HTTP/HTTPS Client Application

User guide

Version 0.2

Sep 2016

Redpine Signals, Inc.
2107 N. First Street, #540
San Jose, CA 95131.
Tel: (408) 748-3385
Fax: (408) 705-2019
Email: info@redpinesignals.com
Website: www.redpinesignals.com

About this Document

This document describes the process of bringing up the RS9113 based module as a HTTP/HTTPs client.

Disclaimer:

The information in this document pertains to information related to Redpine Signals, Inc. products. This information is provided as a service to our customers, and may be used for information purposes only. Redpine assumes no liabilities or responsibilities for errors or omissions in this document. This document may be changed at any time at Redpine's sole discretion without any prior notice to anyone. Redpine is not committed to updating this document in the future.

Table of Contents

Table of Figures

Table of Tables

No table of figures entries found.

1 Introduction

This project is applicable to all the WiSeConnect variants like WiSeConnect Plus, WiSeMCU and WyzBee. The term WiSeConnect refers to its appropriate variant.

1.1 Protocol Overview

HTTP client is a client side HTTP transport library. HTTP client's purpose is to transmit and receive HTTP messages. HTTP client will not attempt to process content, execute javascript embedded in HTML pages, try to guess content type, or other functionality unrelated to the HTTP transport.

1.2 Application Overview

1.2.1 Overview

This application demonstrates how to create WiSeConnect device as HTTP/HTTPs client and do HTTP PUT, GET and POST operations with the HTTP/HTTPs server opened on remote peer.

In this application, WiSeconnect device configures as WiFi station and connects to Access point and do HTTP/HTTPs PUT, GET and post operation with HTTP/HTTPs server opened on remote peer.

1.2.2 Sequence of Events

This Application explains user how to:

- Load appropriate CA certificate to the Device to interact with HTTPS Server.
- Connect to Access Point
- Run HTTP/HTTPS Server Remote side.
- Request for HTTP/HTTPs PUT , GET and POST.

1.3 Application Setup

The WiSeConnect in its many variants supports SPI and UART interfaces. Depending on the interface used, the required set up is as below:

1.3.1 SPI based Setup Requirements

- Windows PC1 with CooCox IDE
- Spansion (MB9BF568NBGL) micro controller

Note: If user does not have Spansion (MB9BF568NBGL) host platform, please go through the SPI-Porting guide **\sapis\docs\RS9113-WiSeConnect-SAPI-Porting-Guide-vx.x.pdf** for SAPIs porting to that particular platform.

- WiSeConnect device
- WiFi Access point
- Windows PC2 with openssl support and python installed

Note: Installed python should support the following modules:
Thread, HTTPServer, BaseHTTPRequestHandler, cgi, curdir, sep, sys

1.3.2 UART/USB-CDC based Setup Requirements

- Windows PC with Dev-C++ IDE
- WiSeConnect device
- WiFi Access point
- Windows PC2 with openssl support and python installed

Note: Installed python should support the following modules:
Thread, HTTPServer, BaseHTTPRequestHandler, cgi, curdir, sep, sys

