

QuecPython Socket Application Note

LTE Standard Module Series

Version: 1.0.0

Date: 2020-11-12 Status:

Preliminary



Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.

Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China

Tel: +86 21 5108 6236

Email: info@quectel.com

Or our local office. For more information, please visit: <http://www.quectel.com/support/sales.htm>.

For technical support, or to report documentation errors, please visit:

<http://www.quectel.com/support/technical.htm> Or email to support@quectel.com.

General Notes

Quectel offers the information as a service to its customers. The information provided is based upon customers' requirements. Quectel makes every effort to ensure the quality of the information it makes available. Quectel does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information. All information supplied herein is subject to change without prior notice.

Disclaimer

While Quectel has made efforts to ensure that the functions and features under development are free from errors, it is possible that these functions and features could contain errors, inaccuracies and omissions. Unless otherwise provided by valid agreement, Quectel makes no warranties of any kind, implied or express, with respect to the use of features and functions under development. To the maximum extent permitted by law, Quectel excludes all liability for any loss or damage suffered in connection with the use of the functions and features under development, regardless of whether such loss or damage may have been foreseeable.

Duty of Confidentiality

The Receiving Party shall keep confidential all documentation and information provided by Quectel, except when the specific permission has been granted by Quectel. The Receiving Party shall not access or use Quectel's documentation and information for any purpose except as expressly provided herein. Furthermore, the Receiving Party shall not disclose any of the Quectel's documentation and information to any third party without the prior written consent by Quectel. For any noncompliance to the above requirements, unauthorized use, or other illegal or malicious use of the documentation and information, Quectel will reserve the right to take legal action.

Copyright

The information contained here is proprietary technical information of Quectel Wireless Solutions Co., Ltd. Transmitting, reproducing, disseminating and editing this document as well as using the content without permission are forbidden. Offenders will be held liable for payment of damages. All rights are reserved in the event of a patent grant or registration of a utility model or design.

Copyright © Quectel Wireless Solutions Co., Ltd. 2020. All rights reserved.

About the Document

Revision History

Version	Date	Author	Description
-	2020-11-12	Rivern/ Kingka	Creation of the document
1.0.0	2020-11-12	Rivern/ Kingka	Preliminary

Contents

About the Document.....	3
Contents.....	4
1 Introduction	4
2 Socket Overview	5
2.1. Brief Introduction on Socket.....	5
2.2. Socket Application.....	5
2.3. QuecPython Socket API Details	5
2.3.1. usocket.socket.....	6
2.3.2. usock.getaddrinfo	6
2.3.3. sock.bind	7
2.3.4. sock.listen.....	7
2.3.5. sock.accept.....	7
2.3.6. sock.connect.....	8
2.3.7. sock.recv	8
2.3.8. sock.send	9
2.3.9. sock.close.....	9
2.3.10. sock.read	10
2.3.11. sock.readinto	10
2.3.12. sock.readline	11
2.3.13. sock.write.....	11
2.3.14. sock.sendall.....	11
2.3.15. sock.sendto	12
2.3.16. sock.recvfrom	12
2.3.17. sock.setsockopt.....	13
2.3.18. sock.setblocking	13
2.3.19. sock.settimeout	14
2.3.20. Socket.makefile	14
3 Example	15
4 Appendix.....	20

1 Introduction

This document takes Quectel EC100Y-CN module as an example to introduce how to use QuecPython class library API on Quectel EC100Y-CN & EC600S-CN modules to realize basic Socket communication functions.

2 Socket Overview

2.1. Brief Introduction on Socket

Socket is an endpoint for the dual communication between application processes on different hosts in the network. One socket represents an endpoint for process communication, providing a mechanism for application layer processes to transfer data using network protocols. In terms of its roles, Socket associates applications process and network protocol stack respectively and is an interface for applications to communicate through network protocols, and for applications to interact with the network protocol root.

