QuecPython HTTP Application Note

LTE Standard Module Series

Version: 1.0.0

Date: 2020-11-09

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

About the Document

History

Revision	Date	Author	Description
-	2020-11-09	Kingka/Kenney	Creation of the document
1.0.0	2020-11-09	Kingka/Kenney	Preliminary

Contents

Table Index

Figure Index

1 Introduction

The HTTP protocol is used for communication between the client and the server, and the communication process is mainly completed by switching between the client request and the server response. This document describes how to use the QuecPython class library API to quickly develop the HTTP protocol on QuecPython platform.

2 HTTP Protocol Basis

2.1. HTTP Protocol

HTTP protocol is used to transfer hypertext from the WWW server to the local browser. The application layer protocol based on TCP doesn't care about the details of data transmission. HTTP is a stateless, application layer protocol based on request and response format. Only by following the unified HTTP request format can the server correctly parse the requests from different clients. Similarly, if the server follows the unified response format, the client can correctly analyze the responses from different websites.

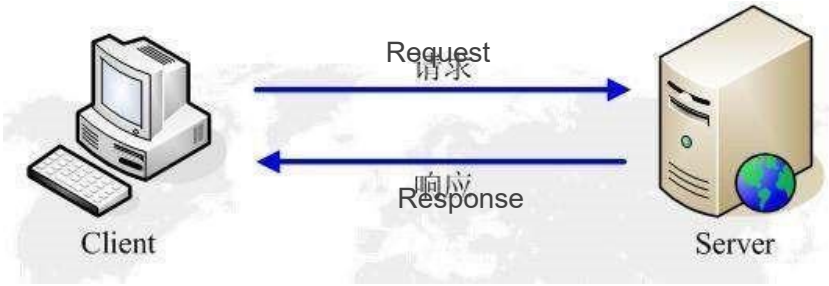

Figure 1: The Communication between the Client and the Server

2.2. HTTP Request

HTTP request consists of Request Line, Request Header, Blank Line, and Request Body.

Figure 2: HTTP Request Format

2.2.1. Request Line

A Request Line specifies the Method Token followed by the Request URL and then the HTTP Protocol that is being used.

- Common Method Token: GET, POST, PUT, DELETE, HEAD.
- URL: The source path to be obtained by the client.
- HTTP Protocol: The HTTP Protocol version currently being used. (Currently is HTTP 1.1)

2.2.2. Request Header

Request Header is the part of the HTTP Request where additional content can be sent to the server.

- host: The request address.
- User-Agent: The name and version of the operating system and browser the client currently used.
- Content-Length: The length of the data sent to the HTTP server.
- Content-Type: The data type of the parameter.
- Cookie：Send the cookie value to the HTTP server.
- Accept-Charset: The character can be accepted by the browser.
- Accept-Language: The language can be accepted by the browser.
- Accept: The media type can be accepted by the browser.

2.2.3. Request Body

The Request Body carries parameter to process current request properly.

- application/json: {"name":"value","name1":"value2”}.
- application/x-www-form-urlencoded: name1=value1&name2=value2.
- multipart/from-data.
- text/xml.
- content-type: octets/stream.

2.3. HTTP Response

HTTP response consists of Status Line, Response Header, Blank Line, and Response Body.