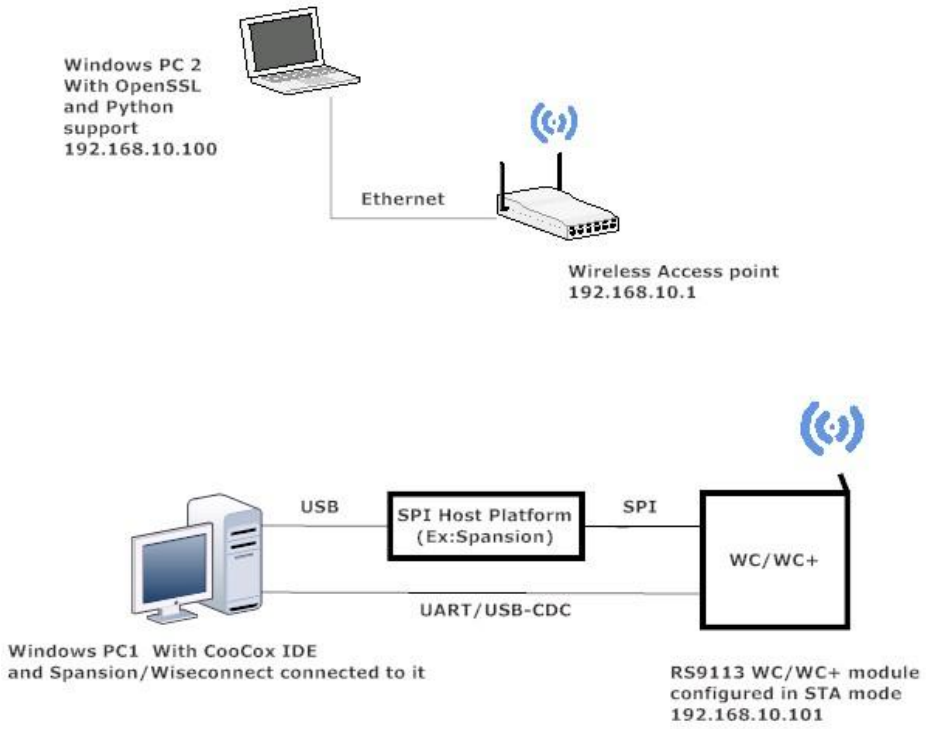


Figure 1: Setup Diagram

2 Configuration and Execution of the Application

The example application is available in the Release at **{Release $}/host/sapis/examples.** These examples will have to be initialized, configured and executed to test the application. The initialization varies based on the interface but configuration and execution are the common.

2.1 Initializing the Application

2.1.1 SPI Interface

If User using SPI interface, Please refer the document ***sapis/platforms/spansion_MB9BF568NBGL/RS9113-WiSeConnect_SAPIS_Spansion_Project_User_guide.pdf*** for opening the ***http_client*** example in CooCox IDE.

2.1.2 UART/USB-CDC Interface

If User using UART interface, Please refer the document ***sapis/platforms/windows_uart/RS9113-WiSeConnect_SAPIS_Windows_Project_UserGuide.pdf*** for opening the ***http_client*** example in Dev-C++ IDE

2.2 Configuring the Application

1. Open ***sapis/examples/wlan/http_client/rsi_http_client_app.c*** file and update/modify following macros,

SSID refers to the name of the Access point.

```
#define SSID                    "<ap name>"
```

CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels

```
#define CHANNEL_NO          0
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.

Valid configuration is

RSI_OPEN - For OPEN security mode

RSI_WPA - For WPA security mode

RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE           RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                     "<psk>"
```

To Load certificate

```
#define  LOAD_CERTIFICATE         1
```

If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API.

By default, application loading "cacert.pem" certificate LOAD_**CERTIFICATE** enable. In order to load different certificate, user has to follow the following steps:

- rsi_wlan_set_certificate API expects the certificate in the form of linear array. So, convert the pem certificate into linear array form using python script provided in the release package "***sapis/examples/utilities/certificates/certificate_script.py***"

  Ex: If the certificate is wifi-user.pem .Give the command in the following way

  `python certificate_script.py ca-cert.pem`

  Script will generate wifiuser.pem in which one linear array named cacert contains the certificate.

- After conversion of certificate, update *rsi_ssl_client.c* source file by including the certificate file and by providing *the* required parameters to rsi_wlan_set_certificate API.
- Once certificate loads into the device, it will write into the device flash. So, user need not load certificate for every boot up unless certificate change.

  So define `LOAD_CERTIFICATE` as 0, if certificate is already present In the Device.

Note: All the certificates are given in the release package
Path: `sapis/examples/utilities/certificates`

To configure IP address

DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE    1
```

Note: If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.

Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define  DEVICE_IP          0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order

Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define  GATEWAY            0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order

Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define    NETMASK            0x00FFFFFF
```

To establish connection and request for HTTP PUT or HTTP GET or HTTP POST to the HTTP/HTTPS Server configure the below macros.

DEVICE_PORT refers internal socket port number.

```
#define DEVICE_PORT                    5001
```

FLAGS refers to open normal HTTP client socket or HTTP client socket over SSL with IPv4 or IPv6

Default configuration of application is normal HTTP client socket with IPv4.

```
#define FLAGS                          0
```

(Or)

If user wants to open HTTP client socket over SSL with IPv4 then set FLAGS to 2 **(HTTPS_SUPPORT)**.

```
#define FLAGS                          HTTPS_SUPPORT
```

(Or)

If user wants to use HTTP client post large data then set FLAGS to 32 **(HTTP_POST_DATA)**.

```
#define FLAGS                          HTTP_POST_DATA
```

(Or)

If user wants to open HTTP client with version 1.1 then set FLAGS to 64 **(HTTP_V_1_1)**.

```
#define FLAGS                          HTTP_V_1_1
```

(Or)

