BC66&BC66-NA HTTP(S) Application Note

NB-IoT Module Series

Version: 1.0

Date: 2022-04-22

Status: Released

At Quectel, our aim is to provide timely and comprehensive services to our customers. If you require any assistance, please contact our 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 offices. 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 us at: support@quectel.com.

Legal Notices

We offer information as a service to you. The provided information is based on your requirements and we make every effort to ensure its quality. You agree that you are responsible for using independent analysis and evaluation in designing intended products, and we provide reference designs for illustrative purposes only. Before using any hardware, software or service guided by this document, please read this notice carefully. Even though we employ commercially reasonable efforts to provide the best possible experience, you hereby acknowledge and agree that this document and related services hereunder are provided to you on an “as available” basis. We may revise or restate this document from time to time at our sole discretion without any prior notice to you.

Use and Disclosure Restrictions

License Agreements

Documents and information provided by us shall be kept confidential, unless specific permission is granted. They shall not be accessed or used for any purpose except as expressly provided herein.

Copyright

Our and third-party products hereunder may contain copyrighted material. Such copyrighted material shall not be copied, reproduced, distributed, merged, published, translated, or modified without prior written consent. We and the third party have exclusive rights over copyrighted material. No license shall be granted or conveyed under any patents, copyrights, trademarks, or service mark rights. To avoid ambiguities, purchasing in any form cannot be deemed as granting a license other than the normal non-exclusive, royalty-free license to use the material. We reserve the right to take legal action for noncompliance with abovementioned requirements, unauthorized use, or other illegal or malicious use of the material.

Trademarks

Except as otherwise set forth herein, nothing in this document shall be construed as conferring any rights to use any trademark, trade name or name, abbreviation, or counterfeit product thereof owned by Quectel or any third party in advertising, publicity, or other aspects.

Third-Party Rights

This document may refer to hardware, software and/or documentation owned by one or more third parties ("third-party materials"). Use of such third-party materials shall be governed by all restrictions and obligations applicable thereto.

We make no warranty or representation, either express or implied, regarding the third-party materials, including but not limited to any implied or statutory, warranties of merchantability or fitness for a particular purpose, quiet enjoyment, system integration, information accuracy, and non-infringement of any third-party intellectual property rights with regard to the licensed technology or use thereof. Nothing herein constitutes a representation or warranty by us to either develop, enhance, modify, distribute, market, sell, offer for sale, or otherwise maintain production of any our products or any other hardware, software, device, tool, information, or product. We moreover disclaim any and all warranties arising from the course of dealing or usage of trade.

Privacy Policy

To implement module functionality, certain device data are uploaded to Quectel's or third-party's servers, including carriers, chipset suppliers or customer-designated servers. Quectel, strictly abiding by the relevant laws and regulations, shall retain, use, disclose or otherwise process relevant data for the purpose of performing the service only or as permitted by applicable laws. Before data interaction with third parties, please be informed of their privacy and data security policy.

Disclaimer

a) We acknowledge no liability for any injury or damage arising from the reliance upon the information.
b) We shall bear no liability resulting from any inaccuracies or omissions, or from the use of the information contained herein.
c) While we have made every effort to ensure that the functions and features under development are free from errors, it is possible that they could contain errors, inaccuracies, and omissions. Unless otherwise provided by valid agreement, we make no warranties of any kind, either implied or express, and exclude all liability for any loss or damage suffered in connection with the use of features and functions under development, to the maximum extent permitted by law, regardless of whether such loss or damage may have been foreseeable.
d) We are not responsible for the accessibility, safety, accuracy, availability, legality, or completeness of information, advertising, commercial offers, products, services, and materials on third-party websites and third-party resources.

About the Document

Revision History

Version	Date	Author	Description
-	2021-05-05	Taber JIANG	Creation of the document
1.0	2022-04-22	Taber JIANG/ Quinten SONG	First official release

Contents

Table Index

1 Introduction

Quectel BC66 and BC66-NA modules provide HTTP(S) applications to HTTP(S) server.

Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed, collaborative, hypermedia information systems.

Hypertext Transfer Protocol Secure (HTTPS) is a variant of the standard web transfer protocol (HTTP) that adds a layer of security on the data in transit through a secure socket layer (SSL) or transport layer security (TLS) protocol connection. The main purpose of HTTPS development is to provide identity authentication for website servers and protect the privacy and integrity of exchanged data.

