

# HTTP/HTTPS Client Post Data 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](mailto:info@redpinesignals.com)

Website: [www.redpinesignals.com](http://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.

Copyright © 2015 Redpine Signals, Inc. All rights reserved.

## Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>4</b>
1.1	Protocol Overview .....	4
1.2	Application Overview .....	4
1.2.1	Overview .....	4
1.2.2	Sequence of Events .....	4
1.3	Application Setup .....	4
1.3.1	SPI based Setup Requirements .....	4
1.3.2	UART/USB-CDC based Setup Requirements .....	5
<b>2</b>	<b>Configuration and Execution of the Application .....</b>	<b>6</b>
2.1	Initializing the Application .....	6
2.1.1	SPI Interface .....	6
2.1.2	UART/USB-CDC Interface .....	6
2.2	Configuring the Application .....	6
2.3	Executing the Application .....	9

## Table of Figures

Figure 1: Setup Diagram .....	5
-------------------------------	---

## 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 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 Post and Get 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 POST and GET.

### 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.

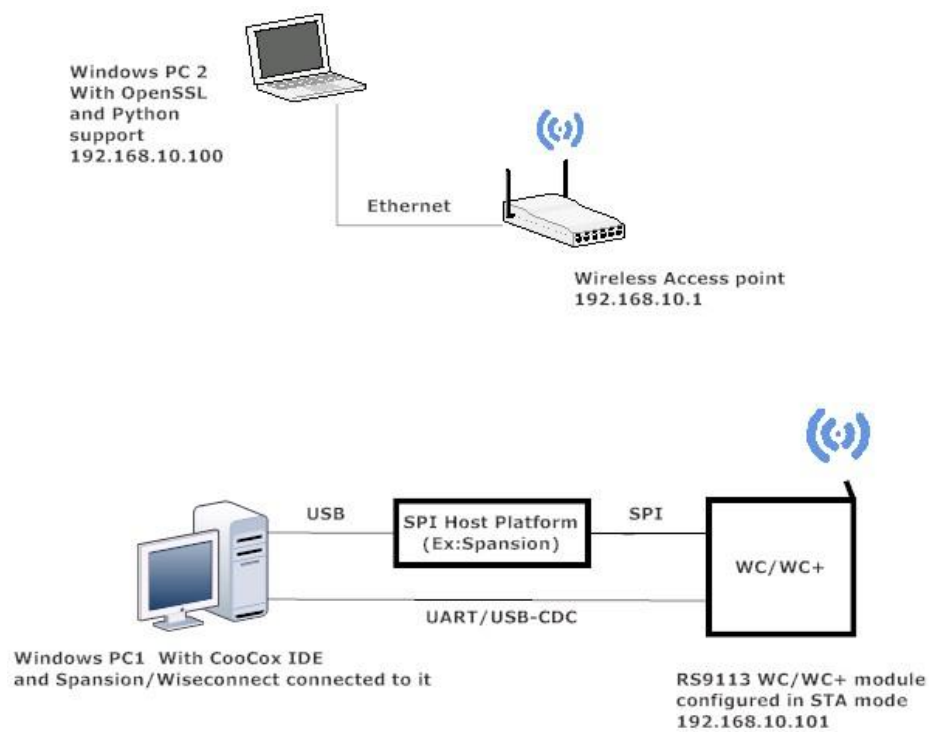
- WiSeConnect device
- WiFi Access point
- Windows PC2 with openssl support and python installed

**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 CoCoX 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 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 (HTTPV6 | HTTPS\_SUPPORT)

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

If user wants to use HTTP client to post large HTTP data set FLAGS macro to 32

(HTTP\_POST\_DATA)

```
#define FLAGS (HTTP_DAT_DATA)
```

If user wants to use HTTP client version 1.1 set FLAGS macro to 64

(HTTP\_V\_1\_1)

```
#define FLAGS (HTTP_V_1_1)
```

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"
```

Max HTTP POST DATA buffer length