If user wants to open normal HTTP client socket with IPv6 then set FLAGS macro to 1 **(HTTPV6)**.

```
#define FLAGS                          HTTPV6
```

(Or)

If user wants to open HTTP client socket over SSL with IPv6 then set FLAGS macro to 3 **(HHTPV6 | HTTPS_SUPPORT)**

```
#define FLAGS                     (HTTPV6 | HTTPS_SUPPORT)
```

HTTP_PORT refers Port number of the remote HTTP/HTTPS server which is opened in Windows PC2.

```
#define HTTP_PORT                 80
```

HTTP_SERVER_IP_ADDRESS refers IP address of the HTTP/HTTPS server

Note: **HTTP_SERVER_IP_ADDRESS** should be as the below mentioned format as it is a string.

```
#define HTTP_SERVER_IP_ADDRESS    "192.168.10.1"
```

HTTP_URL refers HTTP resource name

```
#define HTTP_URL                  "/index.html"
#define HTTP_HOSTNAME             "192.168.10.1"
```

HTTP_HOSTNAME refers host name

```
      #define HTTP_HOSTNAME                 "192.168.10.1"
```

HTTP extended header

```
      #define HTTP_EXTENDED_HEADER          NULL
```

HTTP/HTTPS user name

```
      #define USERNAME                      "admin"
```

Password for server

```
      #define PASSWORD                      "admin"
```

HTTP/HTTPS post data

```
      #define HTTP_DATA "employee_name=MR.REDDY&employee_id=
RSXYZ123&designation=Engineer&company=REDPINE&location=Hyderab
ad"
```

Max HTTP PUT buffer length

```
      #define MAX_HTTP_CLIENT_PUT_BUFFER_LENGTH 900
```

Application memory length which is required by the driver

```
      #define  GLOBAL_BUFF_LEN              8000
```

Application buffer length

```
      #define  APP_BUFF_LEN                 2000
```

2. Open ***sapis/include/rsi_wlan_config.h*** file and update/modify following macros,

```
 #define  CONCURRENT_MODE              RSI_DISABLE
 #define  RSI_FEATURE_BIT_MAP          FEAT_SECURITY_OPEN
 #define  RSI_TCP_IP_BYPASS            RSI_DISABLE
```

If user wants to connect with HTTP server set RSI_TCP_IP_FEATURE_BIT_MAP as follows,

```
 #define RSI_TCP_IP_FEATURE_BIT_MAP  (TCP_IP_FEAT_DHCPV4_CLIENT
                                    | TCP_IP_FEAT_HTTP_CLIENT)
```

If user wants to connect with HTTPs server set RSI_TCP_IP_FEATURE_BIT_MAP as follows,

```
 #define RSI_TCP_IP_FEATURE_BIT_MAP  (TCP_IP_FEAT_DHCPV4_CLIENT
                   | TCP_IP_FEAT_SSL | TCP_IP_FEAT_HTTP_CLIENT)


 #define  RSI_CUSTOM_FEATURE_BIT_MAP   0
 #define  RSI_BAND                     RSI_BAND_2P4GHZ
```

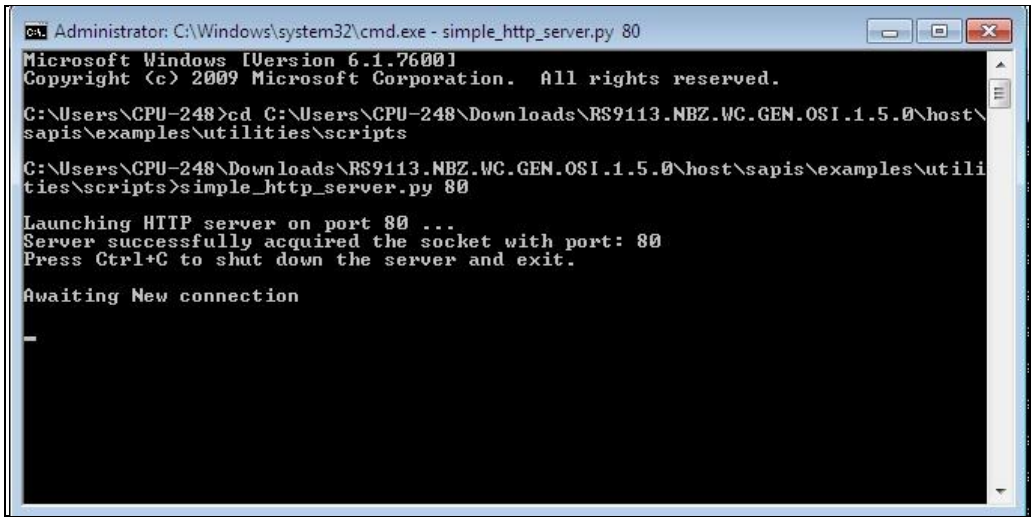
2.3 Executing the Application

1. In Windows PC2, install python and run HTTP server or HTTPs server,

In release package python scripts are provided to open HTTP server and HTTPs server in the following path:

sapis/examples/utilities/scripts

Run ***simple_http_server.py*** by giving port number 80 as argument to open HTTP server.