This document is a reference guide to all the AT commands defined for HTTP(S).

1.1. Description of HTTP(S) Request Header

1.1.1. Customize HTTP(S) Request Header

HTTP(S) request header is filled by the module automatically. HTTP(S) POST request header can be customized by specifying **<request_header>** to 1 via **AT+QHTTPCFG** (see ***Chapter 2.3.1***) and then inputting the header (see ***Chapter 2.3.4***) according to the following requirements:

- Follow HTTP(S) POST request header syntax.
- The URI in HTTP(S) POST request header must be in line with the URL configured by **AT+QHTTPURL**.
- The HTTP(S) POST request header must end with **<CR><LF>**.

The following example shows a valid HTTP(S) POST request:

POST /processorder.php HTTP/1.1<CR><LF>
Host: 220.180.239.212:8011<CR><LF>
Accept: */*<CR><LF>
User-Agent: QUECTEL_MODULE<CR><LF>
Connection: Keep-Alive<CR><LF>
Content-Type: application/x-www-form-urlencoded<CR><LF>
Content-Length: 48<CR><LF>
<CR><LF>
Message=1111&Appleqty=2222&Orangeqty=3333&find=1

1.1.2. Output HTTP(S) Response Header

HTTP(S) response header is not outputted automatically. HTTP(S) response header information can be obtained by specifying **<response_header>** to 1 via **AT+QHTTPCFG** (see ***Chapter 2.3.1***), and then HTTP(S) response header is outputted with HTTP(S) response body after executing **AT+QHTTPREAD** (see ***Chapter 2.3.5***).

2 Description of HTTP(S) AT Commands

2.1. AT Command Syntax

2.1.1. Definitions

- **<CR>** Carriage return character.
- **<LF>** Line feed character.
- **<...>** Parameter name. Angle brackets do not appear on the command line.
- **[...]** Optional parameter of a command or an optional part of TA information response. Square brackets do not appear on the command line. When an optional parameter is not given in a command, the new value equals its previous value or the default settings, unless otherwise specified.
- **Underline** Default setting of a parameter.

2.1.2. AT Command Syntax

All command lines must start with **AT** or **at** and end with **<CR>**. Information responses and result codes always start and end with a carriage return character and a line feed character: **<CR><LF><response><CR><LF>**. In tables presenting commands and responses throughout this document, only the commands and responses are presented, and **<CR>** and **<LF>** are deliberately omitted.

Table 1: Types of AT Commands

Command Type	Syntax	Description
Test Command	**AT+<cmd>=?**	Test the existence of the corresponding command and return information about the type, value, or range of its parameter.
Read Command	**AT+<cmd>?**	Check the current parameter value of the corresponding command.
Write Command	**AT+<cmd>=<p1>[,<p2>[,<p3>[...]]]**	Set user-definable parameter value.
Execution Command	**AT+<cmd>**	Return a specific information parameter or perform a specific action.

2.2. Declaration of AT Command Examples

The AT command examples in this document are provided to help you learn about the use of the AT commands introduced herein. The examples, however, should not be taken as Quectel's recommendations or suggestions about how to design a program flow or what status to set the module into. Sometimes multiple examples may be provided for one AT command. However, this does not mean that there is a correlation among these examples, or that they should be executed in a given sequence.

2.3. AT Command Description

2.3.1. AT+QHTTPCFG Configure Parameters for HTTP(S) Server

This command configures the parameters for HTTP(S) server, including configuring a PDP context ID, whether to customize HTTP(S) request header, whether to output HTTP(S) response header and querying SSL settings. If the Write Command only executes one parameter, it queries the current settings.

