

BC660K-GL&BC950K-GL

HTTP(S) Application Note

NB-IoT Module Series

Version: 1.0

Date: 2023-06-19

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.

Copyright © Quectel Wireless Solutions Co., Ltd. 2023. All rights reserved.

About the Document

Revision History

Version	Date	Author	Description
-	2023-04-13	Randy LI/ Caden ZHANG	Creation of the document
1.0	2023-06-19	Yance YANG/ Randy LI/ Caden ZHANG	First official release

Contents

About the Document.....	3
Contents.....	4
Table Index.....	5
1 Introduction	6
1.1. Description of HTTP(S) Request Header	6
1.1.1. Customize HTTP(S) Request Header	6
1.1.2. Output HTTP(S) Response Header.....	7
2 Description of HTTP(S) AT Commands.....	8
2.1. AT Command Syntax	8
2.1.1. Definitions.....	8
2.1.2. AT Command Syntax	8
2.2. Declaration of AT Command Examples	9
2.3. AT Command Description	9
2.3.1. AT+QHTTPCFG Configure Parameters for HTTP(S) Server	9
2.3.2. AT+QHTTPURL Set URL of HTTP(S) Server	12
2.3.3. AT+QHTTPGET Send GET Request to HTTP(S) Server	13
2.3.4. AT+QHTTPGETEX Send GET Request to HTTP(S) Server to Get Data With Specified Range.....	14
2.3.5. AT+QHTTPPOST Send POST Request to HTTP(S) Server via UART/USB.....	15
2.3.6. AT+QHTTPREAD Read Response from HTTP(S) Server via UART/USB	17
3 Examples	19
3.1. Access to HTTP Server	19
3.1.1. Send HTTP GET Request and Read the Response	19
3.1.2. Send HTTP POST Request and Read the Response.....	20
3.2. Access to HTTPS Server.....	21
3.2.1. Send HTTPS GET Request and Read the Response.....	21
3.2.2. Send HTTPS POST Request and Read the Response	23
4 Error Handling.....	26
4.1. Executing HTTP(S) AT Commands Fails.....	26
4.2. DNS Parse Fails.....	26
4.3. Entering Data Mode Fails	26
4.4. Sending GET/POST Requests Fails.....	27
4.5. Reading Response Fails.....	27
5 Summary of ERROR Codes	28
6 Summary of HTTP(S) Response Codes	30
7 Appendix and References.....	31

Table Index

Table 1: Types of AT Commands	8
Table 2: Summary of Error Codes.....	28
Table 3: Summary of HTTP Response Codes.....	30
Table 4: Related Documents	31
Table 5: Terms and Abbreviations	31

1 Introduction

Quectel BC660K-GL and BC950K-GL modules support HTTP(S) applications through accessing HTTP(S) servers.

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. It can be customized by configuring `<request_header>` as 1 via `AT+QHTTPCFG` (see [Chapter 2.3.1](#)), and then by inputting HTTP(S) request header (see [Chapter 2.3.5](#)) according to the following requirements:

- Apply HTTP(S) request header syntax.
- The value of URI in HTTP(S) request line and the “Host:” request header must be in line with the URL set with `AT+QHTTPURL`.
- The HTTP(S) request header must end with `<CR><LF>`.

