute_script</code>	20
5.5.10	Keyword: <code>set_default_verify_timeout</code>	20
5.5.11	Keyword: <code>set_default_emergency_timeout</code>	20
5.5.12	Keyword: <code>verify</code>	20
5.6	Class: <code>TestOption</code>	21
<b>6</b>	<b><code>constants.py</code></b>	<b>22</b>
6.1	Class: <code>SocketType</code>	22
6.2	Class: <code>String</code>	22
<b>7</b>	<b><code>rabbitmq_client.py</code></b>	<b>23</b>
7.1	Class: <code>RabbitmqClientConfig</code>	23
7.2	Class: <code>RabbitmqClient</code>	23
7.2.1	Method: <code>on_response</code>	23
7.2.2	Method: <code>connect</code>	23
7.2.3	Method: <code>close</code>	23
7.2.4	Method: <code>quit</code>	23
7.3	Class: <code>RMQSignal</code>	23
7.3.1	Method: <code>send_signal</code>	24
7.3.2	Method: <code>unset_signal_receiver_name</code>	24

7.3.3	Method: <code>set_signal_receiver_name</code> . . . . .	24
7.3.4	Method: <code>consume_channel</code> . . . . .	24
7.3.5	Method: <code>wait_for_signal</code> . . . . .	25
7.3.6	Method: <code>wait_for_signals</code> . . . . .	25
<b>8</b>	<b><code>qlogger.py</code></b>	<b>27</b>
8.1	Class: <code>ColorFormatter</code> . . . . .	27
8.1.1	Method: <code>format</code> . . . . .	27
8.2	Class: <code>QFileHandler</code> . . . . .	27
8.2.1	Method: <code>get_log_path</code> . . . . .	27
8.2.2	Method: <code>get_config_supported</code> . . . . .	28
8.3	Class: <code>QDefaultFileHandler</code> . . . . .	28
8.3.1	Method: <code>get_log_path</code> . . . . .	28
8.3.2	Method: <code>get_config_supported</code> . . . . .	28
8.4	Class: <code>QConsoleHandler</code> . . . . .	29
8.4.1	Method: <code>get_config_supported</code> . . . . .	29
8.5	Class: <code>QLogger</code> . . . . .	29
8.5.1	Method: <code>get_logger</code> . . . . .	29
8.5.2	Method: <code>set_handler</code> . . . . .	29
<b>9</b>	<b><code>serial_base.py</code></b>	<b>31</b>
9.1	Class: <code>SerialConfig</code> . . . . .	31
9.2	Class: <code>SerialSocket</code> . . . . .	31
9.2.1	Method: <code>connect</code> . . . . .	31
9.2.2	Method: <code>disconnect</code> . . . . .	31
9.2.3	Method: <code>quit</code> . . . . .	31
9.3	Class: <code>SerialClient</code> . . . . .	32
9.3.1	Method: <code>connect</code> . . . . .	32
<b>10</b>	<b><code>raw_tcp.py</code></b>	<b>33</b>
10.1	Class: <code>RawTCPBase</code> . . . . .	33
10.2	Class: <code>RawTCPServer</code> . . . . .	33
10.3	Class: <code>RawTCPClient</code> . . . . .	33
<b>11</b>	<b><code>ssh_client.py</code></b>	<b>34</b>
11.1	Class: <code>AuthenticationType</code> . . . . .	34
11.2	Class: <code>SSHConfig</code> . . . . .	34
11.3	Class: <code>SSHClient</code> . . . . .	34
11.3.1	Method: <code>connect</code> . . . . .	34
11.3.2	Method: <code>transfer_file</code> . . . . .	34
11.3.3	Method: <code>close</code> . . . . .	35
11.3.4	Method: <code>quit</code> . . . . .	35
<b>12</b>	<b><code>tcp_base.py</code></b>	<b>36</b>
12.1	Class: <code>TCPConfig</code> . . . . .	36
12.1.1	Method: <code>validate</code> . . . . .	36
12.2	Class: <code>TCPBase</code> . . . . .	36
12.2.1	Method: <code>close</code> . . . . .	36