AT+QHTTPCFG Configure Parameters for HTTP(S) Server	
Test Command **AT+QHTTPCFG=?**	Response **+QHTTPCFG: "contextid",(**range of supported **<contextID>**s**)** **+QHTTPCFG: "requestheader",(**list of supported **<request_header>**s**)** **+QHTTPCFG: "responseheader",(**list of supported **<response_header>**s**)** **+QHTTPCFG: "contenttype",(**range of supported **<content_type>**s**)** **+QHTTPCFG: "ssl",(**range of supported **<SSL_contextID>**s**),(**range of supported **<SSL_connectID>**s**)** **OK**
Read Command **AT+QHTTPCFG?**	Response **+QHTTPCFG: "contextid",<contextID>** **+QHTTPCFG: "requestheader",<request_header>** **+QHTTPCFG: "responseheader",<response_header>** **+QHTTPCFG: "contenttype",<content_type>** **+QHTTPCFG: "ssl",<SSL_contextID>,<SSL_connectID>** **OK**
Write Command Configure/query the context ID. **AT+QHTTPCFG="contextid"[,<contextID>]**	Response If the optional parameter is omitted, query the current settings: **+QHTTPCFG: "contextid",<contextID>**

	OK If the optional parameter is specified, configure the context ID: **OK** Or **ERROR**
Write Command Configure/query whether to enable to customize HTTP(S) request header. **AT+QHTTPCFG="requestheader"[,<request_header>]**	Response If the optional parameter is omitted, query the current settings: **+QHTTPCFG: "requestheader",<request_header>** **OK** If the optional parameter is specified, enable or disable to customize HTTP(S) request header: **OK** Or **ERROR**
Write Command Configure/query whether to enable to output HTTP(S) response header. **AT+QHTTPCFG="responseheader"[,<response_header>]**	Response If the optional parameter is omitted, query the current settings: **+QHTTPCFG: "responseheader",<response_header>** **OK** If the optional parameter is specified, enable or disable to output HTTP(S) response header: **OK** Or **ERROR**
Write Command Configure/query SSL context ID and connection ID. **AT+QHTTPCFG="ssl"[,<SSL_contextID>,<SSL_connectID>]**	Response If the optional parameters are omitted, query the current settings: **+QHTTPCFG: "ssl",<SSL_contextID>,<SSL_connectID>** **OK** If the optional parameters are specified, configure SSL context ID and connection ID: **OK** Or **ERROR**
Maximum Response Time	300 ms
Characteristics	The command takes effect immediately. The configurations will not be saved.

Parameter

<contextID>	Integer type. PDP context ID. Range: 1–3 (only 1 is supported currently). Default value: 1.
<request_header>	Integer type. Disable or enable to customize HTTP(S) request header. 0 Disable 1 Enable
<response_header>	Integer type. Disable or enable to output HTTP(S) response header. 0 Disable 1 Enable
<content_type>	Integer type. Data type of HTTP(S) body. 0 application/x-www-form-urlencoded 1 text/plain 2 application/octet-stream 3 multipart/form-data
<SSL_contextID>	Integer type. SSL context index. Range: 1–3. Default value: 1.
<SSL_connectID>	Integer type. SSL connection index. Range: 0–5. Default value: 0.

NOTE

The SSL/TLS connection configurations must be set by **AT+QSSLCFG**. For details of the command, see ***document [1]***

2.3.2. AT+QHTTPURL Set URL of HTTP(S) Server

This command sets a URL of HTTP(S) server. A URL must begin with http:// or https://, which indicates the access to an HTTP or HTTPS server.

AT+QHTTPURL Set URL of HTTP(S) Server	
Test Command **AT+QHTTPURL=?**	Response **+QHTTPURL: (**range of supported **<URL_length>**s**),(**range of supported **<timeout>**s**)** **OK**
Read Command **AT+QHTTPURL?**	Response **+QHTTPURL: <URL>** **OK**
Write Command **AT+QHTTPURL=<URL_length>[,<timeout>]**	Response a) If the parameter format is right and HTTP(S) GET/POST requests are not sent: **>** After **>** is returned, the module enters data mode. Then type

	the URL. When the total size of the input data reaches the **<URL_length>**, the module returns to command mode and reports the following code: **OK** If the **<timeout>** has reached, but the length of the received URL is less than the **<URL_length>**, the module returns to command mode and reports the following code: **ERROR** b) In case of incorrect parameter format or other errors: **ERROR**
Maximum Response Time	300 ms
Characteristics	The command takes effect immediately. The configurations will not be saved.

Parameter

<URL_length>	Integer type. Length of a URL. Range: 1–700. Unit: byte.
<timeout>	Integer type. The maximum time for inputting a URL. Range: 1–300. Default value: 60. Unit: second.