Note: Release package includes only HTTP server script. If user wants to test HTTPs client, then user has to run HTTPs server which supports HTTPs PUT, GET and POST.

2. **SPI Interface**

If User using SPI interface, Please refer the document ***sapis/platforms/spansion_MB9BF568NBGL/RS9113-WiSeConnect_SAPIS_Spansion_Project_User_guide.pdf*** for executing the ***http_client*** example in CooCox IDE.

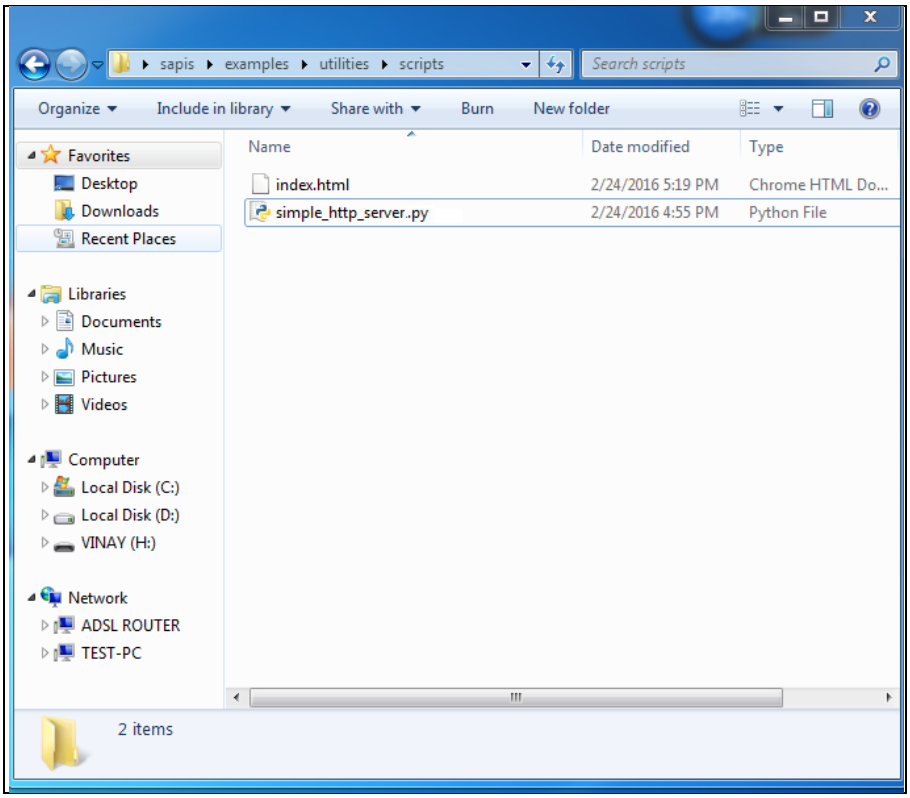
3. **UART/USB-CDC Interface**

If User using UART interface, Please refer the document ***sapis/platforms/windows_uart/RS9113-WiSeConnect_SAPIS_Windows_Project_UserGuide.pdf*** for executing the ***http_client*** example in Dev-C++ IDE

4. After the program gets executed, WiSeConnect device connects to AP and get IP.
5. After successful connection with Access Point, WiSeConenct device request for HTTP PUT to PUT/Create the file on to the server, which is given in index.txt file and wait until put file complete.
6. Remote web server accepts a PUT request and writes the received