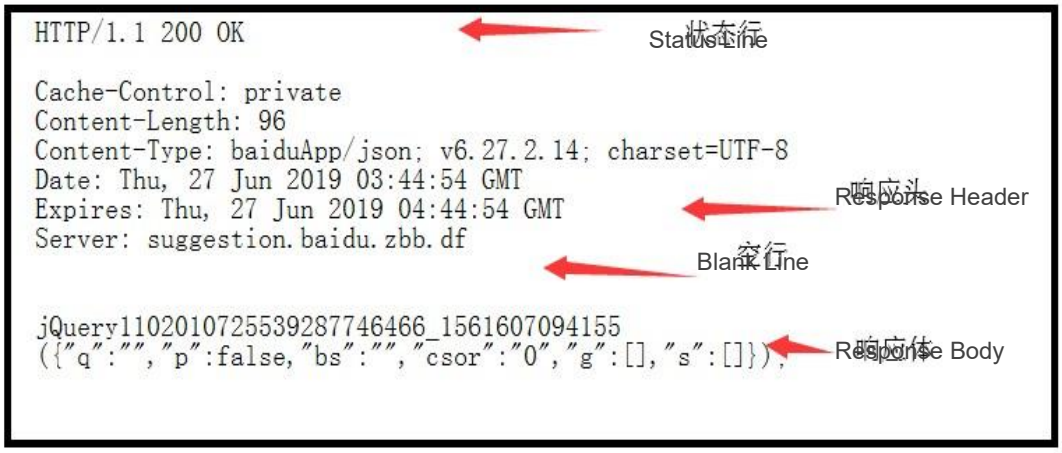

Figure 3: HTTP Response Format

2.3.1. Status Line

A Status Line consists of three parts: HTTP Protocol Version, Status Code, Reason Phrase.

Status Code are 1XX, 2XX, 3XX, 4XX, 5XX.

- 1XX: Informational – It means the request has been received and the process is continuing.
- 2XX: Success – It means the action was successfully received, understood and accepted.
- 3XX: Redirection – It means further action must be taken in order to complete the request.
- 4XX: Client Error – It means the request contains incorrect syntax or cannot be fulfilled.
- 5XX: Server Error - It means the server failed to fulfill an apparently valid request.

2.3.2. Response Header

Just like a Request Header, Request Header is the additional content of the HTTP Response.

2.3.3. Response Body

Response Body is the real response data, that is, the HTML source code of the web page.

2.4. URL

A URL (Uniform Resource Locator) is a unique identifier used to locate a resource on the internet.

URL format: https://host:port/path?xxx=aaa&ooo=bbb.

- http/https: The protocol or scheme.
- host: IP address or domain name of the server.
- port: The port of HTTP server. The default is 80.
- path: A path refers to file or location on the server.
- The question mark in the URL. This symbol is a divider to distinguish between the path in front of the question mark and the parameter after the question mark.
- url-params: After the question mark are the request parameters. Format: xxx=aaa. Multiple parameters can be separated by ampersands (&)

The set of common methods for HTTP/1.1 is defined below.

- **GET**: The GET method is used to retrieve information from the given server using a given URL.
- **POST**: A POST request is used to send data to the server, for example, customer information, file upload, etc.
- **HEAD**: Same as GET, but transfers the Status Line and Header Section only.
- **OPTIONS**: Describes the communication options for the target resource.
- **PUT**: Replaces all current representations of the target resource with the uploaded content.
- **DELETE**: Removes all current representations of the target resource given by a URL.
- **TRACE**: Performs a message loop-back test along the path to the target resource.
- **CONNECT**: Establishes a tunnel to the server identified by a given URL.

3 HTTP APIs

3.1. request.get

This function sends the GET request.

- **Prototype**

```
request.get(url, data, json, headers)
```

- **Parameter**

url:
Required. The string URL of the request.

data:
Optional. It is a dictionary as the request string. Default None.

json:
Optional. It is the body in Json format used to attach to the request. Default None.

headers:
Optional. It is a dictionary of HTTP headers to send to the specified URL. Default None.

- **Return Value**

Returns the request object.

3.2. request.post

This function sends the POST request.

- **Prototype**

```
request.post(url, data, json, headers)
```

- **Parameter**

url:
Required. The string URL of the request.

data:
Optional. It is a dictionary as the request string. Default None.

json:
Optional. It is the body in Json format used to attach to the request. Default None.

headers:
Optional. It is a dictionary of HTTP headers to send to the specified URL. Default None.

- **Return Value**

Returns the request object.

3.3. request.put

This function sends the PUT request.