2.3.3. AT+QHTTPGET Send GET Request to HTTP(S) Server

This command sends GET request to HTTP(S) server. It is not supported to customize GET request header. If **<request_header>** is set to 1, executing **AT+QHTTPGET** responses **ERROR**. You can use **AT+QHTTPPOST** (see ***Chapter 2.3.4***) to send the custom HTTP(S) GET packet.

After **AT+QHTTPGET** Write Command has been sent, it is suggested to wait for a specific period of time (refer to the maximum response time below) for **+QHTTPGET: <err>[,<HTTP_rspcode>[,<content_length>]]** to be output after **OK** is reported.

In **+QHTTPGET: <err>[,<HTTP_rspcode>[,<content_length>]]**, the **<HTTP_rspcode>** can only be reported when **<err>** equals 0.

AT+QHTTPGET Send GET Request to HTTP(S) Server	
Test Command **AT+QHTTPGET=?**	Response **+QHTTPGET: (**range of supported **<rsptime>**s**)** **OK**
Write Command **AT+QHTTPGET=[<rsptime>]**	Response a) If the parameter format is right and no other errors occur:

	OK When the module has received response from HTTP(S) server , it reports the following URC: **+QHTTPGET: <err>[,<HTTP_rspcode>[,<content_length>]]** b) In case of incorrect parameter format or other errors: **ERROR**
Maximum Response Time	Determined by **<rsptime>.**
Characteristics	-

Parameter

<rsptime>	Integer type. Timeout for the URC **+QHTTPGET: <err>[,<HTTP_rspcode>[,<content_length>]]** to be outputted after **OK** is returned. Range: 1–300. Default value: 60. Unit: second.
<HTTP_rspcode>	HTTP(S) response code. See ***Chapter 5***.
<content_length>	Integer type. Length of HTTP(S) response body. This parameter is only returned when HTTP(S) response header contains CONTENT-LENGTH information. Unit: byte.
<err>	Integer type. Error code of the operation. See ***Chapter 4***.

2.3.4. AT+QHTTPPOST Send POST Request to HTTP(S) Server via UART/USB

The command sends POST request to HTTP(S) server. According to the configured **<request_header>** in **AT+QHTTPCFG="requestheader"[,<request_header>]**, the **AT+QHTTPPOST** Write Command has two different formats (see ***Chapter 2.3.1***). If **<request_header>** is set to 0, HTTP(S) POST body should be inputted via UART/USB port. If **<request_header>** is set to 1, both HTTP(S) POST header and body should be inputted via UART/USB port.

After **AT+QHTTPPOST** has been sent, **>** may be outputted in 50 s to indicate that the connection is successful. If **>** is not received during the time, it is a socket error and the module responds **+QHTTPPOST: 716**.

It is recommended to wait for a specific period of time (refer to the maximum response time below) for **+QHTTPPOST: <err>[,<HTTP_rspcode>[,<content_length>]]** to be outputted after **OK** is reported.

AT+QHTTPPOST Send POST Request to HTTP(S) Server via UART/USB	
Test Command **AT+QHTTPPOST=?**	Response **+QHTTPPOST: (**range of supported **<data_length>**s**)**

	OK
If <request_header> is set to 0 (disable to customize HTTP(S) request header) Write Command **AT+QHTTPPOST=<data_length>[,<input_time>,<rsptime>[,<flag>]]**	Response a) If the parameter format is right, HTTP(S) server is connected successfully and HTTP(S) request header is sent completely: **>** After **>** is returned, the module switches to data mode, and the HTTP(S) request body can be inputted. When the total size of the inputted data reaches **<data_length>**, the module returns to command mode and reports the following code: **OK** When the module has received response from HTTP(S) server, it reports the following URC: **+QHTTPPOST: <err>[,<HTTP_rspcode>[,<content_length>]]** If the **<input_time>** has reached, but the received length of data is less than **<data_length>**, the module returns to command mode and reports the following code: **ERROR** b) In case of incorrect parameter format or other errors: **ERROR**
If <request_header> is set to 1 (enable to customize HTTP(S) request header) Write Command **AT+QHTTPPOST=<data_length>[,<input_time>,<rsptime>[,<flag>]]**	Response a) If the parameter format is right, HTTP(S) server is connected successfully: **>** After **>** is returned, the module switches to data mode, and the HTTP(S) request header and body can be inputted. When the total size of the inputted data reaches **<data_length>**, the module returns to command mode and reports the following code: **OK** When the module has received response from HTTP(S) server, it reports the following URC: **+QHTTPPOST: <err>[,<HTTP_rspcode>[,<content_length>]]** If the **<input_time>** has reached, but the received length of data is less than **<data_length>**, the module returns to command mode and reports the following code: **ERROR** b) In case of incorrect parameter format or other errors: **ERROR**