```
#define MAX_HTTP_CLIENT_POST_DATA_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
```

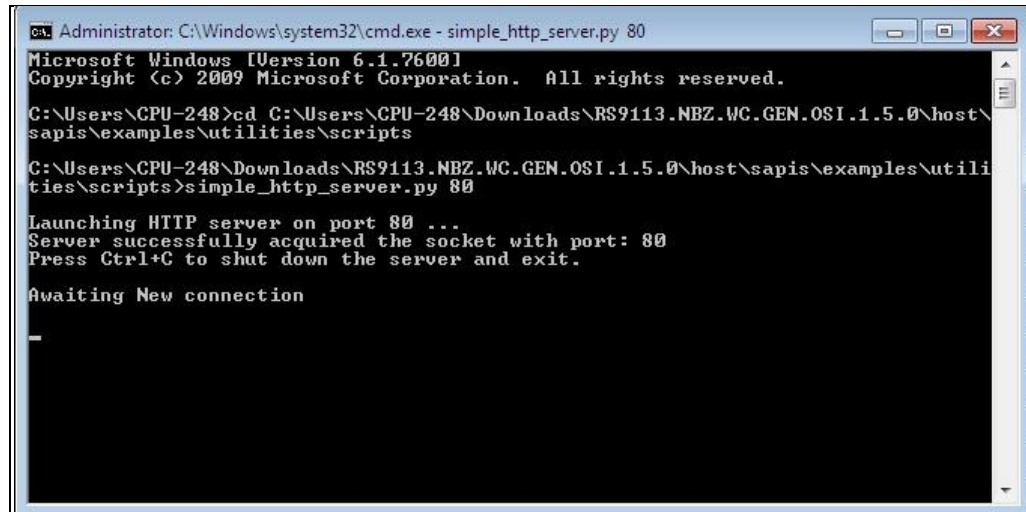
## 2.3 Executing the Application

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\_post\_data.py*** by giving port number 80 as argument to open HTTP server.



```
Administrator: C:\Windows\system32\cmd.exe - simple_http_server.py 80
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\CPU-248>cd C:\Users\CPU-248\Downloads\RS9113.NBZ.WC.GEN.OSI.1.5.0\host\
sapis\examples\utilities\scripts
C:\Users\CPU-248\Downloads\RS9113.NBZ.WC.GEN.OSI.1.5.0\host\sapis\examples\uti
lies\scripts>simple_http_server.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
-
```

**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\_post\_data*** example in CooCox IDE.

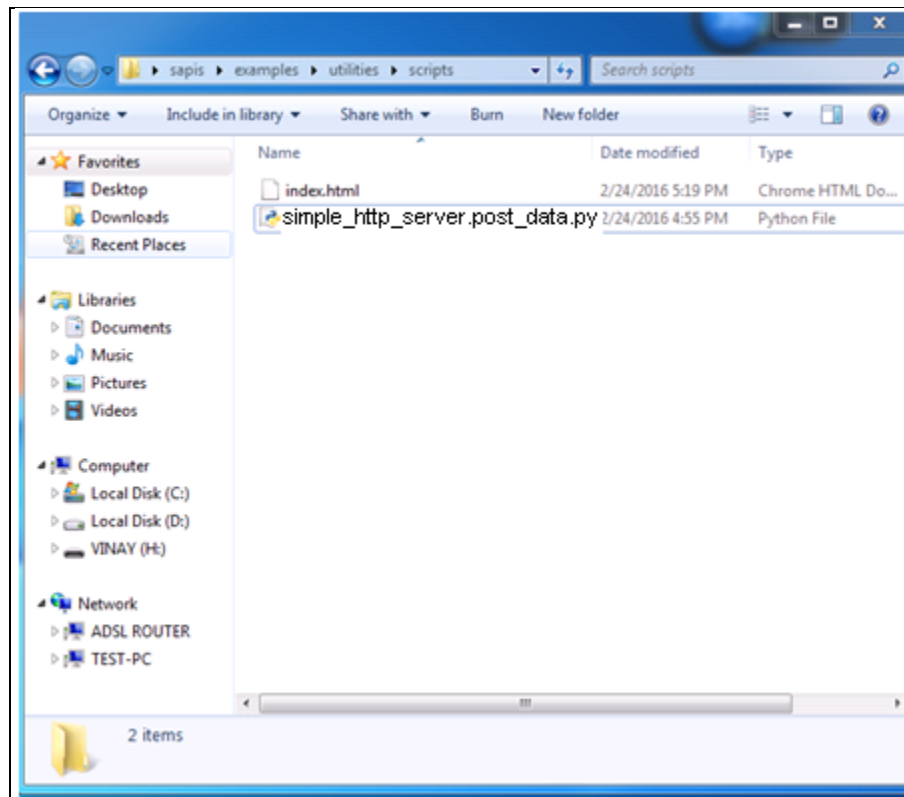
## 3. UART/USB-CDC Interface

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\_post\_data*** example in Dev-C++ IDE

- After the program gets executed, WiSeConnect device connects to AP and get IP.
- After successful connection with Access Point, WiSeConenct device request for HTTP POST to send the usr given file to the server, which is given in index.txt file and wait until post file complete.

Remote web server accepts a POST request and give response. User can find the created new file "index.html" on Windows PC2 in the following path,

***sapis/examples/utilities/scripts***



6. After successful sending of file using HTTP POST, WiSeConnect device request for the file "index.html" using HTTP GET method and wait until complete response receive from Server.