12.2.2 Method: quit . . . . .	36
12.2.3 Method: connect . . . . .	36
12.2.4 Method: disconnect . . . . .	37
12.3 Class: TCPBaseServer . . . . .	37
12.3.1 Method: accept_connection . . . . .	37
12.3.2 Method: connect . . . . .	37
12.3.3 Method: disconnect . . . . .	37
12.4 Class: TCPBaseClient . . . . .	37
12.4.1 Method: connect . . . . .	37
12.4.2 Method: disconnect . . . . .	37
<b>13 utils.py</b>	<b>38</b>
13.1 Class: Singleton . . . . .	38
13.2 Class: DictToClass . . . . .	38
13.2.1 Method: validate . . . . .	38
13.3 Class: Utils . . . . .	38
13.3.1 Method: get_all_descendant_classes . . . . .	38
13.3.2 Method: get_all_sub_classes . . . . .	39
13.3.3 Method: is_valid_host . . . . .	39
13.3.4 Method: execute_command . . . . .	39
13.3.5 Method: kill_process . . . . .	39
13.3.6 Method: caller_name . . . . .	39
13.3.7 Method: load_library . . . . .	39
13.3.8 Method: is_ascii_or_unicode . . . . .	40
13.4 Class: Job . . . . .	40
13.4.1 Method: stop . . . . .	40
13.4.2 Method: run . . . . .	40
13.5 Class: ResultType . . . . .	40
13.6 Class: ResponseMessage . . . . .	40
13.6.1 Method: get_json . . . . .	40
13.6.2 Method: get_data . . . . .	40
13.6.3 Method: create_from_string . . . . .	40
<b>14 Appendix</b>	<b>41</b>
<b>15 History</b>	<b>42</b>

# Chapter 1

## Introduction

QConnectBaseLibrary is a connection testing library for [Robot Framework](#). Library will be supported to download from PyPI soon. It provides a mechanism to handle trace log continuously 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 overflowed response trace log from an asynchronous trace systems. It is supporting Python 3.7+ and RobotFramework 3.2+.

## Chapter 2

# Description

### 2.1 Getting Started

We have a plan to publish all the sourcecode as OSS in the near future so that you can download 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_name]    conn_type=[conn_type]    conn_mode=[conn_mode]  
conn_conf=[conn_conf]
```

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=[conn_name]    command=[command]    [argument
name]=[argument value]
```

Arguments:

**conn\_name:** Name of the connection.

**command:** Command to be sent.

## 2.2.4 transfer file

Use for transferring file from local to remote and vice versa. (Only available for SSHClient connection type)

Syntax:

```
transfer file conn_name=[conn_name]    src=[source]    dest=[destination]
type=[transfer type]
```



**Arguments:**

**conn\_name:** Name of the connection.  
**src:** Your source file's path.  
**dest:** The destination path on the remote side.  
**type:** 'put' to send from local to remote, 'get' to bring it back.

**2.2.5 verify**

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

**Syntax:**

```
verify conn_name=[conn_name]    search_pattern=[search_pattern]    timeout=[timeout]
match_try=[match try time]    fetch_block=[is using fetchblock]    eob_pattern=[end
of block pattern]    filter_pattern=[filter pattern]    send_cmd=[send
comand]    [argument name]=[argument value]
```

**Arguments:**

**conn\_name:** Name of the connection.  
**search\_pattern:** Regular expression for matching with the response.  
**timeout:** Timeout for waiting response matching pattern.  
**match\_try:** Number of time for trying to match the 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.
    ${config_string}=    concatenate
    ...    {
    ...    "address": "127.0.0.1",
    ...    "port": 8022,
    ...    "username": "root",
    ...    "password": "",
    ...    "authentication": "password",
    ...    "key_filename": null
    ...    }
    log to console    \nConnecting with configurations:\n${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 ..

${res}=             Verify            conn_name=${CONNECTION_NAME}
...                 send_cmd=echo  ~~START~~;ls -la;echo  ~~END$?~~\r\n
...                 search_pattern=~\~START\~\~\r\n([\s\S]*)\~\~END(\\d+)\~\~
...                 fetch_block=True
...                 eob_pattern=^~\~END(\\d*)\~\~

Log To Console      ${res}[1]

# Disconnect
disconnect test_ssh

```

Listing 2.1: Robot code example

**Explanation:**

In the example above, we will establish an SSH connection to the remote host 127.0.0.1:8002 and then navigate back one directory level to list all files/folders within that directory.

To achieve this, we utilize the following steps:

1. First, we use the `send command` keyword, which is designed to send commands to the server and immediately return without waiting for a response. In this case, we send the command "cd .." to the remote host.
2. Next, we employ the `verify` keyword, specifically designed for verifying the response from the server after sending a command. The command sent is specified by the `send_cmd` argument (here, it is the "ls -la" command). We use a regular expression defined by the `search_pattern` argument to verify the response and extract the list of files/folders after executing the "ls -la" command.

However, by default, if no additional arguments are provided, only the first line of the response after running the command will be captured. To address this limitation, we introduce two additional arguments: `fetch_block` and `eob_pattern`.

When `fetch_block` is set to "True", the `Verify` keyword will capture all lines returned from the server until it encounters a line that matches the pattern defined by `eob_pattern` (End of Block pattern). This extended functionality ensures that we can effectively capture multi-line returns from the server.

To handle variable-length output, the `send_cmd` parameter encapsulates the command between "START" and "END\$?". By setting `fetch_block` to "True" and specifying the `eob_pattern`, we can accurately capture the entire list returned by the server between "START" and "END\$?".

## 2.4 Auxiliary Utilities

### 2.4.1 Introduction to RMQSignal

The *RMQSignal* class serves as an extension to the core functionalities, providing support for Inter-Process Communication (IPC) based on the RabbitMQ infrastructure. This utility module is designed to facilitate the exchange of signals accompanied by payloads between various processes, enhancing the overall communication mechanism.

### 2.4.2 Prerequisites

**For Windows:**

To use this library, installing RabbitMQ is required. For RabbitMQ installation on Windows, Earlang needs to be installed beforehand.

For user convenience, we have created a .bat file to install all necessary components. Users simply need to run the file QConnectBase/tools/setup-rabbitmq.bat to install the entire infrastructure required for these Auxiliary Utilities.

#### For Ubuntu:

1. Update the system. First, ensure your system is up to date by running the command:

```
sudo apt update && sudo apt upgrade -y
```

2. Add the required repositories. Add the official RabbitMQ signing key and repository by running the following commands:

```
sudo apt install curl gnupg -y
curl -fsSL https://packages.rabbitmq.com/gpg | sudo apt-key add -
sudo add-apt-repository 'deb https://dl.bintray.com/rabbitmq/debian focal main'
```

3. Install RabbitMQ Server. Now, proceed with installing RabbitMQ:

```
sudo apt update && sudo apt install rabbitmq-server -y
```

4. Enable and start the RabbitMQ Service. After installation, enable and start the RabbitMQ service:

```
sudo systemctl enable rabbitmq-server
sudo systemctl start rabbitmq-server
```

### 2.4.3 Library Keywords and Usage

The keywords provided by this library offer a range of functionalities designed to streamline various tasks and processes. Below are some of the key functionalities along with their usage:

#### Send Signal Keyword

**Usage:** Used to send a signal to other processes.

**Syntax:** Send Signal [signal\_name] [payloads] [Receiver]

**Arguments:**

- signal\_name (string): Defines the name of the signal.
- payloads: Specifies the payloads of the signal.
- Receiver (optional, default is None): Specifies the receiver. If defined, the signal will be sent only to that specified receiver. If not defined, it sends a broadcast to all receivers.

#### Wait For Signal Keyword

**Usage:** Used to wait for a specific signal within a timeout.

**Syntax:** Wait For Signal [signal\_name] [timeout]

**Arguments:**

- signal\_name (string): Defines the name of the signal to wait for.
- timeout (optional, default is 10): Specifies the timeout in seconds to wait for the specific signal.

**Return:** Returns the payloads of the received signal, if received within the timeout.

#### Wait For Signals Keyword

**Usage:** Used to wait for multiple specific signals within a timeout.

**Syntax:** Wait For Signals [signal\_names] [timeout]

**Arguments:**

- signal\_names (list of strings): Defines the list of signal names to wait for.
- timeout (optional, default is 10): Specifies the timeout in seconds to wait for the full set of specific signals.

**Return:** Returns a list of payloads of the signals in the same order if the full set of signals is received within the timeout.

### Set Signal Receiver Name Keyword

**Usage:** Used to set a specific name for the Signal Receiver, which defines a unique name for the current process. Other processes can send signals directly to the current process using this receiver name.

**Syntax:** Set Signal Receiver Name [receiver] [force]

**Arguments:**

- receiver (string): Specifies the name to set for the current process signal receiver.
- force (optional, default is True): Determines whether to force-create the signal receiver. If force is set to False, an exception will be raised if any process has already defined this name as a signal receiver.

### Unset Signal Receiver Name Keyword

**Usage:** Used to unset the current Signal Receiver name of the current process.

**Syntax:** Unset Signal Receiver Name

## 2.4.4 Example of Inter-Process Communication

**Content of ProcessA.robot file:**

```
*** Settings ***
Library          QConnectBase.message.RMQSignal

*** Test Cases ***
Test signal
    Log To Console      Process A start
    Send Signal          HelloFromProcA      Hello, I'm A          receiverB
    Sleep                1s
    Send Signal          Hello2FromProcA      Hello 2nd time, I'm A          receiverB
    ${res}=              Wait For Signal      HelloFromProcB      ${20}
    Log To Console       Get resp signal from ProcB: ${res}
```

**Content of ProcessB.robot file:**