Maximum Response Time	Determined by network and **<rsptime>**.
Characteristics	-

Parameter

<data_length>	Integer type. If **<request_header>** is 0, it indicates the length of HTTP(S) POST body. If **<request_header>** is 1, it indicates the length of HTTP(S) POST request information, including HTTP(S) request header and HTTP(S) request body. Range: 1–2048. Unit: byte.
<input_time>	Integer type. The maximum time for inputting HTTP(S) POST body or HTTP(S) POST request information. Range: 1–300. Default value: 60. Unit: second.
<rsptime>	Integer type. Timeout for the URC **+QHTTPPOST: <err>[,<HTTP_rspcode>[,<content_length>]]** to be outputted after **OK** is returned. Range: 1–300. Default value: 60. Unit: second.
<flag>	Integer type. Whether the current packet is the last packet. 0 The packet is the last one. 1 The packet is not the last one.
<HTTP_rspcode>	HTTP(S) response code. See ***Chapter 5***.
<content_length>	Integer type. Length of HTTP(S) response body. Unit: byte.
<err>	Integer type. Error code of the operation. See ***Chapter 4***.

2.3.5. AT+QHTTPREAD Read Response from HTTP(S) Server via UART/USB

This command reads response from HTTP(S) server via UART/USB. After sending HTTP(S) GET/POST requests, HTTP(S) response information can be retrieved from HTTP(S) server via UART/USB port by **AT+QHTTPREAD**. **+QHTTPGET: <err>[,<HTTP_rspcode>[,<content_length>]]** or **+QHTTPPOST: <err>[,<HTTP_rspcode>[,<content_length>]]** must be received before executing **AT+QHTTPREAD** command.

AT+QHTTPREAD Read Response from HTTP(S) Server via UART/USB	
Test Command **AT+QHTTPREAD=?**	Response **+QHTTPREAD: (**range of supported **<read_length>**s**)** **OK**
Write Command **AT+QHTTPREAD=<read_length>**	Response a) If the parameter format is right and the server response is read successfully: **+QHTTPREAD:** **<actual_read_length>,<remaining_length>** **<response_information>**

	OK b) In case of incorrect parameter format or other errors: **ERROR**
Maximum Response Time	Determined by network and **<rsptime>**.
Characteristics	-

Parameter

<read_length>	Integer type. Length of data requested to be read. Range: 1–1024. Default value: 1024. Unit: byte.
<actual_read_length>	Integer type. Actual length of the received data. Unit: byte.
<remaining_length>	Integer type. Remaining length of the last received data. Unit: byte.
<response_information>	String type. HTTP(S) response information, including the HTTP(S) response header.
<err>	Integer type. Error code of the operation. See ***Chapter 4***.

3 Examples

3.1. Access to HTTP Server

3.1.1. Send HTTP GET Request and Read the Response

The following examples show how to send HTTP GET request and how to read HTTP GET response.

```
//Example of how to send HTTP GET request.
AT+QSCLK=0                          //Disable the module to enter sleep mode.
OK
AT+QHTTPCFG="contextid",1           //Configure the PDP context ID as 1.
OK
AT+QHTTPCFG="responseheader",1      //Enable to output HTTP response header.
OK
AT+QHTTPURL=19,80                   //Set the URL of HTTP server to be accessed.
>
http://example.com/                 //Input URL whose length is 19 bytes.

OK
AT+QHTTPGET=80                      //Send HTTP GET request and set the maximum response time
                                      of HTTP GET request to 80 s.
OK

+QHTTPGET: 0,200,1256               //If HTTP response header contains CONTENT-LENGTH
                                      information, the <content_length> is reported.

//Example of how to read HTTP response.
//Read HTTP response information via UART port.
AT+QHTTPREAD=80                     //Read 80 bytes of HTTP response information via UART.
+QHTTPREAD:
80,1431                             //The actual length of the read data is 80 bytes, and the
                                      remaining length of the HTTP response is 1431 bytes
HTTP/1.1 200 OK
Age: 430547
Cache-Control: max-age=604800
Content-Type: text/
```

```
OK
AT+QSCLK=1                          //Enable the module to enter sleep mode.
OK
```