- **Prototype**

```
request.put(url, data, json, headers)
```

- **Parameter**

url:
Required. The string URL of the request.

data:
Optional. It is a dictionary as the request string. Default None.

json:
Optional. It is the body in Json format used to attach to the request. Default None.

headers:
Optional. It is a dictionary of HTTP headers to send to the specified URL. Default None.

- **Return Value**

Returns the request object.

3.4. request.head

This function sends the HEAD request.

- **Prototype**

```
request.head(url, data, json, headers)
```

- **Parameter**

url:
Required. The string URL of the request.

data:
Optional. It is a dictionary as the request string. Default None.

json:
Optional. It is the body in Json format used to attach to the request. Default None.

headers:
Optional. It is a dictionary of HTTP headers to send to the specified URL. Default None.

- **Return Value**

Returns the request object.

3.5. request.patch

This function sends the PATCH request.

- **Prototype**

```
request.patch(url, data, json, headers)
```

- **Parameter**

url:
Required. The string URL of the request.

data:
Optional. It is a dictionary as the request string. Default None.

json:
Optional. It is the body in Json format used to attach to the request. Default None.

headers:
Optional. It is a dictionary of HTTP headers to send to the specified URL. Default None.

- **Return Value**

Returns the request object.

3.6. request.delete

This function sends the DELETE request.

- **Prototype**

```
request.delete(url, data, json, headers)
```

- **Parameter**

url:
Required. The string URL of the request.

data:
Optional. It is a dictionary as the request string. Default None.

json:
Optional. It is the body in Json format used to attach to the request. Default None.

headers:
Optional. It is a dictionary of HTTP headers to send to the specified URL. Default None.

- **Return Value**

Returns the request object.

3.7. response Class

```
response =request.get(url)
```

Table 1: Response Class

API	Description
response.content	Returns the content of the response. Unit: byte.
response.text	Returns the text content of the response in Unicode.
response.json()	Returns the JSON encoded content of the response and converts it to dict type.
response.close()	Closes the socket.

4 Example

Connect the QuecPython EVB to PC, and please refer to *Quectel_QuecPython_Basic_Operation_Guide*.

After the connection, create *test.py* file and import *request* module. Then write the request codes of HTTP GET/PUT/POST/DELETE respectively, and upload the files to the EVB, run the *test.py* file. Please refer to *Quectel_QuecPython_Basic_Operation_Guide*

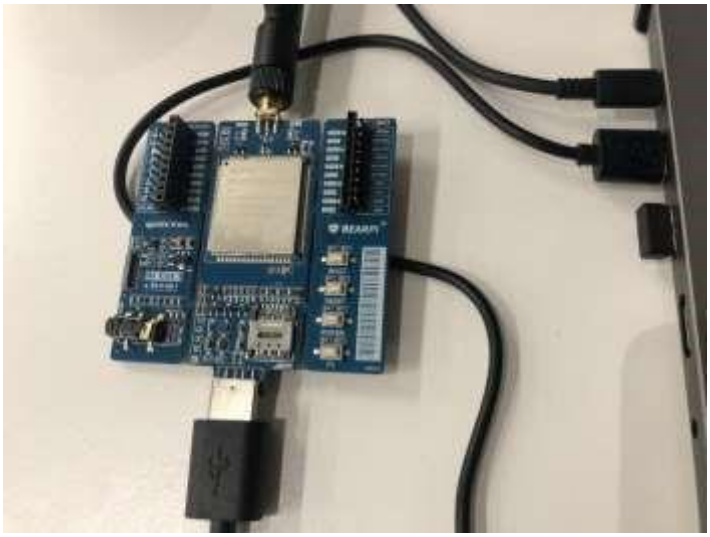

Figure 4: Connect QuecPython EVB to PC

4.1. POST Request

- **Code Example**

```
import request
import ujson

url = "http://httpbin.org/post"
data = {"key1": "value1", "key2": "value2", "key3": "value3"}

# POST Request
response = request.post(url, data=ujson.dumps(data)) print(response.text)
```