```
*** Settings ***
Library          QConnectBase.message.RMQSignal

*** Test Cases ***
Receiving Multiple Signals
    Log To Console      message
    Set Signal Receiver Name      receiverB
    ${list_of_signal} =      Create List      HelloFromProcA      Hello2FromProcAfull
    ${res}=                  Wait For Signals      ${list_of_signal}      ${15}
    Sleep                    2s
    Log To Console          Get signal from ProcA: ${res}
    Send Signal              HelloFromProcB      Hello A, I'm B. Got your signal:${res}
```

**Explanation:** In the above test cases, two separate Robot Framework test files, ProcessA.robot, and ProcessB.robot are used for inter-process communication.

In ProcessB.robot, the test case 'Receiving Multiple Signals' starts by setting up a signal receiver 'receiverB'. It then waits for a set of signals (HelloFromProcA and Hello2FromProcA) from ProcessA. Once both signals are received within 15 seconds, it logs to console the received signals' payloads and waits for 2 seconds before sending a response signal 'HelloFromProcB' to ProcessA.

In ProcessA.robot, the test case 'Sending Multiple Signals' sends HelloFromProcA signal to 'receiverB' in ProcessB. After a 1-second pause, it sends another signal 'Hello2FromProcA' to 'receiverB'. Then, it waits for a response, expecting signal 'HelloFromProcB' from ProcessB within 20 seconds. Upon receiving signal, it logs the received payloads to console.

## 2.5 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 protocol. 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 handling connection class should be derived from **QConnectBase.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 determine 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)
```
    - In case you use the lowlevel receiver thread for receiving data byte by byte. 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.

## 2.6 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, then you can assure that all EOL are managed by git correctly as defined.

## 2.7 Sourcecode Documentation

For investigating sourcecode, please refer to [QConnectBase library documentation](#)

## 2.8 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!

## 2.9 About

### 2.9.1 Maintainers

[Nguyen Huynh Tri Cuong](#)

### 2.9.2 Contributors

[Nguyen Huynh Tri Cuong](#)

[Thomas Pollerspöck](#)

## 2.10 License

Copyright 2020-2023 Robert Bosch GmbH

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

## Chapter 3

### `__init__.py`

#### 3.1 Class: `ConnectionManager`

*Imported by:*

```
from QConnectBase.__init__ import ConnectionManager
```

Class to manage all connections.

# Chapter 4

## connection\_base.py

### 4.1 Class: BrokenConnError

*Imported by:*

```
from QConnectBase.connection_base import BrokenConnError
```

### 4.2 Class: ConnectionBase

*Imported by:*

```
from QConnectBase.connection_base import ConnectionBase
```

Base class for all connection classes.

#### 4.2.1 Method: is\_supported\_platform

Check if current platform is supported.

**Returns:**

*/ Type: bool /*

True if platform is supported.

False if platform is not supported.

#### 4.2.2 Method: is\_precondition\_pass

Check for precondition.

**Returns:**

*/ Type: bool /*

True if passing the precondition.

False if failing the precondition.

#### 4.2.3 Method: get\_connection\_type

Get the connection type.

**Returns:**

*/ Type: str /*

The connection type.



#### 4.2.4 Method: error\_instruction

Get the error instruction.

**Returns:**

*/ Type: str /*  
Error instruction string.

#### 4.2.5 Method: quit

>> This method MUST be overridden in derived class <<

Abstract method for quitting the connection.

**Arguments:**

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

**Returns:**

*(no returns)*

#### 4.2.6 Method: connect

>> This method MUST be overridden in derived class <<

Abstract method for quitting the connection.

**Arguments:**

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

**Returns:**

*(no returns)*

#### 4.2.7 Method: disconnect

>> This method MUST be overridden in derived class <<

Abstract method for disconnecting connection.

**Arguments:**

- `device`  
*/ Condition: required / Type: str /*  
Device's name.

**Returns:**

*(no returns)*

### 4.2.8 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:**

(no returns)

### 4.2.9 Method: read\_obj

Wrapper method to get the response from connection.

**Returns:**

- `msg`  
/ *Type*: str /  
Responded message.

### 4.2.10 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*: 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'.
- `use_fetch_block`  
/ *Condition*: optional / *Type*: bool / *Default*: False /  
Determine if 'fetch block' feature is used.
- `end_of_block_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*: int / *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`

/ *Type*: `re.Match` /

If no trace message matched to the specified regular expression and a timeout occurred, return `None`.

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: `wait_4_trace_continuously`

Getting trace log continuously without creating a new trace queue.

##### Arguments:

- `trace_queue`

/ *Condition*: required / *Type*: `Queue` /

Queue to store the traces.

- `timeout`

/ *Condition*: optional / *Type*: `int` / *Default*: `0` /

Timeout for waiting a matched log.

- `fct_args`

/ *Condition*: optional / *Type*: `Tuple` / *Default*: `None` /

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 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'`.

#### 4.2.12 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*: 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'`.#

- `use_fetch_block`

/ *Condition*: optional / *Type*: `bool` / *Default*: `False` /

Determine if `'fetch block'` feature is used.

- `end_of_block_pattern`

/ *Condition*: optional / *Type*: `str` / *Default*: `'.*'` /

The end of block pattern.

- `regex.line_filter_pattern`  
/ *Condition*: optional / *Type*: `re.Pattern` / *Default*: `None` /  
Regular expression object to filter message line by line.