In addition to the connection endpoints for the application communication, Socket is also an API for inter-process communication in the network environment. It is also a communication endpoint that can be named and addressed. Each socket has its own type and a process to connected to. In communication, an application writes a message to be transferred into the Socket of its host, then this socket sends the message to the Socket of another host through the transmission medium connected to the network interface card (NIC) so that the other party can receive the message. Socket is composed of IP address and port, and provides a mechanism for transmitting data packets to application layer processes

2.2. Socket Application

Socket enables applications to read and write data from the network, and two applications on different computers can send and receive byte streams through the connection. But, sending message requires a IP and port from other party. There are many cases of socket applications in our daily life, such as when you browse the web, the browser process communicates with the web server process through Socket. When you chat with QQ, the QQ process communicates with the server or friend QQ process through Socket, and so on.

2.3. QuecPython Socket API Details

Socket originated from Unix, and one of the basic philosophy of Unix/Linux is that "everything is a file", which can be operated in the mode of "open→write/read→close". In the communication through Socket, the server is regarded as a web server, and the client is regarded as a browser that wants to access the web server. The access process is the flow of open→write/read→close.

The QuecPython class library implements the Socket function through usocket which provides access to the BSD socket interface. Usocket module achieves a subset of the corresponding CPython module, see the [socket](#) for details about CPython. The Details of API related to QuecPython Socket are as follows:

2.3.1. `usocket.socket`

This function creates a socket object by the server or client.

★ **Prototype**

```
sock=usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM)
```

★ **Parameter**

usocket.AF_INET:

Network protocol, IPv4.

usocket.SOCK_STREAM:

TCP stream socket. See *Quectel QuecPython class library API Introduction* for more constant definitions.

★ **Return Value**

None.

2.3.2. `usock.getaddrinfo`

This function converts the host domain name and port into a 5-tuple sequence for a socket creation. The tuple structure is as follows:

(family, type, proto, canonname, sockaddr)

★ **Prototype**

```
usocket.getaddrinfo(host, port)
```

★ **Parameter**

host:

Host domain name.

port:

Host port.

★ **Return Value**

None.

2.3.3. sock.bind

This function binds the Socket object and the server IP: port. Since the TCP port is dynamic, there is no need to bind with the client. Before using this function, a unbonded Socket is required.

★ Prototype

```
sock.bind(address)
```

★ Parameter

address:

A list or tuple of port numbers of an address.

★ Return Value

None.

2.3.4. sock.listen

This function allows the server to accept Socket connections and specifies the maximum number of connections.

★ Prototype

```
sock.listen(backlog)
```

★ Parameter

backlog:

The maximum number of socket connected, at least 0.

★ Return Value

None.

2.3.5. sock.accept

This function is used by the server to accept connection requests from the client.

★ Prototype

```
(conn, address) = sock.accept()
```


★ **Parameter**

None.

★ **Return Value**

Return a tuple, including new Socket and client address, in the form: (conn, address).

conn:

New Socket object to send and receive data.

address:

The client address connected to the server.

2.3.6. **sock.connect**

This function is used by the client to connect to the server with the specified address.

★ **Prototype**

```
sock.connect(address)
```

★ **Parameter**

address:

The server address connected to the client.

★ **Return Value**

None.

2.3.7. **sock.recv**

This function receives data sent by the client or server.

★ **Prototype**

```
recv_data = sock.recv(bufsize)
```

★ **Parameter**

bufsize:

The maximum amount of data received at one time

★ **Return Value**

The received data in bytes.

2.3.8. **sock.send**

This function sends data to the server or client.

★ **Prototype**

```
sock.send(send_data.encode("utf8"))
```

★ **Parameter**

send_data:

Data to be sent.

★ **Return Value**

The number of bytes actually sent.

★ **NOTE**

Since the socket of TCP protocol is based on byte stream, before sending data through the socket, please use `encode("utf8")` to encode the data, where "utf8" is the encoding method.

2.3.9. **sock.close**

This function closes the socket communication.

★ **Prototype**

```
sock.close()
```

★ **Parameter**

None.

★ Return Value

None.

2.3.10. sock.read

This function reads data in bytes from the socket. If parameter *[size]* is not specified, all data will be read from the socket until the data is completely read.

★ Prototype

```
socket.read([ size ])
```

★ Parameter

[size]:

Data to be read, in bytes.