- **Running Result**

```
>>> import example
>>> example.exec('test.py')
{
  "args": {},
  "data": "{\"key3\": \"value3\", \"key1\": \"value1\", \"key2\": \"value2\"}",
  "files": {},
  "form": {},
  "headers": {
    "Content-Length": "54",
    "Host": "httpbin.org",
    "X-Amzn-Trace-Id": "Root=1-5f8eaf92-1b8bbde8012cdddd47d7b15f"
  },
  "json": {
    "key1": "value1",
    "key2": "value2",
    "key3": "value3"
  },
  "origin": "36.61.65.119",
  "url": "http://httpbin.org/post"
}

>>>
```

4.2. GET Request

- **Code Example**

```
import request

url = "http://httpbin.org/get"

# GET Request response =
request.get(url)
print(response.text)
```

- **Running Result**

```
>>> import example
>>> example.exec('test.py')
{
  "args": {},
  "headers": {
    "Host": "httpbin.org",
    "X-Amzn-Trace-Id": "Root=1-5f8eb05d-4e742c1b2e51f1286b96c870"
  },
  "origin": "36.61.65.119",
  "url": "https://httpbin.org/get"
}
```

4.3. PUT Request

- **Code Example**

```
import request

url = "http://httpbin.org/put"

# PUT Request response =
request.put(url)
print(response.text)
```

- **Running Result**

```
>>> import example
>>> example.exec('test.py')
{
  "args": {},
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Host": "httpbin.org",
    "X-Amzn-Trace-Id": "Root=1-5f8eb0ed-505e003f408841920b9a935d"
  },
  "json": null,
  "origin": "36.61.65.119",
  "url": "https://httpbin.org/put"
}
```

4.4. PATCH Request

- **Code Example**

```
import request

url = "http://httpbin.org/patch"

# PATCH Request response = request.patch(url) print(response.text)
```

- **Running Result**

```
>>> import example
>>> example.exec('test.py')
{
  "args": {},
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Host": "httpbin.org",
    "X-Amzn-Trace-Id": "Root=1-5f8eb3d2-0e8ed2be20eeeff84a2f58cf"
  },
  "json": null,
  "origin": "36.61.65.119",
  "url": "https://httpbin.org/patch"
}
```

4.5. DELETE Request

- **Code Example**

```
import request

url = "http://httpbin.org/delete"

# DELETE Request response = request.delete(url) print(response.text)
```

- **Running Result**

```
>>>
>>> import example
>>> example.exec('test.py')
{
  "args": {},
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Host": "httpbin.org",
    "X-Amzn-Trace-Id": "Root=1-5f8eb44c-630673217781803043cb8c2c"
  },
  "json": null,
  "origin": "36.61.65.119",
  "url": "https://httpbin.org/delete"
}
```

4.6. Request HTTP Connection

- **Code Example**

```
import request

url = "https://myssl.com"

# HTTPS Request
response = request.get(url)
print(response.text)
```

- **Running Result**

```
>>>
>>> import example
>>> example.exec('test.py')
<!doctype html>
<html class="no-js" lang="zh-CN">

<head>
  <meta charset="utf-8">

  <meta http-equiv="x-dns-prefetch-control" content="on">
  <link rel="dns-prefetch" href="//myssl.com">
  <link rel="dns-prefetch" href="//cdn.bootcss.com">
  <link rel="dns-prefetch" href="//zz.bdstatic.com">
  <link rel="dns-prefetch" href="//script.hotjar.com">
  <link rel="dns-prefetch" href="//static.hotjar.com">
  <link rel="dns-prefetch" href="//browser-update.org">
  <link rel="dns-prefetch" href="//www.google-analytics.com">
  <link rel="dns-prefetch" href="//hm.baidu.com">
```

5 Terms and Abbreviations

Table 2: Terms and Abbreviations

Abbreviation	Description
API	Application Programming Interface
HTTP	Hyper Text Transfer Protocol
SDK	Software Development Kit
TCP	Transmission Control Protocol
URL	Uniform Resource Locator,
WWW	World Wide Web