**Returns:**

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

**4.2.13 Method: deactivate\_and\_delete\_trace\_queue**

Deactivate trace queue and delete.

**Arguments:**

- `trq.handle`  
/ *Condition*: required / *Type*: `int` /  
Trace queue handle.
- `trace_queue`  
/ *Condition*: required / *Type*: `Queue` /  
Trace queue object.

**Returns:**

(no returns)

**4.2.14 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*: 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'.
- `trace_queue`  
/ *Condition*: required / *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` / *Default*: `False` /  
Determine if 'fetch block' feature is used.
- `end_of_block_pattern`  
/ *Condition*: optional / *Type*: `str` / *Default*: `'.*'` /  
The end of block pattern.
- `line_filter_pattern`  
/ *Condition*: optional / *Type*: `re.Pattern` / *Default*: `None` /  
Regular expression object to filter message line by line.

**Returns:**

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

#### 4.2.15 Method: deactivate\_trace\_queue

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

**Arguments:**

- handle  
/ Condition: required / 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.

#### 4.2.16 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: required / Type: int /  
Timeout in seconds.

**Returns:**

(no returns)

#### 4.2.17 Method: pre\_msg\_check

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

Pre-checking message when receiving it from connection.

**Arguments:**

- msg  
/ Condition: required / Type: str /  
Received message to be checked.

**Returns:**

(no returns)

#### 4.2.18 Method: post\_msg\_check

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

Post-checking message when receiving it from connection.

**Arguments:**

- msg  
/ Condition: required / Type: str /  
Received message to be checked.

**Returns:**

(no returns)

## Chapter 5

# connection\_manager.py

### 5.1 Class: InputParam

*Imported by:*

```
from QConnectBase.connection_manager import InputParam
```

#### 5.1.1 Method: get\_attr\_list

### 5.2 Class: ConnectParam

*Imported by:*

```
from QConnectBase.connection_manager import ConnectParam
```

Class for storing parameters for connect action.

### 5.3 Class: SendCommandParam

*Imported by:*

```
from QConnectBase.connection_manager import SendCommandParam
```

Class for storing parameters for send command action.

### 5.4 Class: VerifyParam

*Imported by:*

```
from QConnectBase.connection_manager import VerifyParam
```

Class for storing parameters for verify action.

### 5.5 Class: ConnectionManager

*Imported by:*

```
from QConnectBase.connection_manager import ConnectionManager
```

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 Keyword: disconnect

Keyword for disconnecting a connection by name.

**Arguments:**

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

**Returns:**

(no returns)

### 5.5.6 Keyword: connect

Making a connection.

**Arguments:**

- `conn_name`  
/ *Condition*: optional / *Type*: str / *Default*: 'default\_conn' /  
Name of connection.
- `conn_type`  
/ *Condition*: optional / *Type*: str / *Default*: 'TCPIPClient' /  
Type of connection.
- `conn_mode`  
/ *Condition*: optional / *Type*: str / *Default*: '' /  
Connection mode.
- `conn_conf`  
/ *Condition*: required / *Type*: dict /  
Configuration for connection.

**Returns:**

(no returns)

### 5.5.7 Keyword: send\_command

Send command to a connection.

**Arguments:**

- `connection_name`  
/ *Condition*: required / *Type*: str /  
Name of connection.
- `command`  
/ *Condition*: required / *Type*: str /  
Command to be sent.
- `kwargs`  
/ *Condition*: optional / *Type*: dict / *Default*: {} /  
Keyword Arguments.

**Returns:**

(no returns)



### 5.5.8 Keyword: transfer\_file

Transfer file from local to remote and vice versa.

**Arguments:**

- `connection_name`  
/ *Condition*: required / *Type*: str /  
Name of connection.
- `src`  
/ *Condition*: required / *Type*: str /  
Source file path.
- `dest`  
/ *Condition*: required / *Type*: str /  
Destination file path.
- `type`  
/ *Condition*: required / *Type*: str /  
Transfer file type.  
  
'get' - Copy a remote file from the SFTP server to the local host.  
'put' - Copy a local file to the SFTP server.

**Returns:**

(no returns)

### 5.5.9 Keyword: execute\_script

Executes a script file by sending commands to a device through the provided connection.

**Arguments:**

- `connection_name`  
/ *Condition*: required / *Type*: str /  
Name of connection.
- `script_path`  
/ *Condition*: required / *Type*: str /  
Script file path.

**Returns:**

(no returns)

### 5.5.10 Keyword: set\_default\_verify\_timeout

### 5.5.11 Keyword: set\_default\_emergency\_timeout

### 5.5.12 Keyword: verify

Verify a pattern from connection response after sending a command.