★ Return Value

The number of bytes actually read.

2.3.11. sock.readinto

This function reads bytes into the buffer.

★ Prototype

```
sock.readinto(buf, [ , nbytes ])
```

★ Parameter

buf:

A buffer for storing read bytes.

nbytes:

The number of bytes read.

★ Return Value

The number of bytes actually read.

2.3.12. sock.readline

This function reads data line by line, and data reading will end with a newline character.

★ Prototype

```
sock.readline()
```

★ Parameter

None.

★ Return Value

The data row read.

2.3.13. sock.write

This function writes data to the buffer area.

★ Prototype

```
sock.write(buf)
```

★ Parameter

buf:

Data written to the buffer.

★ Return Value

The number of bytes actually written.

2.3.14. sock.sendall

This function sends all data to the socket.

★ Prototype

```
sock.sendall(bytes)
```

★ Parameter

bytes:

Buffer for storing data in bytes.

★ **Return Value**

None.

2.3.15. sock.sendto

This function sends data to the socket. The socket should not be connected to the remote socket, because the target socket is specified by *address*.

★ **Prototype**

```
sock.sendto(bytes, address)
```

★ **Parameter**

bytes:

Buffer for storing data in bytes.

address:

A tuple or list containing addresses and ports.

★ **Return Value**

None.

2.3.16. sock.recvfrom

This function receives data from the socket. A tuple containing the bytes and address will be returned.

★ **Prototype**

```
socket.recvfrom(bufsize)
```

★ **Parameter**

bufsize:

The received buffer data.

★ **Return Value**

A tuple containing the bytes and address, in format of (bytes, address).

bytes:

The received data in bytes.

address:

The socket address for sending data.

2.3.17. **sock.setsockopt**

This function sets the information about socket options.

★ **Prototype**

```
socket.setsockopt(level, optname, value)
```

★ **Parameter**

level:

Socket option level.

optname:

Socket option.

value: an integer or bytes of the buffer.

★ **Return Value**

None.

2.3.18. **sock.setblocking**

This function sets the socket to blocking mode or non-blocking mode. If *flag* is false, set the socket to non-blocking mode, otherwise, set it to blocking mode.

★ **Prototype**

```
socket.setblocking(flag)
```

★ **Parameter**

flag:

True blocking mode

False non-blocking mode

★ Return Value

None.

2.3.19. sock.settimeout

This function sets the expiration time of the socket. Unit: second.

★ Prototype

```
socket.settimeout(value)
```

★ Parameter

value:

A non-negative floating point in seconds, or None. If it is set to a non-zero value, OSError is beyond the expiration value before the operation was completed, so the later socket operation will be abnormal. If it is set to zero, the socket is in non-blocking mode. If this value is not specified, the socket will be in blocking mode.

★ Return Value

None.

2.3.20. Socket.makefile

This function generates a file to associate with the socket, and then you can use the socket like reading a file. (file operations include open and write, etc.)

★ Prototype

```
socket.makefile(mode='rb')
```

★ Parameter

mode:

Binary mode. Values are rb and wb.

★ Return Value

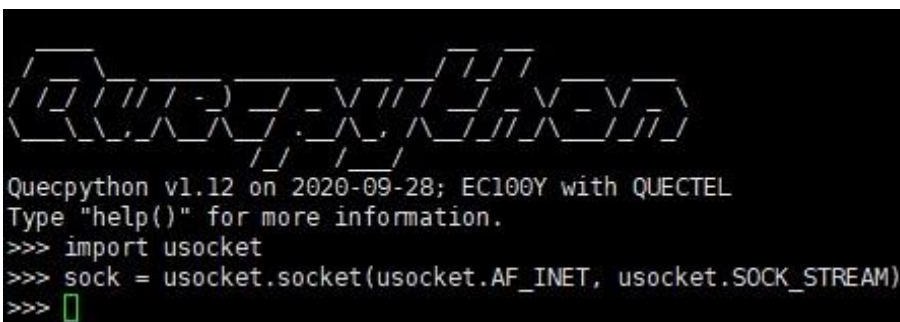
The file associated with the socket.