A valid HTTP(S) POST request header is shown in the following example:

```
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 automatically output. Outputting of the HTTP(S) response header can be enabled by setting `<response_header>` to 1 via `AT+QHTTPCFG` (see **Chapter 2.3.1**). The HTTP(S) response header will be output with HTTP(S) response body after executing `AT+QHTTTPREAD` (see **Chapter 2.3.6**).

1.2. Description of Data Mode

BC660K-GL and BC950K-GL support two working modes of the COM port: AT command mode and data mode. In the AT command mode, the data input via the COM port are interpreted as AT commands; whereas in data mode, they are interpreted as data.

By default, the BC660K-GL and BC950K-GL modules operate in AT command mode. After receiving the > response, the modules switch to data mode within 500 ms. To exit data mode and transmit the data to the COM port, enter "Ctrl" + "Z". Alternatively, entering "Esc" will make the module exit data mode and cancel the sending process.

NOTE

1. After receiving the > response, it is recommended for the MCU to wait for 500 ms before sending the data.
2. In data mode, URCs will be lost. To prevent this, please enter the data to be sent immediately after the > response, and promptly exit data mode.

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>**. Throughout this document, only the commands and responses are presented, while carriage return and line feed characters 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

The command configures the parameters for HTTP(S) server, including configuring a PDP context ID, customizing HTTP(S) request header, outputting 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) +QHTTPCFG: "readformat",(list of supported <read_format>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> +QHTTPCFG: "readformat",<read_format> OK

<p>Write Command Set/query the PDP context ID. AT+QHTTPCFG="contextid"[,<contextid>]</p>	<p>Response If the optional parameter is omitted, query the current settings: +QHTTPCFG: "contextid",<contextid></p> <p>OK</p> <p>If the optional parameter is specified, set the context ID: OK Or ERROR</p>
<p>Write Command Set/query whether to enable customizing HTTP(S) request header. AT+QHTTPCFG="requestheader"[,<request_header>]</p>	<p>Response If the optional parameter is omitted, query the current setting: +QHTTPCFG: "requestheader",<request_header></p> <p>OK</p> <p>If the optional parameter is specified, enable or disable customizing HTTP(S) request header: OK Or ERROR</p>
<p>Write Command Set/query whether to enable customizing HTTP(S) response header. AT+QHTTPCFG="responseheader"[,<response_header>]</p>	<p>Response If the optional parameter is omitted, query the current setting: +QHTTPCFG: "responseheader",<response_header></p> <p>OK</p> <p>If the optional parameter is specified, enable or disable customizing HTTP(S) response header: OK Or ERROR</p>
<p>Write Command Set/query data type of HTTP(S) body. AT+QHTTPCFG="contenttype"[,<content_type>]</p>	<p>Response If the optional parameter is omitted, query the current setting: +QHTTPCFG: "contenttype",<content_type></p> <p>OK</p> <p>If the optional parameter is specified, set data type of HTTP(S) body: OK Or ERROR</p>

<p>Write Command Set/query SSL context ID and connection ID. AT+QHTTPCFG="ssl",<SSL_contextID>,<SSL_connectID>]</p>	<p>Response If the optional parameters are omitted, query the current setting: +QHTTPCFG: "ssl",<SSL_contextID>,<SSL_connectID> OK If the optional parameters are specified, set SSL context ID and connection ID: OK Or ERROR</p>
<p>Write Command AT+QHTTPCFG="readformat",<read_format>]</p>	<p>Response If the optional parameter is omitted, query the current setting: +QHTTPCFG: "readformat",<read_format> OK If the optional parameter is specified, set the display format of the data returned by AT+QHTTPREAD: OK If there is any error: ERROR Or +CME ERROR: <result></p>
<p>Maximum Response Time</p>	<p>300 ms</p>
<p>Characteristics</p>	<p>The command takes effect immediately. The configurations are not saved.</p>

Parameter

<p><contextID></p>	<p>Integer type. PDP context ID. Range: 0–10 (currently only 0 is supported).</p>
<p><request_header></p>	<p>Integer type. Disable or enable customizing HTTP(S) request header. <u>0</u> Disable 1 Enable</p>
<p><response_header></p>	<p>Integer type. Disable or enable outputting HTTP(S) response header. <u>0</u> Disable 1 Enable</p>
<p><content_type></p>	<p>Integer type. Data type of HTTP(S) body. <u>0</u> application/x-www-form-urlencoded 1 text/plain 2 application/octet-stream 3 multipart/form-data</p>

<SSL_contextID>	Integer type. SSL context ID. Range: 0–10 (currently only 0 is supported).
<SSL_connectID>	Integer type. SSL connection ID. Range: 0–4 (currently only 0 is supported).
<read_format>	String type. Indicates whether a carriage return and line feed should be included in AT+QHTTPREAD response. <u>0</u> No 1 Yes
<result>	Integer type. Result code. See Chapter 5 .

NOTE

1. SSL/TLS connection configurations must be set by **AT+QSSLCFG**. For details of the command, see **document [1]**.
2. Currently only default **<contextID>**, **<SSL_contextID>** and **<SSL_connectID>** are supported.
3. Due to chip space limitation, currently HTTPS only supports one-way authentication and no authentication. Two-way authentication is not supported.

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

This command sets URL of HTTP(S) server. URL must begin with "http://" or "https://", which indicates that an HTTP or HTTPS server will be accessed.

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 correct, but HTTP(S) GET/POST requests are not being sent: > After receiving the > response, the module enters data mode, and the URL can be input. When the total size of the input data reaches <URL_length> , the module returns to command mode and responds with: OK If <timeout> has been reached, but the length of the received URL is less than <URL_length> , the module returns to

	command mode and responds with: ERROR
	b) If the parameter format is incorrect or other errors occur: ERROR
Maximum Response Time	300 ms
Characteristics	The command takes effect immediately. The configurations are not saved.

Parameter

<URL_length>	Integer type. Length of URL. Range: 1–256. Unit: byte.
<timeout>	Integer type. 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 a GET request to HTTP(S) server. The format of the command depends on the configured **<request_header>** in **AT+QHTTPCFG="requestheader" [<request_header>]** (see **Chapter 2.3.1**). Customizing GET request header is not supported. If **<request_header>** is set to 1, executing **AT+QHTTPGET** will result in an **ERROR** response. In such cases, you can use **AT+QHTTPPOST** (see **Chapter 2.3.5**) to send a custom HTTP(S) GET packet.

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

<HTTP_rspcode> can only be reported in **+QHTTPGET: <result>[,<HTTP_rspcode>[,<content_length>]]**, when **<result>** is 0. If HTTP(S) response header contains content-length information, it will be reported as **<content_length>**.

AT+QHTTPGET Send GET Request to HTTP(S) Server	
Test Command AT+QHTTPGET=?	Response +QHTTPGET: (range of supported <rsptime>s),(range of supported <read_timeout>s) OK