**Arguments:**

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

- `search_pattern`  
*/ Condition: optional / Type: str / Default: .\* /*  
 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'.
- `timeout`  
*/ Condition: optional / Type: float / Default: 5 /*  
 Timeout parameter specified as a floating point number in the unit 'seconds'.
- `match_try`  
*/ Condition: optional / Type: int / Default: 1 /*  
 Number of time for trying to match the pattern.
- `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.
- `send_cmd`  
*/ Condition: optional / Type: str / Default: "" /*  
 Command to be sent.
- `kwargs`  
*/ Condition: optional / Type: Dict / Default: None /*  
 The optional arguments depend on the connection type used in the 'connect' keyword. Here are the supported options:  

Connection Type	Argument	Explanation
Winapp	<code>element_def</code>	Definition for detecting GUI item
	<i>   / Type: str / Default: "" /</i>	
	<i>  </i>	

**Returns:**

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

## 5.6 Class: TestOption

Imported by:

```
from QConnectBase.connection_manager import TestOption
```

## Chapter 6

# constants.py

### 6.1 Class: SocketType

*Imported by:*

```
from QConnectBase.constants import SocketType
```

### 6.2 Class: String

*Imported by:*

```
from QConnectBase.constants import String
```

## Chapter 7

# rabbitmq\_client.py

### 7.1 Class: RabbitmqClientConfig

*Imported by:*

```
from QConnectBase.message.rabbitmq_client import RabbitmqClientConfig
```

Class to store the configuration for SSH connection.

### 7.2 Class: RabbitmqClient

*Imported by:*

```
from QConnectBase.message.rabbitmq_client import RabbitmqClient
```

Rabbitmq client connection class.

#### 7.2.1 Method: on\_response

#### 7.2.2 Method: connect

Implementation for creating a rabbitmq connection.

**Returns:**

*(no returns)*

#### 7.2.3 Method: close

Close rabbitmq connection.

**Returns:**

*(no returns)*

#### 7.2.4 Method: quit

Quit and stop receiver thread.

**Returns:**

*(no returns)*

### 7.3 Class: RMQSignal

*Imported by:*

```
from QConnectBase.message.rabbitmq-client import RMQSignal
```

RMQSignal class.

### 7.3.1 Method: send\_signal

Send signal to other processes.

#### Arguments:

- `signal_name`  
/ *Condition*: required / *Type*: str /  
Signal to be sent.
- `payload`  
/ *Condition*: required / *Type*: str /  
Payloads for signal.
- `receiver`  
/ *Condition*: optional / *Type*: str / *Default*: None /  
Specific the signal receiver to send signal to.

#### Returns:

(no returns)

### 7.3.2 Method: unset\_signal\_receiver\_name

Unset signal receiver.

#### Returns:

(no returns)

### 7.3.3 Method: set\_signal\_receiver\_name

Set the signal receiver to be received signal.

#### Arguments:

- `receiver`  
/ *Condition*: optional / *Type*: str / *Default*: " /  
Name a signal receiver to receive signal.
- `force`  
/ *Condition*: optional / *Type*: bool / *Default*: True /  
Force create the signal receiver (delete the exist signal receiver with the same name).

#### Returns:

(no returns)

### 7.3.4 Method: consume\_channel

Consume the message from specific queue.

#### Arguments:

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

- `queue_name`  
/ *Condition*: required / *Type*: str /  
Name of the queue.
- `routing_key`  
/ *Condition*: required / *Type*: str /  
Routing key string.
- `stop_event`  
/ *Condition*: required / *Type*: Event /  
Event to notify stopping consuming.
- `signal_name`  
/ *Condition*: required / *Type*: str or list /  
Name of the signal to be wait for.
- `messages`  
/ *Condition*: required / *Type*: list or dict /  
Storage for received messages.
- `queue_delete`  
/ *Condition*: optional / *Type*: bool / *Default*: False /  
Determine if we should delete the queue at the end.

**Returns:***(no returns)***7.3.5 Method: wait\_for\_signal**

Wait for specific signal in timeout.

**Arguments:**

- `signal_name`  
/ *Condition*: optional / *Type*: str /  
Name of the signal to wait for.
- `timeout`  
/ *Condition*: optional / *Type*: int / *Default*: 10 /  
Timeout for waitting the signal. Default is 10 seconds.

**Returns:**

/ *Type*: str /  
Payloads of the waitting signal if received.

**7.3.6 Method: wait\_for\_signals**

Wait for multiple specific signals in timeout.

**Arguments:**

- `signal_name`  
/ *Condition*: optional / *Type*: list /  
List of the signals to wait for.

- `timeout`

*/ Condition: optional / Type: int / Default: 10 /*

Timeout for waiting the signal. Default is 10 seconds.

**Returns:**

*/ Type: str /*

List of payloads of the waiting signals if received.