3 Example

This chapter provides an example to create a socket on QuecPython, that is, taking the browser to access the web server to obtain the web page content as an example. In Xshell, connect to the main serial port of the module, enter the communication interface, and then follow the steps below to implement the Socket function:

Step 1: Import the usocket module and create a Socket instance:

```
import usocket
sock = usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM)
```



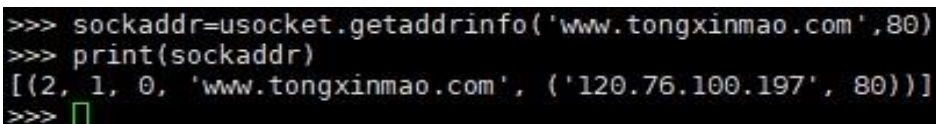
```
Quecpython v1.12 on 2020-09-28; EC100Y with QUECTEL
Type "help()" for more information.
>>> import usocket
>>> sock = usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM)
>>> 
```

Step 2: Resolve the domain name:

```
sockaddr=usocket.getaddrinfo('www.tongxinmao.com',80)[0][-1]
```

Convert the format of host domain name and port into a 5-tuple sequence used to create a Socket. The tuple structure is as follows:

(family, type, proto, canonname, sockaddr)



```
>>> sockaddr=usocket.getaddrinfo('www.tongxinmao.com',80)
>>> print(sockaddr)
[(2, 1, 0, 'www.tongxinmao.com', ('120.76.100.197', 80))]
>>> 
```

Step 3: Establish a connection with the server

```
sock.connect(sockaddr)
```



```
>>> #
>>> sock.connect(sockaddr)
>>>
```

Step 4: Send a message to the server

```
ret=sock.send('GET /News HTTP/1.1\r\nHost: www.tongxinmao.com\r\nAccept-Encoding:
deflate\r\nConnection: keep-alive\r\n\r\n')
print('send %d bytes' % ret)
```

```
>>> ret=sock.send('GET /News HTTP/1.1\r\nHost: www.tongxinmao.com\r\nAccept-Encoding: deflate\r\nConnection: keep-alive\r\n\r\n')
>>> print('send %d bytes' % ret)
send 98 bytes
>>> []
```