Write Command AT+QHTTPGET[=<rsptime>[,<read_timeout>]]	Response If the parameter format is correct and no other errors occur: OK When the module receives a response from HTTP(S) server, it

	<p>reports the following URC: +QHTTPGET: <result>[,<HTTP_rspcode>[,<content_length>]]</p> <p>If there is any error: ERROR Or +CME ERROR: <result></p>
Maximum Response Time	Determined by <rsptime>
Characteristics	/

Parameter

<rsptime>	Integer type. Timeout for the HTTP(S) GET response +QHTTPGET: <result>[,<HTTP_rspcode>[,<content_length>]] to be output after OK is returned. Range: 1–300. Default value: 60. Unit: second.
<read_timeout>	Integer type. Maximum time for executing AT+QHTTPGET before releasing the HTTP resources. Range: 1–300. Default value: 60. Unit: second.
<result>	Integer type. Result code. See Chapter 5 .
<HTTP_rspcode>	Integer type. HTTP(S) response code. See Chapter 6 .
<content_length>	Integer type. Length of HTTP(S) response body. Unit: byte.

2.3.4. AT+QHTTPGETEX Send GET Request to HTTP(S) Server to Get Data With Specified Range

MCU can retrieve data with a specific position and length from HTTP(S) server by using **AT+QHTTPGETEX**. This command is only executable when **AT+QHTTPCFG="requestheader",0** configuration is set. After sending the command, HTTP(S) server will always respond to the GET request for retrieving data with a specified position and length, by returning a **206** response code.

AT+QHTTPGETEX Send GET Request to HTTP(S) Server to Get Data With Specified Range

Test Command AT+QHTTPGETEX=?	Response +QHTTPGETEX: (range of supported <rsptime>s),<start_position>,<read_len>,(range of supported <read_timeout>s) OK
Write Command AT+QHTTPGETEX=<rsptime>,<start_position>,<read_len>[,<read_timeout	Response If the parameter format is correct and no other errors occur: OK

>]	<p>When the module receives a response from HTTP(S) server, it will report the following URC: +QHTTPGETEX: <result>[,<HTTP_rspcode>,<content_length>]</p> <p>If there is any error: ERROR Or +CME ERROR: <result></p>
Maximum Response Time	Determined by <rsptime>
Characteristics Description	/

Parameter

<rsptime>	Integer type. Timeout for the HTTP(S) GET response +QHTTPGETEX: <result>,<HTTP_rspcode>[,<content_length>] to be output after OK is returned. Range: 1–300. Default: 60. Unit: second.
<start_postion>	Integer type. The start position of the data that the HTTP(S) client wants to get.
<read_len>	Integer type. The length of the data that the HTTP(S) client wants to get.
<read_timeout>	Integer type. Maximum time for executing AT+QHTTPGETEX before releasing the resources. Range: 1–300. Default value: 60. Unit: second.
<result>	Integer type. Result code. See Chapter 5 .
<HTTP_rspcode>	Integer type. HTTP response code. See Chapter 6 for details.
<content_length>	Integer type. The length of HTTP(S) response body. Unit: byte.

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

The command sends a POST request to an HTTP(S) server. Depending on the configuration of **<request_header>** in **AT+QHTTPCFG="requestheader"[,<request_header>]**, **AT+QHTTPPOST** Write Command can have two different formats (see **Chapter 2.3.1**):

- If **<request_header>** is set to 0, HTTP(S) POST body should be input via UART/USB port.
- If **<request_header>** is set to 1, both HTTP(S) POST header and body should be input via UART/USB port.

After **AT+QHTTPPOST** is sent, the module may output **>** within 50 s to indicate a successful connection. If **>** is not received within this time, it indicates a socket error and the module responds with **+QHTTPPOST: 716**. It is recommended to wait for a specific period of time (refer to the maximum response time below) for **+QHTTPPOST: <result>[,<HTTP_rspcode>[,<content_length>]]** to be output after **OK** is returned.

AT+QHTTPPOST Send POST Request to HTTP(S) Server via UART/USB	
Test Command AT+QHTTPPOST=?	<p>Response</p> <p>+QHTTPPOST: (range of supported <data_length>s),(range of supported <input_time>s), (range of supported <rsptime>s),(list of supported <flag>s),(range of supported <read_timeout>s)</p> <p>OK</p>
Write Command AT+QHTTPPOST=<data_length>[,<input_time>,<rsptime>[,<flag>[,<read_timeout>]]]	<p>Response</p> <p>a) If the parameter format is correct, HTTP(S) server is connected successfully and HTTP(S) request header is sent:</p> <p>></p> <p>After > is returned, the module switches to data mode, and the HTTP(S) POST body can be input. When the total size of the input data reaches <data_length>, the module returns to command mode and responds with:</p> <p>OK</p> <p>When the module receives a response from HTTP(S) server, it reports the following URC:</p> <p>+QHTTPPOST: <result>[,<HTTP_rspcode>[,<content_length>]]</p> <p>If the <input_time> has been reached, but the received length of data is less than <data_length>, the module returns to command mode and responds with:</p> <p>ERROR</p> <p>b) If the parameter format is incorrect or other errors occur:</p> <p>ERROR</p>
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. 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 HTTP(S) POST response +QHTTPPOST: <result>[,<HTTP_rspcode>[,<content_length>]] to be output after OK is returned. Range: 1–300. Default value: 60. Unit: second.
<flag>	Integer type. Whether the current packet is the last packet. 0 Packet is the last one 1 Packet is not the last one
<read_timeout>	Integer type. Maximum time for executing AT+QHTTPPOST before releasing the HTTP resources. Range: 1–300. Default value: 60. Unit: second.
<result>	Integer type. Result code. See Chapter 5 .
<HTTP_rspcode>	Integer type. HTTP(S) response code. See Chapter 6 .
<content_length>	Integer type. Length of HTTP(S) response body. Unit: byte.

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

This command retrieves the HTTP(S) server response via the UART/USB port, after HTTP(S) GET/POST requests are sent. It must be executed after one of the following URCs is received.

- **+QHTTPGET: <result>[,<HTTP_rspcode>[,<content_length>]]**
- **+QHTTPPOST: <result>[,<HTTP_rspcode>[,<content_length>]]**

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 If the parameter format is correct and the server response is read successfully: +QHTTPREAD: <actual_read_length>,<remaining_length> <Output HTTP(S) response information> OK If the parameter format is incorrect or other errors occur: 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 received data. Unit: byte.
<remaining_length>	Integer type. Remaining length of last received data. Unit: byte.

3 Examples

3.1. Access to HTTP Server

3.1.1. Send HTTP GET Request and Read Response

The following examples show how to send HTTP GET request with a custom HTTP request header and how to read HTTP GET response.

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

OK
AT+QHHTTPGET=80 //Send HTTP GET request and set the maximum response time
of HTTP GET request to 80 s.
OK
+QHHTTPGET: 0,200,1256 //If HTTP response header contains CONTENT-LENGTH
information, <content_length> is returned.

//Example of how to read HTTP response.
//Read HTTP response information via UART port.
AT+QHHTTPREAD=80 //Read 80 bytes of HTTP response information via UART.
+QHHTTPREAD: 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 sleep mode.
OK
```