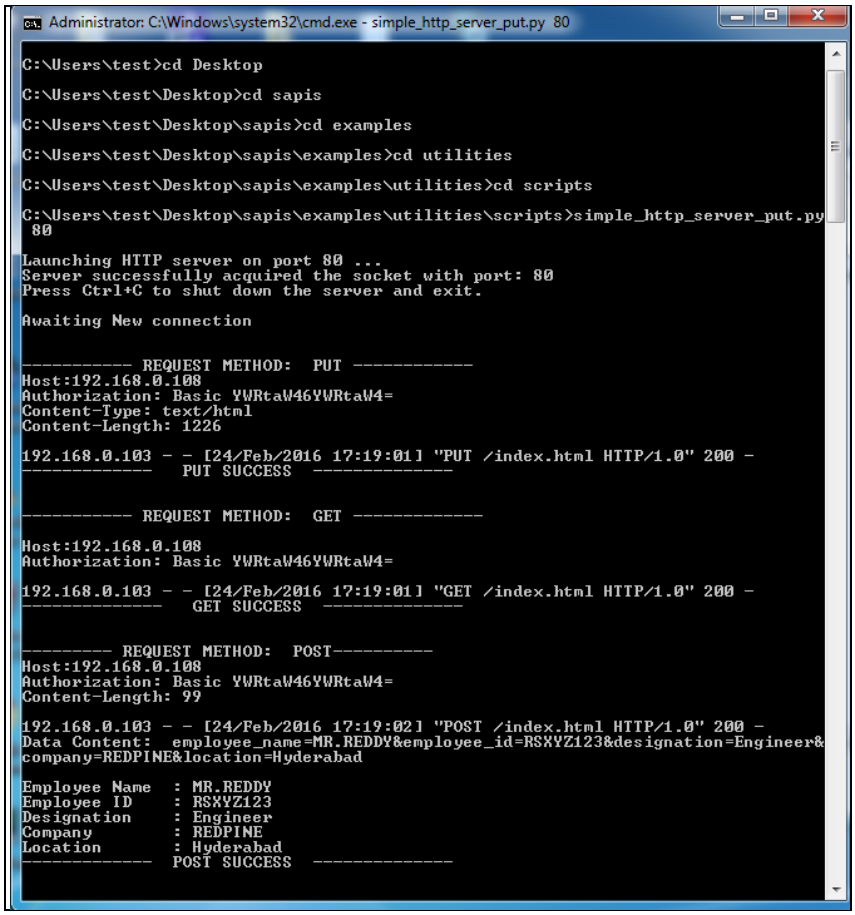
data to a file. User can find the created new file “index.html” on Windows PC2 in the following path,

sapis/examples/utilities/scripts



7. After successful creation of file using HTTP PUT, WiSeConnect device request for the file “index.html” using HTTP GET method and wait until complete response receive from Server.
8. After receiving complete response for the given HTTP GET, WiSeConnecr device post the given data in HTTP_DATA macro to HTTP server using HTTP POST method.

   User can see the log messages at HTTP server. Pease find the below image for success responses for HTTP PUT, HTTP GET and HTTP POST.


Administrator: C:\Windows\system32\cmd.exe - simple_http_server_put.py 80
C:\Users\test>cd Desktop
C:\Users\test\Desktop>cd sapis
C:\Users\test\Desktop\sapis>cd examples
C:\Users\test\Desktop\sapis\examples>cd utilities
C:\Users\test\Desktop\sapis\examples\utilities>cd scripts
C:\Users\test\Desktop\sapis\examples\utilities\scripts>simple_http_server_put.py 80
Launching HTTP server on port 80 ...
Server successfully acquired the socket with port: 80
Press Ctrl+C to shut down the server and exit.
Awaiting New connection
---------- REQUEST METHOD: PUT ----------
Host:192.168.0.108
Authorization: Basic YWRtaW46YWRtaW4=
Content-Type: text/html
Content-Length: 1226
192.168.0.103 - - [24/Feb/2016 17:19:01] "PUT /index.html HTTP/1.0" 200 -
---------- PUT SUCCESS ----------
---------- REQUEST METHOD: GET ----------
Host:192.168.0.108
Authorization: Basic YWRtaW46YWRtaW4=
192.168.0.103 - - [24/Feb/2016 17:19:01] "GET /index.html HTTP/1.0" 200 -
---------- GET SUCCESS ----------
---------- REQUEST METHOD: POST----------
Host:192.168.0.108
Authorization: Basic YWRtaW46YWRtaW4=
Content-Length: 99
192.168.0.103 - - [24/Feb/2016 17:19:02] "POST /index.html HTTP/1.0" 200 -
Data Content: employee_name=MR.REDDY&employee_id=RSXYZ123&designation=Engineer&company=REDPINE&location=Hyderabad
Employee Name : MR.REDDY
Employee ID : RSXYZ123
Designation : Engineer
Company : REDPINE
Location : Hyderabad
---------- POST SUCCESS ----------