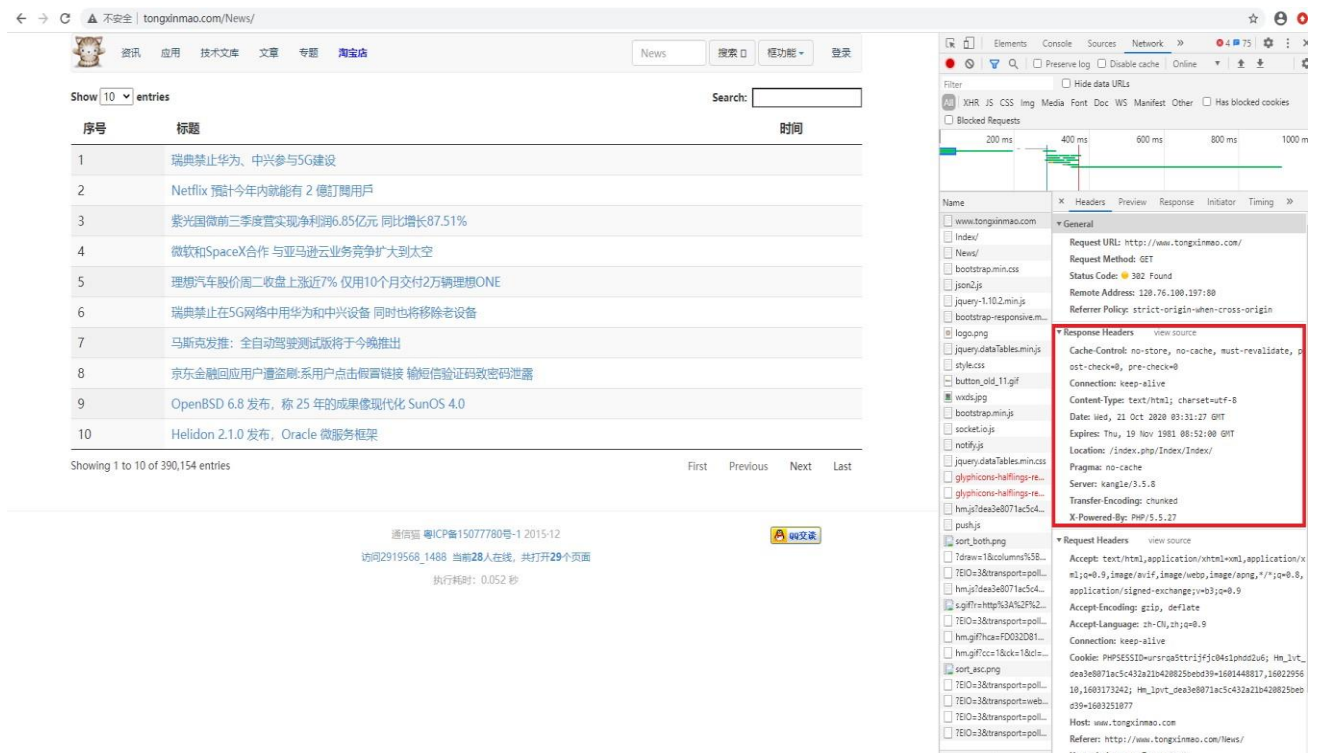
Step 5: Receive the message from the server

```
data=sock.recv(1024)
print('recv %s bytes:' % len(data))
print(data.decode())
```

```
>>> ret=sock.send('GET /News HTTP/1.1\r\nHost: www.tongxinmao.com\r\nAccept-Encoding: deflate\r\nConnection: keep-alive\r\n\r\n')
>>> print('send %d bytes' % ret)
send 98 bytes
>>>
>>> data=sock.recv(1024)
>>> print('recv %s bytes:' % len(data))
recv 1024 bytes:
>>> print(data.decode())
HTTP/1.1 200 OK
Server: kangle/3.5.8
Date: Wed, 21 Oct 2020 03:42:33 GMT
Set-Cookie: PHPSESSID=bu0l6f31lc0ml153k8g8nsq531; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Type: text/html; charset=utf-8
X-Powered-By: SBT
X-Server: SBT
Server: SBT
Transfer-Encoding: chunked
Connection: keep-alive

1ee0
<!DOCTYPE html>
<html lang="zh">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="author" content="tcm123@126.com">
<meta name="keywords" content="通信猫">
<meta name="description" content="通信猫 ">
<title>通信猫--业界资讯</title>
<link rel="stylesheet" href="//tongxinmao.com/assets/css/bootstrap.min.css">
<link href="//tongxinmao.com/assets/css/bootstrap-responsive.min.css" rel="stylesheet">
<link href="//tongxinmao.com/assets/css/style.css" rel="stylesheet">
<script src="//tongx
>>> □
```

After the server receives the message successfully, you can initiate a request on the browser to verify whether the returned message is consistent with the message received by the Socket, as shown below:



Step 6: Close the socket.

```
sock.close()
```

Part of the code above is stored in the path of *moudles/socket/example_socket.py* in SDK provided by Quectel. You can also execute the script by the example module.

The code to achieve usocket function are as follows:

```
#Import an usocket modules. import
usocket

# create a socket instance.
sock = usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM) #Set
multiplexing port.
sock.setsockopt(usocket.SOL_SOCKET, usocket.SO_REUSEADDR, 1)

sock.bind(('127.0.0.1', 6000))

sock.listen(50) while
True:
    newSock, addr = sock.accept()
    newSock.send('hello world')    recv_data =
    newSock.recv(256)
    print(recv_data.decode())

    newSock.close()
    break
```

For details about usocket-API, see gpy.quectel.com/wiki/#/zh-cn/api/.

4 Appendix

Table 1: Terms and Abbreviations

Abbreviation	Description
API	Application Programming Interface
BSD	Berkeley Socket
HTTP	Hypertext Transfer Protocol
IPv4	Internet Protocol version 4
NIC	Network Interface Controller
SDK	Software Development Kit
TCP	Transmission Control Protocol