3.1.2. Send HTTP POST Request and Read 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 sleep mode.
OK
AT+QHHTTPCFG="contextid",0 //Set the PDP context ID to 0.
OK
AT+QHHTTPURL=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+QHHTTPPOST=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
+QHHTTPPOST: 0,200,177 //If the HTTP response header contains CONTENT-LENGTH,
// <content_length> is returned.
//Example of how to read HTTP response.
AT+QHHTTPREAD=80 //Read 80 bytes of HTTP response body via UART.
+HHTTPREAD: 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="httpHTTPs://api.efxnow.co
OK
AT+QSCLK=1 //Enable sleep mode.
OK
```

3.2. Access to HTTPS Server

3.2.1. Send HTTPS GET Request and Read Response

The following examples show how to send HTTPS GET request with a custom HTTPS request header and how to read HTTPS GET response.

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

+CFUN: 1

+CPIN: READY
AT+CGPADDR
+CGPADDR: 1,"26.186.218.184"

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

OK
AT+QHTTPCFG="ssl",0,0 //Set SSL context ID and connection ID to 0 and 0 respectively.
OK
AT+QHTTPCFG="responseheader",1 //Enable outputting of HTTPS response header.
OK
AT+QHTTPURL=24 //Set the URL of HTTPS server to be accessed
>
https://www.example.com/ //Input URL whose length is 24 bytes.

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=583 //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 sleep mode.
OK
```