3.1.2. Send HTTP POST Request and Read the Response

The following examples show how to send HTTP POST request and retrieve post body via UART port, and how to read HTTP POST response.

```
AT+QSCLK=0                          //Disable the module to enter sleep mode.
OK
AT+QHTTPCFG="contextid",1           //Configure the PDP context ID as 1.
OK
AT+QHTTPURL=59,80                   //Set the URL of HTTP server to be accessed.
>
http://api.efxnow.com/DEMOWebServices2.8/Service.asmx/Echo?   //Input URL whose length is 59
                                                               bytes.
OK
AT+QHTTPPOST=20,80,80               //Send HTTP POST request. POST body is obtained via UART. The
                                      maximum time for inputting HTTP POST body is 80 s and the
                                      maximum timeout for HTTP POST response is 80 s.
>
Message=HelloQuectel                //Input HTTP POST body whose length is 20 bytes.

OK

+QHTTPPOST: 0,200,177               //If the HTTP response header contains CONTENT-LENGTH, the
                                      <content_length> is reported.

//Example of how to read HTTP response.
AT+QHTTPREAD=80                     //Read 80 bytes of HTTP response body via UART.
+HTTPREAD:
80,97                               //The actual length of the read data is 80 bytes, and the remaining
                                      length of the HTTP response is 97 bytes.
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="https://api.efxnow.co

OK
AT+QSCLK=1                          //Enable the module to enter sleep mode.
OK
```

3.2. Access to HTTPS Server

3.2.1. Send HTTPS GET Request and Read the Response

The following examples show how to send HTTPS GET request and how to read HTTPS GET response.

```
//Example of how to send HTTPS GET request.
RDY

+CFUN: 1

+CPIN: READY

+IP: 100.111.131.221
AT+QSCLK=0                      //Disable the module to enter sleep mode.
OK
AT+QSSLCFG=1,5,"seclevel",1     //Configure the authentication mode to manage server authentication
                                  for SSL context 1.
OK
AT+QSSLCFG=1,5,"cacert"         //Configure CA certificate.
>                               //Input the content of the trusted CA certificate in PEM format. Tap
                                  "Ctrl" + "Z" to send.
+QSSLCFG: 1,5,"cacert",1360

OK
AT+QHTTPCFG="ssl",1,5           //Configure SSL context ID and connection ID as 1 and 5 respectively.
OK
AT+QHTTPCFG="responseheader",1  //Enable to output HTTPS response header.
OK
AT+QHTTPURL=24                  //Set the URL of HTTPS server to be accessed.
>                               //Input URL whose length is 24 bytes.
https://www.example.com/

OK
AT+QHTTPGET=80                  //Send HTTPS GET request and set the maximum response time of
                                  HTTPS GET request to 80 s.
OK

+QHTTPGET: 0,200,1256

//Example of how to read HTTP response.
AT+QHTTPREAD=1024               //Read 1024 bytes of HTTPS response header and body via UART.
+QHTTPREAD:
1024,583                        //The actual length of the read data is 1024 bytes, and the remaining
```

length of the HTTPS response is 583 bytes.

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 557023
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Wed, 06 May 2020 14:04:53 GMT
Etag: "3147526947"
Expires: Wed, 13 May 2020 14:04:53 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (sjc/4E73)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans",
"Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shado

OK
```

AT+QHTTPREAD=1024 //Read 583 bytes of HTTPS response header and body via UART port.

```
+QHTTPREAD:
```

```
583,0                                   //The actual length of the read data is 583 bytes, and the remaining
                                          length of the HTTP response is 0 bytes.
w: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>


OK
AT+QSCLK=1                              //Enable the module to enter sleep mode.
OK
```

3.2.2. Send HTTPS POST Request and Read the Response

The following examples show how to send HTTPS POST request and retrieve post body via UART port, and how to read HTTPS POST response.