# Chapter 8

## qlogger.py

### 8.1 Class: ColorFormatter

*Imported by:*

```
from QConnectBase.qlogger import ColorFormatter
```

Custom formatter class for setting log color.

#### 8.1.1 Method: format

Set the color format for the log.

**Arguments:**

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

**Returns:**

/ *Type:* logging.Formatter /  
Log with color formatter.

### 8.2 Class: QFileHandler

*Imported by:*

```
from QConnectBase.qlogger import QFileHandler
```

Handler class for user defined file in config.

#### 8.2.1 Method: get\_log\_path

Get the log file path for this handler.

**Arguments:**

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

**Returns:**



*/ Type: str /*

Log file path.

### 8.2.2 Method: get\_config\_supported

Check if the connection config is supported by this handler.

#### Arguments:

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

#### Returns:

*/ Type: bool /*

True if the config is supported.

False if the config is not supported.

## 8.3 Class: QDefaultFileHandler

*Imported by:*

```
from QConnectBase.qlogger import QDefaultFileHandler
```

Handler class for default log file path.

### 8.3.1 Method: get\_log\_path

Get the log file path for this handler.

#### Arguments:

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

#### Returns:

*/ Type: str /*

Log file path.

### 8.3.2 Method: get\_config\_supported

Check if the connection config is supported by this handler.

#### Arguments:

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

#### Returns:

*/ Type: bool /*

True if the config is supported.

False if the config is not supported.

## 8.4 Class: QConsoleHandler

*Imported by:*

```
from QConnectBase.qlogger import QConsoleHandler
```

Handler class for console log.

### 8.4.1 Method: get\_config\_supported

Check if the connection config is supported by this handler.

**Arguments:**

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

**Returns:**

/ *Type*: bool /  
True if the config is supported.  
False if the config is not supported.

## 8.5 Class: QLogger

*Imported by:*

```
from QConnectBase.qlogger import QLogger
```

Logger class for QConnect Libraries.

### 8.5.1 Method: get\_logger

Get the logger object.

**Arguments:**

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

**Returns:**

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

### 8.5.2 Method: set\_handler

Set handler for logger.

**Arguments:**

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

**Returns:**

- `handler_ins`  
/ *Type*: `logging.handler` /  
None if no handler is set.  
Handler object.

# Chapter 9

## serial\_base.py

### 9.1 Class: SerialConfig

*Imported by:*

```
from QConnectBase.serialclient.serialbase import SerialConfig
```

Class to store the configuration for Serial connection.

### 9.2 Class: SerialSocket

*Imported by:*

```
from QConnectBase.serialclient.serialbase import SerialSocket
```

Class for handling serial connection.

#### 9.2.1 Method: connect

Connect to serial port.

**Returns:**

*(no returns)*

#### 9.2.2 Method: disconnect

Disconnect serial port.

**Arguments:**

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

**Returns:**

*(no returns)*

#### 9.2.3 Method: quit

Quit serial connection.

**Returns:**

*(no returns)*

## 9.3 Class: SerialClient

*Imported by:*

```
from QConnectBase.serialclient.serialbase import SerialClient
```

Serial client class.

### 9.3.1 Method: connect

Connect to the Serial port.

**Returns:**

*(no returns)*

# Chapter 10

## raw\_tcp.py

### 10.1 Class: RawTCPBase

*Imported by:*

```
from QConnectBase.tcp.raw.raw_tcp import RawTCPBase
```

Base class for a raw tcp connection.

### 10.2 Class: RawTCPServer

*Imported by:*

```
from QConnectBase.tcp.raw.raw_tcp import RawTCPServer
```

Class for a raw tcp connection server.

### 10.3 Class: RawTCPClient

*Imported by:*

```
from QConnectBase.tcp.raw.raw_tcp import RawTCPClient
```

Class for a raw tcp connection client.

# Chapter 11

## ssh\_client.py

### 11.1 Class: AuthenticationType

*Imported by:*

```
from QConnectBase.tcp.ssh.ssh_client import AuthenticationType
```

### 11.2 Class: SSHConfig

*Imported by:*

```
from QConnectBase.tcp.ssh.ssh_client import SSHConfig
```

Class to store the configuration for SSH connection.

### 11.3 Class: SSHClient

*Imported by:*

```
from QConnectBase.tcp.ssh.ssh_client import SSHClient
```

SSH client connection class.

#### 11.3.1 Method: connect

Implementation for creating a SSH connection.

**Returns:**

*(no returns)*

#### 11.3.2 Method: transfer\_file

Transfer file from local to remote and vice versa.

**Arguments:**

- `connection_name`  
/ *Condition*: required / *Type*: str /  
Name of connection.
- `src`  
/ *Condition*: required / *Type*: str /  
Source file path.

- `dest`  
/ *Condition*: required / *Type*: str /  
Destination file path.
- `type`  
/ *Condition*: required / *Type*: str /  
Transfer file type.

'get' - Copy a remote file from the SFTP server to the local host

'put' - Copy a local file to the SFTP server

**Returns:**

(no returns)

### 11.3.3 Method: close

Close SSH connection.

**Returns:**

(no returns)

### 11.3.4 Method: quit

Quit and stop receiver thread.

**Returns:**

(no returns)



# Chapter 12

## tcp\_base.py

### 12.1 Class: TCPConfig

*Imported by:*

```
from QConnectBase.tcp.tcp_base import TCPConfig
```

Class to store configurations for TCP connection.

#### 12.1.1 Method: validate

### 12.2 Class: TCPBase

*Imported by:*

```
from QConnectBase.tcp.tcp_base import TCPBase
```

Base class for a tcp connection.

#### 12.2.1 Method: close

Close connection.

**Returns:**

*(no returns)*

#### 12.2.2 Method: quit

Quit connection.

**Arguments:**

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

**Returns:**

*(no returns)*

#### 12.2.3 Method: connect

>> Should be override in derived class.

Establish the connection.

**Returns:**

*(no returns)*

#### 12.2.4 Method: disconnect

>> Should be override in derived class.

Disconnect the connection.

**Returns:**

*(no returns)*

### 12.3 Class: TCPBaseServer

*Imported by:*

```
from QConnectBase.tcp.tcp_base import TCPBaseServer
```

Base class for TCP server.

#### 12.3.1 Method: accept\_connection

Wrapper method for handling accept action of TCP Server.

**Returns:**

*(no returns)*

#### 12.3.2 Method: connect

#### 12.3.3 Method: disconnect

### 12.4 Class: TCPBaseClient

*Imported by:*

```
from QConnectBase.tcp.tcp_base import TCPBaseClient
```

Base class for TCP client.

#### 12.4.1 Method: connect

#### 12.4.2 Method: disconnect

# Chapter 13

## utils.py

### 13.1 Class: Singleton

*Imported by:*

```
from QConnectBase.utils import Singleton
```

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.

### 13.2 Class: DictToClass

*Imported by:*

```
from QConnectBase.utils import DictToClass
```

Class for converting dictionary to class object.

#### 13.2.1 Method: validate

### 13.3 Class: Utils

*Imported by:*

```
from QConnectBase.utils import Utils
```

Class to implement utilities for supporting development.

#### 13.3.1 Method: get\_all\_descendant\_classes

Get all descendant classes of a class

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

**Returns:**

/ *Type:* list /  
Array of descendant classes.

### 13.3.2 Method: get\_all\_sub\_classes

Get all children classes of a class

#### Arguments:

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

#### Returns:

/ *Type*: list /  
Array of children classes.

### 13.3.3 Method: is\_valid\_host

### 13.3.4 Method: execute\_command

### 13.3.5 Method: kill\_process

### 13.3.6 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:

/ *Type*: str /  
An empty string is returned if skipped levels exceed stack height

### 13.3.7 Method: load\_library

Load native library depend on the calling convention.

#### Arguments:

- `path`  
/ *Condition*: required / *Type*: str /  
Library path.
- `is_stdcall`  
/ *Condition*: optional / *Type*: bool / *Default*: True /  
Determine if the library's calling convention is stdcall or cdecl.

#### Returns:

*Loaded library object.*

### 13.3.8 Method: is\_ascii\_or\_unicode

Check if the string is ascii or unicode

**Arguments:** str\_check: string for checking codecs: encoding type list

**Returns:**

*/ Type: bool /*

True : if checked string is ascii or unicode

False : if checked string is not ascii or unicode

## 13.4 Class: Job

*Imported by:*

```
from QConnectBase.utils import Job
```

### 13.4.1 Method: stop

### 13.4.2 Method: run

## 13.5 Class: ResultType

*Imported by:*

```
from QConnectBase.utils import ResultType
```

Result Types.

## 13.6 Class: ResponseMessage

*Imported by:*

```
from QConnectBase.utils import ResponseMessage
```

Response message class

### 13.6.1 Method: get\_json

Convert response message to json

**Returns:**

*Response message in json format*

### 13.6.2 Method: get\_data

Get string data result

**Returns:**

*String result*

### 13.6.3 Method: create\_from\_string

## Chapter 14

# Appendix

### About this package:

Table 14.1: Package setup

Setup parameter	Value
Name	QConnectBase
Version	1.1.4
Date	27.05.2025
Description	Robot Framework test library for TCP, SSH, serial connection
Package URL	<a href="#">robotframework-qconnect-base</a>
Author	Nguyen Huynh Tri Cuong
Email	<a href="mailto:cuong.nguyenhuyhtri@vn.bosch.com">cuong.nguyenhuyhtri@vn.bosch.com</a>
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 15

# History

1.1.0	07/2022
<i>Initial version</i>	