3.2.2. Send HTTPS POST Request and Read 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
AT+CGPADDR
+CGPADDR: 1, "26.186.219.185"
```



```

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

OK
AT+QHHTPCFG="ssl",0,0 //Set SSL context ID and connection ID to 0 and 0 respectively.
OK
AT+QHHTPCFG="responseheader",1 //Enable outputting of HTTPS response header.
OK
AT+QHHTPURL=32 //Set the URL of HTTPS server to be accessed.
>
https://api.quectel.com/v1/token //Input URL whose length is 32 bytes.

OK
AT+QHHTPPOST=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

+QHHTPPOST: 0,200 //HTTPS response header does not contain content-length,
<content_length> is not reported.

//Example of how to read HTTPS response.
AT+QHHTPREAD=1024 //If response has no <content_length>, wait for maximum 90 s.
+QHHTPREAD: 354,0 //The actual length of 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 sleep mode.
OK
```

4 Error Handling

4.1. Executing HTTP(S) AT Commands Failure

If **ERROR** response is received from the module after executing HTTP(S) AT commands, check whether the (U)SIM card is inserted and whether **+CPIN: READY** is returned after executing **AT+CPIN?**.

4.2. DNS Parse Failure

If **714** (HTTP(S) DNS error) is returned after executing **AT+QHTTPGET** or **AT+QHTTPPOST**, check the following:

1. Make sure the domain name of HTTP(S) server is valid.
2. Query the status of the PDP context with **AT+CGATT?** and **AT+CGPADDR** sequentially to make sure the specified PDP context has been activated successfully.
3. Query the address of the DNS server with **AT+QIDNSCFG** to make sure the DNS server address is not "0.0.0.0".

If the DNS server address is null or "0.0.0.0", there are two solutions:

1. Reassign a valid DNS server address by **AT+QIDNSCFG**.
2. Deactivate the PDP context with **AT+CFUN=0**, and re-activate the PDP context via **AT+CFUN=1**.

NOTE

For details of above commands, see *document [1]*.

4.3. Entering Data Mode Failure

If **704** (HTTP(S) UART busy) is returned after executing **AT+QHTTPURL** or **AT+QHTTPPOST**, check if there are other ports in data mode, since the module only supports one port in data mode at a time. If any,

re-execute these commands after other ports have exited data mode.

4.4. Sending GET/POST Request Failure

If a failed result is received after executing **AT+QHTTPGET**, **AT+QHTTPGETEX** or **AT+QHTTPPOST**, check the following configurations:

1. Make sure the URL input via **AT+QHTTPURL** is valid and can be accessed.
2. Make sure the specified server supports GET/POST requests.
3. Make sure the PDP context has been activated successfully.

If all above configurations are correct, but sending GET/POST requests with **AT+QHTTPGET**, **AT+QHTTPPOST** still fails, deactivate the PDP context with **AT+CFUN=0** and re-activate it with **AT+CFUN=1** to resolve this issue. If activating the PDP context fails, see **Chapter 4.2** to resolve this issue.

4.5. Reading Response Failure

Before reading responses with **AT+QHTTPREAD**, execute **AT+QHTTPGET**, and **AT+QHTTPPOST**, and the following URC information will be reported:

```
+QHTTPGET: <result>,<HTTP_rspcode>[,<content_length>]
+QHTTPPOST: <result>,<HTTP_rspcode>[,<content_length>]
```

In case of errors during execution of **AT+QHTTPREAD**, such as: **717** (HTTP(S) socket read error), resend HTTP(S) GET/POST requests to HTTP(S) server with **AT+QHTTPGET**, and **AT+QHTTPPOST**. If the sending of GET/POST requests to HTTP(S) server fails, see **Chapter 4.4** to resolve this issue.

5 Summary of Result Codes

The result code **<result>** indicates a result related to mobile equipment or network operation. The details about **<result>** are described in the following table.

Table 2: Summary of Error Codes

<result>	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

6 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

7 Appendix and References

Table 4: Related Documents

Document Name
[1] Quectel_BC660K-GL&BC950K-GL_SSL_Application_Note
[2] Quectel_BC660K-GL&BC950K-GL_AT_Commands_Manual

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