```
RDY

+CFUN: 1

+CPIN: READY

+IP: 100.123.237.115
```

```
AT+QSCLK=0                          //Disable the module to enter sleep mode.
OK
AT+QSSLCFG=1,5,"seclevel",1         //Configure the authentication mode to manage server authentication
                                      for SSL context 1.
OK
AT+QSSLCFG=1,5,"cacert"             //Configure CA certificate.
>                                   //Input the content of the trusted CA certificate in PEM format. Tap
                                      "Ctrl" + "Z" to send.
+QSSLCFG: 1,5,"cacert",1250

OK
AT+QHTTPCFG="ssl",1,5               //Configure SSL context ID and connection ID as 1 and 5 respectively.
OK
AT+QHTTPCFG="responseheader",1      //Enable to output HTTPS response header.
OK
AT+QHTTPURL=32                      //Set the URL of HTTPS server to be accessed.
>                                   //Input URL whose length is 32 bytes.
https://api.quectel.com/v1/token

OK
AT+QHTTPPOST=38                     //Send HTTPS POST request. POST body is obtained via UART.
>
appId=xxxxxx&secret=xxxxxx          //Input HTTPS POST body whose length is 38 bytes.

OK

+QHTTPPOST: 0,200                   //HTTPS response header does not contain CONTENT-LENGTH,
                                      the <content_length> is not reported.

//Example of how to read HTTPS response.
AT+QHTTPREAD=1024                   //If response has no <content_length>, wait for maximum 90 s.
+QHTTPREAD:
354,0                               //The actual length of the read data is 354 bytes, and the remaining
                                      length of the HTTP response is 0 bytes.
HTTP/1.1 200 OK
Server: nginx/1.16.1
Date: Wed, 06 May 2020 14:40:44 GMT
Content-Type: application/json;charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Application-Context: quechub-portal:8087
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
```

```
3d
{"code":70029,"msg":"Application information does not exist"}
0

OK
AT+QSCLK=1                    //Enable the module to enter sleep mode.
OK
```

4 Summary of ERROR Codes

The error code **<err>** indicates an error related to mobile equipment or network. The details about **<err>** are described in the following table.

Table 2: Summary of Error Codes

<err>	Meaning
0	Operation successful
701	HTTP(S) unknown error
702	HTTP(S) timeout
703	HTTP(S) busy
704	HTTP(S) UART busy
705	HTTP(S) no GET/POST requests
706	HTTP(S) network busy
707	HTTP(S) network open failed
708	HTTP(S) network no configuration
709	HTTP(S) network deactivated
710	HTTP(S) network error
711	HTTP(S) URL error
712	HTTP(S) empty URL
713	HTTP(S) IP address error

714	HTTP(S) DNS error
715	HTTP(S) socket create error
716	HTTP(S) socket connect error
717	HTTP(S) socket read error
718	HTTP(S) socket write error
719	HTTP(S) socket closed
720	HTTP(S) data encode error
721	HTTP(S) data decode error
722	HTTP(S) read timeout
723	HTTP(S) response failed
726	Input timeout
727	Wait data timeout
728	Wait HTTP(S) response timeout
729	Memory allocation failed
730	Invalid parameter

5 Summary of HTTP(S) Response Codes

<HTTP_rspcode> indicates the response codes from HTTP(S) server. The details about **<HTTP_rspcode>** are described in the following table.

Table 3: Summary of HTTP Response Codes

<HTTP_rspcode>	Meaning
200	OK
403	Forbidden
404	Not found
409	Conflict
411	Length required
500	Internal server error

6 Appendix References

Table 4: Related Document

Document Name	
[1]	Quectel_BC66&BC66-NA_SSL_Application_Note

Table 5: Terms and Abbreviations

Abbreviation	Description
DNS	Domain Name Server
DTR	Data Terminal Ready
HTTP	Hyper Text Transport Protocol
HTTPS	Hypertext Transfer Protocol Secure
PDP	Packet Data Protocol
SSL	Security Socket Layer
TA	Terminal Adapter
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver/Transmitter
URC	Unsolicited Result Code
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus