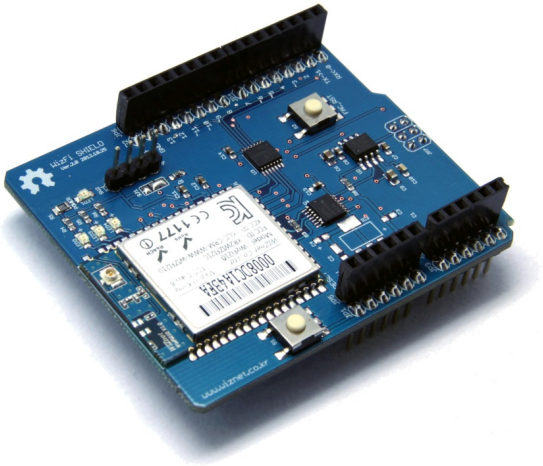
WizFi Shield
User's Manual
Rev1.0



Contents

1. Overview 5
2. Hardware 5
   2.1. Introduction 5
   2.2. Hardware connection 5
   2.2.1. Power Input and GPIO pin for reset 6
   2.2.2. SPI bus and GPIO 6
   2.2.3. Mode setting and UART 6
3. Library 8
   3.1. Library Installation 8
      3.1.1. Install Arduino IDE 8
      3.1.2. Install WizFi library 8
   3.2. WizFi library 8
   3.3. Files of WizFi2x0 library 9
      3.3.1. Description 9
      3.3.2. Class library 10
         3.3.2.1. WizFi2x0Class 11
         3.3.2.2. WizFiClient 17
         3.3.2.3. WizFiServer 20
         3.3.2.4. WizFiUDP 21
         3.3.2.5. TimeoutClass 25
4. Sketch programming guide 28
   4.1. Declaratives 28
      4.1.1. Including header files 28
      4.1.2. Variables 28
      4.1.3. Timer1_ISR callback function 29
   4.2. setup() function 30
   4.3. loop() function 30
5. Examples 32
   5.1. WizFiBasicTest 32
      5.1.1. Introduction 32
      5.1.2. Declarative 33
         5.1.2.1. Including header files 33
         5.1.2.2. Global variable 33
         5.1.2.3. Timer1_ISR callback function 34
      5.1.3. setup() function 34

5.1.4. loop() function 36
5.2. WizFiBasicServerTest 39
5.2.1. Introduction 39
5.2.2. Declarative 40
5.2.2.1. Including header files 40
5.2.2.2. Variables 40
5.2.2.3. Timer1_ISR callback function 41
5.2.3. setup() fucntion 41
5.2.4. loop() function 42
5.3. WizFiLimitedAPTest 44
5.3.1. Introduction 44
5.3.2. Declarative 45
5.3.2.1. Including headers 45
5.3.2.2. Variables 45
5.3.2.3. Timer1_ISR callback function 46
5.3.3. setup() function 46
5.3.4. loop() function 47
5.4. WizFiLimitedAPTest 49
5.4.1. Introduction 49
5.4.2. Declarative 50
5.4.2.1. Including headers 50
5.4.2.2. Variables 50
5.4.2.3. Timer1_ISR callback function 51
5.4.3. setup() function 51
5.4.4. loop() function 52
5.5. WizFiUDPClientTest 54
5.5.1. Introduction 54
..... 54
5.5.2. Declarative 55
5.5.2.1. Including headers 55
5.5.2.2. Variables 55
5.5.2.3. Timer1_ISR Callback function 55
5.5.3. setup() function 56
Set the peer's IP address, port number, and user's port number to create myUDP 58
5.5.4. loop() function 58
5.6. WizFiUDPServerTest 59

List of Figures

FIGURE 1 USED PINS TO CONNECT ARDUINO UNO 5
FIGURE 2 SCHEMATIC FOR MODE SETTING AND UART 7
FIGURE 3 WIZFI2X0 LIBRARY 8
FIGURE 4 FILES OF WIZFI2X0 LIBRARY 9
FIGURE 5 CLASS OF WIZFI2X0 LIBRARY 10
FIGURE 9 NETWORK DIAGRAM OF WIZFIBASICTEST 32
FIGURE 10 RESULT OF WIZFIBASICTEST EXAMPLE 33
FIGURE 22 EXAMPLE OF CHECKING THE SOCKET CONNECTION AND DATA SENDING AND RECEIVING 38
FIGURE 26 NETWORK DIAGRAM OF WIZFIBASICSERVERTEST 39
FIGURE 27 NORMAL OPERATION OF WIZFIBASICSERVERTEST 40
FIGURE 35 NETWORK DIAGRAM WIZFILIMITEDAPTEST 44
FIGURE 36 SERIAL MONITOR OUTPUT OF WIZFILIMITEDAPTEST 45
FIGURE 35 NETWORK DIAGRAM OF WIZFILIMITEDAPTEST 49
FIGURE 36 SERIAL MONITOR OUTPUT OF WIZFILIMITEDAPTEST 50
FIGURE 46 SERIAL MONITOR OUTPUT OF WIZFIUDPCLIENTTEST 54

1. Overview

WizFi Shield is WIZnet's Open Source Hardware based Wi-Fi shield that uses WIZnet's Wi-Fi module, WizFi210, to implement wireless communication in Arduino development environments. The Wi-Fi module used in this shield, WizFi210, features low power and stable wireless capability so that any developer can utilize the shield in Arduino development environments.

Features

- **Hardware compatible to an Arduino board.**
- **Support SPI Interface**
- **Multiple TCP/IP communication sockets are supported (maximum 16 sockets).**
- **Provide Arduino Library (similar to Arduino Ethernet Shield library).**

2. Hardware

2.1. Introduction

WizFi Shield connects with the Arduino board by SPI bus. Besides SPI bus, 3 GPIO pins are required.

2.2. Hardware connection

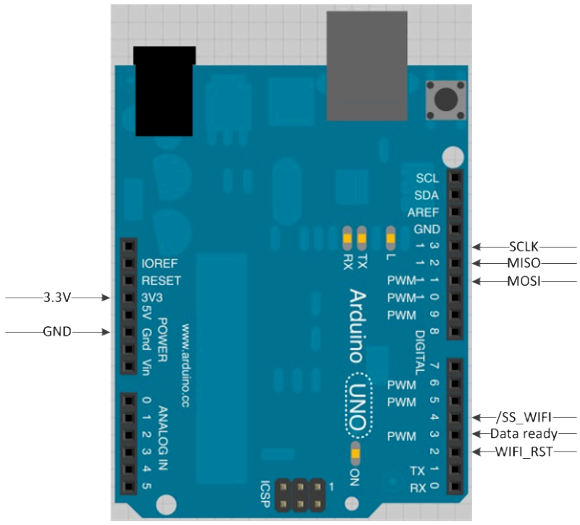


Figure 1 Pins to connect Arduino UNO

2.2.1. Power Input and GPIO pin for reset

WizFi Shield operates 3.3V power from Aruino board and can be reset by setting the digital pin #2 as 'low'.

```
#define WizFi2x0_RST 2
pinMode(WizFi2x0_RST, OUTPUT);
```

2.2.2. SPI bus and GPIO

WizFi Shield communicates with the Arduino board through the SPI bus.

Use digital pin #4 for 'SPI slave enable (chip select) signal' and use digital pin #3 for 'the data ready signal' from WizFi210 module. These signals are listed as /SS_Wifi and GPIO19 on the circuit diagram.

The 'data ready signal' indicates whether the WizFi210 module has data to send to Arduino board or not. If there is data to send, the 'data ready signal' is enabled (high). Since sll these signals are 3.3V level, voltage level transceiver was used for the 5V I/O interface.

```
#define WizFi2x0_DataReady 3
#define WizFi2x0_CS        4
 pinMode(WizFi2x0_DataReady, INPUT);
 pinMode(WizFi2x0_CS, OUTPUT);
```

2.2.3. Mode setting and UART

The default mode of the WizFi210 module is 'run mode.' If the user needs to update the firmware of the module, the user must switch to 'program mode.' In order to switch to 'program mode,' short the number 1 and 2 of SJ1 and cut the pattern of number 2 and 3.

WizFi210's UART pin is connected to the WizFi Shield's 4 pin header SV2. The UART pin is used to update the module's firmware under 'program mode.' [*Only available to H/W version 2.0]

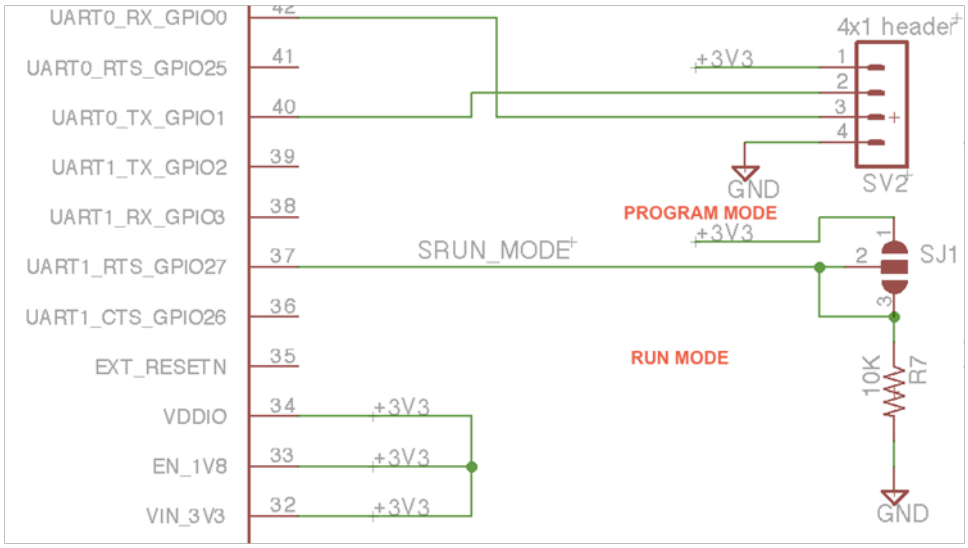


Figure 2 Schematic for mode setting and UART

3. Library

3.1. Library Installation

Library must be installed in order to implement wireless communication with WizFi Shield. Development environment settings and library settings are covered below.

3.1.1. Install Arduino IDE

If the user has not installed Arduino IDE yet, click the following link to download and install. http://arduino.cc/en/Main/Software (Users who already installed Arduino IDE may skip).

3.1.2. Install WizFi library

Copy/paste the WizFi2x0.zip and TimeOne.zip files in the Arduino IDE library directory. Extract both zip files and then both WizFi2x0 directory and TimerOne directory will be created as shown in below image.

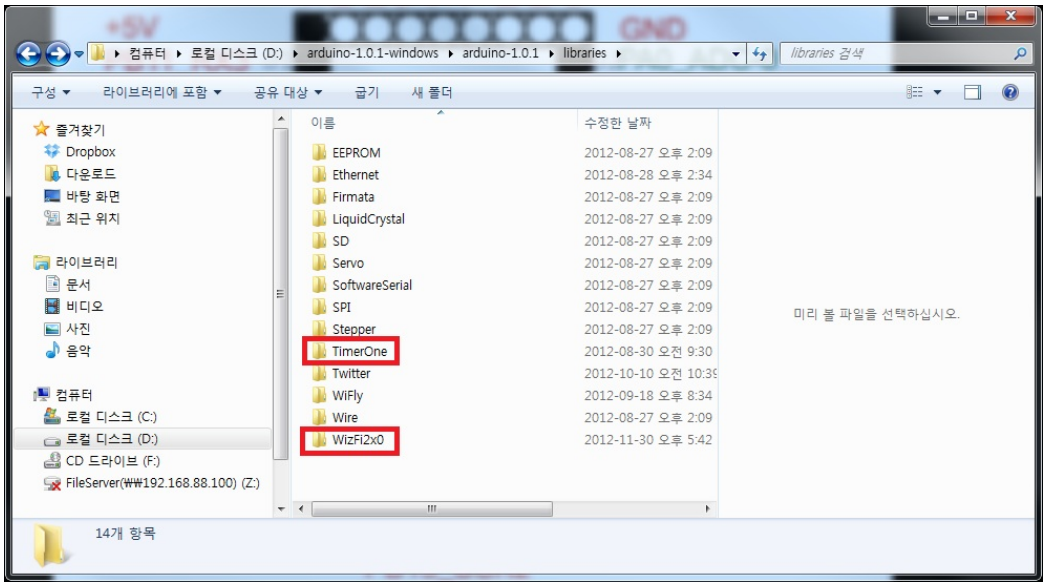


Figure 3 WizFi2x0 library

3.2. WizFi library

The WizFi Shield library is composed of the WizFi2x0 and TimerOne library.

The WizFi2x0 directory includes all functions for controlling and processing responses of the WizFi210 module. The TimerOne directory includes the function used for Timer1

interrupt, which is provided from the Arduino board's MCU. The functions in the WizFi2x0 directory use the Timer1 interrupt to check the time response. Both directories must be installed.

3.3. Files of WizFi2x0 library

3.3.1. Description

File	Description
Keywords.txt	Identical as Arduino library standard. Functions, commands, and keywords for approaching the sketch is specified.
WizFi2x0.cpp	Implementation of functions for controlling the WizFi2x0 module.
WizFi2x0.h	Definition of functions for controlling the WizFi2x0 module.
WizFiClient.cpp	Implementation of functions for using the TCP Client socket with the WizFi2x0 module.
WizFiClient.h	Definition of functions for using the TCP Client socket with the WizFi2x0 module.
WizFiServer.cpp	Implementation of functions for using the TCP Server socket with the WizFi2x0 module.
WizFiServer.h	Definition of functions for using the TCP server socket with the WizFi2x0 module.
WizFiUDP.cpp	Implementation of functions for using the UDP socket with the WizFi2x0 module.
WizFiUDP.h	Definition of functions for using the UDP socket with the WizFi2x0 module.

Figure 4 Files of WizFi2x0 library

3.3.2. Class library

Class	Description	etc
WizFi2x0Class (in WizFi2x0.cpp, WizFi2x0.h)	- Resets WizFi2x0. - Connects and disconnects with AP. - Controls the WizFi2x0 module.	
TimeoutClass (in WizFi2x0.cpp, WizFi2x0.h)	- Sets the response wait time of WizFi2x0 module. - Processes an event over a period of time.	
WizFiClient (in WizFiClient.cpp, WizFiClient.h)	- Creates TCP client socket. - Connects and disconnects with server. - Data transmission.	- WizFi2x0Class is used to control the WizFi210 module while processing function.
WizFiServer (in WizFiServer.cpp, WizFiServer.h)	- Creates TCP server(Listen) socket. **Note) There is no data transmission function in the WizFiServer class. Data transmission is possible through the WizFiClient class after connecting to the server. When using the WizFiServer class, the WizFiClient class is used depending on the number of sockets.**	- WizFi2x0Class is uesd to control the WizFi210 module while processing function.
WizFiUDP (in WizFiUDP.cpp, WizFiUDP.h)	- Creates UDP socket - Data transmission Sets IP address and port number	- WizFi2x0Class is used to control the WizFi210 module while processing function.

Figure 5 Class of WizFi2x0 library

Member function of WizFi2x0 library

3.3.2.1. WizFi2x0Class

Function Name	**WizFi2x0Class(void)**
Return value	**None**
Parameter	**None**
Description	Constructor of WizFi2x0Class.

Function Name	**void begin(void)**
Return value	**None**
Parameter	**None**
Description	- Sets the pins connected to WizFi2x0 module. - Sets the SPI speed. - Initiakize and resets WizFi2x0. - Wait 4 seconds after setting HW reset pin to high. - Initilize Reset ReplyCheckTimer. - Reset current state of all statediagram.

Function Name	**uint8_t associate(void)**
Return value	**1 : success, 0 : fail**
Parameter	**None**
Description	- Attempt connection to AP. - Set the SSID and other information to WizFi2x0Class. This function is used when the connection to AP is attempted while information like SSID of the AP is already set in WizFi2x0Class during its original connection with AP.

Function Name	**uint8_t associate(ssid, passphrase, EncryptType, isDHCP)**
Return value	**1 : success, 0 : fail**
Parameter	**ssid : SSID value of AP, string type** **passphrase: Security Key value (HEX format)** **EncryptType: Set Enumeration value according to Encryption type** **isDHCP: boolean type, set DHCP mode if needed**

<table>
<tr><td>Description</td><td>- Attempt connection with AP.
- This function is used when attempting to connect to AP for the first time or connecting to AP when the parameter value has changed.
- EncryptType
<table><tr><th>TYPE</th><th>Meaning</th></tr><tr><td>NO_SECURITY</td><td>No security</td></tr><tr><td>WEP_SECURITY</td><td>WEP</td></tr><tr><td>WPA_SECURITY</td><td>WPA</td></tr><tr><td>WPA2PSK_SECURITY</td><td>WPA2-PSK</td></tr></table>Note) Select one of the 4 options above from the EncryptType.</td></tr>
</table>

Function Name	uint8_t disassociate(void)
Return value	1 : success, 0: fail
Parameter	None
Description	- Disconnect with AP. Note) When there are many wireless devices connected, it is recommended to disconnect with AP after all data transmission is completed.

Function Name	boolean IsAssociated(void)
Return value	true: associated, false: not associated
Parameter	None
Description	- This function checks the connection status with AP.

Function Name	void SetOperatingMode[1](mode)
Return value	None

[1] Additional function for Limited AP feature

The WizFi210 module can operate like AP. However, due to the limit of the program memory size, only the essential features of AP are provided and this is the Limited AP mode of the WizFi210 module. This mode is useful when wireless connection between a smart phone without a separate AP is needed.

Parameter	**Mode: set the enumeration value of OPMODE Type**
Description	- This function sets the operation mode of the WizFi2x0 module. - The following are the different types of operation modes. **Note)** The default mode of the WizFi2x0 module is INFRA_MODE. Use this function to switch the operation mode before using the associate(...) function.

Operation mode	Meaning
INFRA_MODE	Infrastructure mode
ADHOC_MODE	Adhoc mode
LIMITEDAP_MODE	AP mode

Function Name	**void SendSync(void)**
Return value	**None**
Parameter	**None**
Description	- This function sends a sync packet to check the status of SPI communication between the Arduino board and WizFi Shield. - The value of the sync packet is 0xF5.

Function Name	**uint8_t CheckSyncReply(void)**
Return value	**None**
Parameter	**None**
Description	- This function checks if a reply for the sync packet has been sent from the WizFi Shield. - The reply will be one of the following; 0xF5, 0xFF and 0x00.

Function Name	**uint8_t SetTxPower[2](uint8_t power_level)**
Return value	**1: success, 0: fail**
Parameter	**Power_level: Transmitting power level**

[2] Added for changing the RF transmitting power of WizFi2x0 module

Description	- This function changes the RF transmission input of the WizFi2x0 module. - Set the value between 0~7 for WizFi210 and 2~15 for WizFi220. [ERROR] will occur when other values are set.

Function Name	uint8_t GetTxPower[3](void)
Return value	Power_level: level of the transmission input currently set.
Parameter	None
Description	- This function checks the level of the RF transmission currently set on the WizFi2x0 module.

Function Name	uint8_t wifiScan (void)
Return value	1: success, 0: fail
Parameter	None
Description	- This function scans all AP around the WizFi2x0 module. - Only the SSID value of the AP is printed but users can modify the library to print information other than SSID value.

Function Name	uint8_t wifiScan (uint8_t channel)
Return value	1: success, 0: fail
Parameter	channel: parameter to select a particular frequency channel.
Description	- This function scans all AP around the WizFi2x0 module and prints the SSID of AP that is open through the selected channel. If user sets '0' as the channel, the function will not search according to the channel. - In the current library, only the SSID value of the AP is printed but users can modify the library to print

	information other than SSID value.

Function Name	uint8_t wifiScan (uint8_t channel, uint8_t RSSI)
Return value	1: success, 0: fail
Parameter	channel: parameter to select a particular frequency channel. RSSI: parameter for setting the power of the signal.
Description	- This function scans all AP around the WizFi2x0 module and prints the SSID of AP that is open through the selected channel and have a stronger signal than the assigned AP. If user sets '0' as the channel, the function will not search according to the channel. - In the current library, only the SSID value of the AP is printed but users can modify the library to print information other than SSID value.

Function Name	void SetSrcIPAddr(byte * buf)
Return value	None
Parameter	buf: Source IP address를 string 형태로 가지고 있는 배열 sequence of source IP address in string form
Description	- This function is used to set the fixed IP address of the WizFi2x0 module; the function sets the buf value as the source IP.

Function Name	void GetSrcIPAddr(byte * buf)
Return value	None
Parameter	buf: Source IP address를 string 형태로 반환 받을 배열 sequence of source IP address to be returned in string form.
Description	- This function returns the source IP of the WizFi2x0 module in the buf variable.

Function Name	void SetSrcSubnet(byte * buf)

Return value	None
Parameter	buf: Array of source subnet mask in string form
Description	- This function is used to set the fixed IP address of the WizFi2x0 module; the function sets the buf value as the source subnet mask

Function Name	void GetSrcSubnet(byte * buf)
Return value	None
Parameter	buf: Array of source subnet mask in string form
Description	- This function returns the source subnet mask in the buf variable.

Function Name	void SetSrcGateway(byte * buf)
Return value	None
Parameter	buf: Array of source gateway address in string form.
Description	- This function is used to set the fixed IP address of the WizFi2x0 module; the function sets the buf value as the source gateway address.

Function Name	void GetSrcGateway(byte * buf)
Return value	None
Parameter	buf: Array of source gateway address to be returned in string form
Description	- This function returns the source gateway address of the WizFi2x0 module in the buf variable.

Function Name	void SetSrcPortnum(uint16_t portnum)
Return value	None
Parameter	portnum: port number used in the server socket.
Description	- This function sets the portnum value at the SrcPortNum member variable of WizFi2x0 Class object. - This function is used as a reference for the WizFiServer object in the member function begin().

Function Name	uint16_t GetSrcPortnum(void)
Return value	None
Parameter	None
Description	- This function returns the value currently set in the WizFi2c0 module SicPortNum variable.

Function Name	void RcvPacket(void)
Return value	None
Parameter	None
Description	- This function is used to read all the received data from the WizFi2x0 module SPI port and the notification messages from the module. **Note) This function should be used simultaneously in order to not lose the communication data or notification messages from the WizFi2x0 module SPI port.**

3.3.2.2. WizFiClient

Function Name	WizFiClient(void)
Return value	None
Parameter	None
Description	- Constructor of WizFiClient.

Function Name	WizFiClient(uint8_t ip, uint16_t port)
Return value	None
Parameter	- ip : server's IP address. ex) ip[4] = {192,168,0,100} - port : server's Listen port number
Description	- Constructor of WizFiClient when the server's IP is known.

Function Name	WizFiClient(const char* domain, uint16_t port)
Return value	None
Parameter	- domain : server's domain name.

	ex)www.google.com - port : Server's listen port number
Description	- Constructor of WizFiClient when the server's domain address is known.

Function Name	uint8_t connect()
Return value	1: success, 0: fail
Parameter	None
Description	- Attempts to connect with server - The server's IP address(or domain address), and port number when created an object are used.

Function Name	uint8_t disconnect()
Return value	1: success, 0: fail
Parameter	None
Description	- Disconnects with the server.

Function Name	boolean available()
Return value	true: connected, false: disconnected
Parameter	None
Description	- This function checks the connection status of the socket. - This function checks whether the notification messages are relevant to the internal events (socket connection, disconnection, and error) of the WizFi2x0 module; then modifies the socket status and socket connection accordingly.

Function Name	void write(byte value)
Return value	None
Parameter	value: 1 byte data for socket transmission
Description	The 1 byte data(value) is transmitted to the server. - To use this function, the socket must be connected.

Function Name	**void write(byte *buf)**
Return value	**None**
Parameter	**buf: byte stream data to be transmitted through the socket**

Description	- This function is used when over 2 bytes of data is transmitted to the server. The next byte of the last byte of the data must be NULL character.
	- **Data can be lost if this function is used to send data including NULL character. In that case, use write(byte *buf, size_t size) function.**
	- ex) when transmitting "abcde,"
	Byte buf[10];
	memset(buf, 0, 6);
	memcpy(buf, "abcde", 5);
	myClient.write(buf); //myClient is the object of WizFiClient type

Function Name	**void write(byte *buf, size_t size)**
Return value	**None**
Parameter	**buf: byte stream data to be transmitted through the socket** **size: actual data size to be transmitted**
Description	- This function is used when over 2 bytes of data are transmitted to the server and the data length is known or partial data is transmitted. - **This function must be used when there is a possibility of NULL characters to be included.**

Function Name	**uint8_t read()**
Return value	**Received data**
Parameter	**None**
Description	- Reads the data received from the server. - Data is sent in return value.

Function Name	uint8_t read(byte *buf)
Return value	Number of received data byte
Parameter	buf: The received data is filled in the buf and returned.
Description	- Reads the received data from server in Array.

Function Name	uint8_t read(byte *buf, size_t size)
Return value	Number of received data byte
Parameter	buf: The received data is filled in the buf and returned. size: Set the number of byte of data to be received.
Description	- Reads the received data from server in array. - Data is filled as much or lesser than the byte size of the buf and returned.

Function Name	void GetCIDstr(byte *strCID)
Return value	None
Parameter	strCID: Sequence pointer of CID value in character stinrg.
Description	- This function enables the CID value to be returned as Str. - If connection is successful, WizFiClient allocates a certain CID value (0~15).72

Function Name	uint8_t GetCID(void)
Return value	CID value is returned as an integer
Parameter	None
Description	- This function enables the CID value to be returned as an integer..

3.3.2.3. WizFiServer

Function Name	WizFiServer(Serverport)
Return value	None

Parameter	None
Description	- Section where WizFiServer object is created.

Function Name	uint8_t begin(void)
Return value	1: success, 0: fail
Parameter	None
Description	- This function enables the server socket to listen as the assigned port value.

Function Name	void GetCIDstr(byte *strCID)
Return value	None
Parameter	strCID: Sequence pointer of CID value in character stinrg..
Description	- This function enables the CID value to be returned in character string.

Function Name	uint8_t GetCID(void)
Return value	None
Parameter	None
Description	- This function enables the CID value to be returned as an integer.

3.3.2.4. WizFiUDP

Function Name	WizFiUDP(void)
Return value	None
Parameter	None
Description	- Constructor of WizFiUDP class.

Function Name	WizFiUDP (uint8_t ip, uint16_t port, uint16_6 src_port)
Return value	None
Parameter	- ip : peer IP address where the data is sent to - port : peer port number where the data is sent to

	- **src_port: UDP socket port number**
Description	- **Constructor of WizFiUDP when IP address is known.** - **IP and port parameter does not have to be set, but src_port must be set.**

Function Name	**WizFiUDP (const char* domain, uint16_t port, uint16_t src_port)**
Return value	**None**
Parameter	- **domain : peer's domain name** - **port : peer's port number** - **src_port: UDP socket port number**
Description	- **Constructor of WizFiUDP when peer's domain name is known.** - **IP and port parameter does not have to be set, but src_port must be set.**

Function Name	**uint8_t open(void)**
Return value	**1: success, 0: fail**
Parameter	**None**
Description	- **This function allows the WizFi2x0 module to create a UDP socket based on the information from WizFiUDP constructor.** - **'1' is returned if the UDP socket is successfully created, and '0' is returned if not.**

Function Name	**uint8_t close(void)**
Return value	**1: success, 0: fail**
Parameter	**None**
Description	- **This function closes the created UDP socket.** - **'1' is returned if the UDP socket is closed, and '0' is returned if the socket is not closed or other error has occurred.**

Function Name	**void write(byte value)**
Return value	**None**
Parameter	**value: 1 byte data to be sent through the socket**
Description	- Send 1 byte date (value) to peer. - Socket must be created in order to use this function.

Function Name	**void write(byte *buf)**
Return value	**None**
Parameter	**buf: byte stream data to be sent through socket**

Description	- This function is used when sending data over two bytes; the byte after the last byte of data must be NULL character.
	- **The following function must be used to end NULL character in order to prevent data loss; write(byte *buf, size_t size)**
	- ex) when sending "abcde"
	Byte buf[10];
	memset(buf, 0, 6);
	memcpy(buf, "abcde", 5);
	myUDP.write(buf); //myUDP는 WizFiUDP 타입의 객체

Function Name	**void write(byte *buf, size_t size)**
Return value	**None**
Parameter	**buf: byte stream data to be sent through the socket** **size: size of data for byte stream**
Description	- This function is used when sending data over two bytes and the data length is known or partial data is to be sent. - **This function must be used when NULL character is included in the data.**

Function Name	**uint8_t read()**
Return value	**Received data**
Parameter	**None**

Description	- Reads the received data. - Sends the data in return value.

Function Name	uint8_t read(byte *buf)
Return value	Number of received data byte
Parameter	buf: Returns the buf filled with received data
Description	- Reads the received data in sequence.

Function Name	uint8_t read(byte *buf, size_t size)
Return value	Number of received data byte
Parameter	buf: Returns the buf filled with received data size: Sets the number of received data byte.
Description	- Reads the received data in sequence. - Data is filled as much or lesser than the byte size of the buf and returned.

Function Name	void GetCIDstr(byte *strCID)
Return value	None
Parameter	strCID: Sequence pointer of CID value in character stinrg.
Description	- This function returns the CID value in Str. - After successfully creating socket, WizFiUDP can be assigned an additional CID value (0~15).

Function Name	uint8_t GetCID(void)
Return value	Returns CID value in integer.
Parameter	None
Description	- This function returns the CID value in integer.

Function Name	void GetCurrentDestInfo(byte *ipaddr, uint16_t *portnum)
Return value	None
Parameter	ipaddr: String character of peer's IP from the

	WizFiUDP constructor. **portnum: Address for the WizFiUDP constructor to save peer's port number.**
Description	- **This function reads the current peer's IP address and port number from WizFiUDP constructor.** - **Peer's IP address and port number are automatically updated when the UDP packet is received; it can also be modified by using the SetCurrentDestInfo(...) function.**

Function Name	**void SetCurrentDestInfo(byte *ipaddr, uint16_t portnum)**
Return value	**None**
Parameter	**ipaddr: String character of IP address for renewing the peer's IP.** **portnum: port number for renewing the peer's port number.**
Description	- **This function modifies the peer's IP address and port number within the WizFiUDP constructor.**

3.3.2.5. TimeoutClass

Function Name	**void init(void)**
Return value	**None**
Parameter	**None**
Description	- **Resets the variable to check timeout.**

Function Name	**void TimerStart(void)**
Return value	**None**
Parameter	**None**
Description	- **Starts timeout check.** - **TimerValue, the variable of TimeoutClass, is used as currently set.**

Function Name	**void TimerStart(uint16_t timevalue)**

Return value	None
Parameter	None
Description	- Starts timeout check. - "timevalue" is used as internal variable "TimerValue" - The resolution of "timevalue" is 1ms.

Function Name	void TimerStop(void)
Return value	None
Parameter	None
Description	- Stops the timeout check.

Function Name	boolean GetIsTimeout(void)
Return value	None
Parameter	None
Description	- Checks if IsTimeout flag, the internal variable of TimoutClass, is set. - True is returned once timeout occurs.

Function Name	void CheckIsTimeout()
Return value	None
Parameter	None
Description	- Checks if the current TimerCount and TimerValue are the same, and then increases TimerCount by 1. - When TimerCount and TimerValue become the same, modify the IsTimeout to 'true' and call TimerStop(). **This function must be used with Timer1 callback function. (Timer1 is set to have Timer interrupt every 1ms.)**

Function Name	void SetIsTimeout(boolean flag);
Return value	None
Parameter	flag: true or false
Description	- Modify the IsTimeout value to flag value.

4. Sketch programming guide

The sketch program of the WizFi Shield is based on the Arduino programming structure; composed of the declaratives, setup() function, and loop() function. This section will go over each steps of the declaratives, setup() function, and loop() function.

4.1. Declaratives

The declaratives is composed of include section, variable declaring section, and Timer1 Callback function section.

4.1.1. Including header files

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SPI.h>
#include <WizFi2x0.h>
#include <WizFiClient.h>
#include <TimerOne.h>
```

The 'include section' includes the library header files for using the WizFi shield.

The header files in red font above are the most necessary files.

SPI.h includes the functions for SPI communication between the Arduino board and WizFi shield.

WizFi2x0.h includes the functions for controlling the WizFi2x0 module.

WizFiClient.h includes the functions for using the client socket; the WizFiServer.h header file must be included when using the server socket.

4.1.2. Variables

```
#define SSID                "AP_SSID"       // SSID of your AP
#define Key                 "1234567890"    // Key or Passphrase
#define ServerName "google.com" // Server Domain Name

unsigned char SIP[4] = {192, 168, 123, 119};
unsigned int ServerPort = 5000;

WizFi2x0Class myWizFi;
WizFiClient myClient;
```

```
WizFiServer myServer;
TimeoutClass ConnectInterval;
```

'SSID' is defined as the SSID value of AP which the WizFi210 module is connecting to. Check the SSID of the AP and modify the value inside the quotation marks.
'Key' is the security key for connecting to the AP with security settings. The value inside the quotation marks must be modified according to the AP.
'ServerName' is the value for connecting with only the server's domain name.
(If the ServerName is not NULL attempt connection with the Domain name only. If the IP address of the server is known, it is recommended to delete the ServerName declarative; this can reduce unnecessary steps like DNS Query).

SIP is the variable used when the server's IP is known. ServerPort is the port number waiting for the server's connection. MyWizFi is a WizFi2x0Class type class and is used for controlling the Wizfi210 module.

MyClient is a WizFiClient type class for processing the Client socket. MyServer is a WizFiServer type class for processing the Server socket.
WizFiClient class has functions for connection attempt, disconnection, and data transmission whereas WizFiServer class only has one function for connection standby (Listen). The WizFi210 module uses the WizFiServer class when it acts a server socket waiting for connection with another device; and WizFiClient class is used for data transmission or disconnection after the connection. Please refer to "Example" from "WizFiBasicServerTest" for more detailed examples.

ConnectionInterval class is a TimeoutClass type for checking time during sketch program. This class does not need to be called if not used. **(TimerOne.h must be included in the include section even if TimeoutClass is not used because it's used in WizFi2x0Class.)**

4.1.3. Timer1_ISR callback function

```
void Timer1_ISR()
{
```

```
        uint8_t i;
        myWizFi.ReplyCheckTimer.CheckIsTimeout();
    }
```

Timer1_ISR callback function is an interrupt service routine where the function is called upon occurrence of Timer1 interrupt.

The 'myWizFi.ReplyCheckTimer.CheckIsTimeout();' part from the Timer1_ISR() function is necessary for MyWizFi class. MyWizFi class processes timeout if there is no response after a command is given to the WizFi210 module.

The 'myWizFi.ReplyCheckTimer.CheckIsTimeout();' part checks the time for how long it has been with no response.

4.2. setup() function

The setup() function is used to process the one-time tasks. The following tasks are appropriate to be done in setup() function.

- Serial port reset for debug message output
- Client socket class reset
- Server socket class reset
- myWizFi class reset
- Timer1 set
- Check Sync of WizFi Shield and SPI communication
- AP association

  (this task can be done in loop() function if needed.)

4.3. loop() function

Loop() function calls the codes in the function infinitely. Therefore, the following tasks in the loop() function are repeated infinitely.

- Call myWizFi.RcvPacket()
- Task of connecting the client socket to server (call myClient.connect())
- Task of checking the connection status with server (call myClient.available())
- Task of reading the data received from server (call myClient.read())
- Task of writing data to be sent to server (call myClient.write())
- Task of disconnecting the client socket with server (call myClient.disconnect())

- **Other tasks**
  - **show the received data on the debug window using the serial port**
  - **analyze the received data and take action accordingly**
  - **etc**

.

5. Examples

5.1. WizFiBasicTest

5.1.1. Introduction

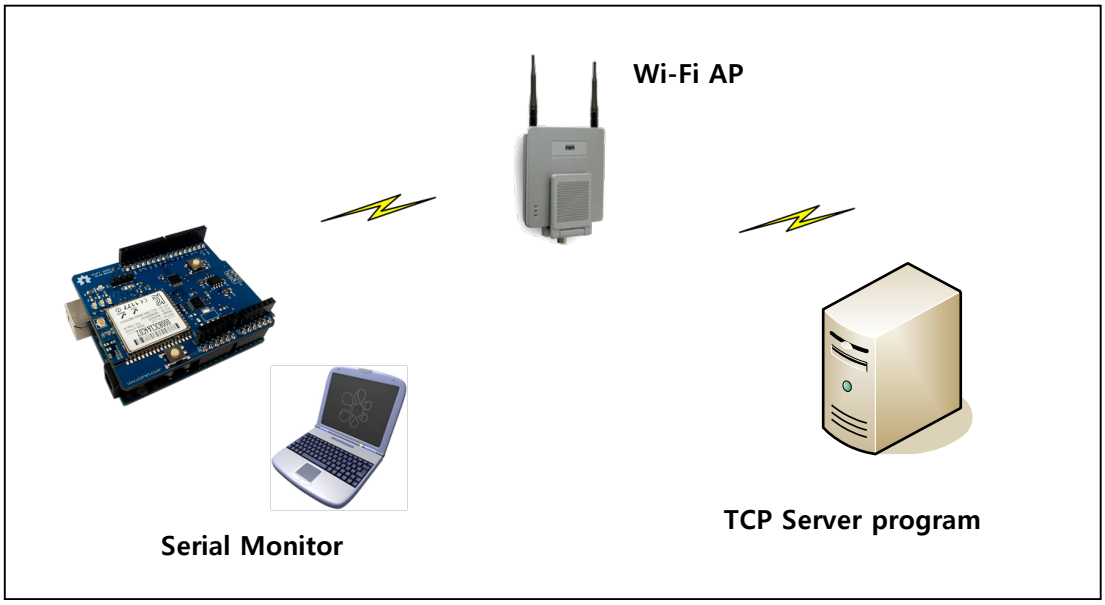


Figure 6 Network diagram of WizFiBasicTest

WizFiBasicTest sketch is processed as the following steps below.

① WizFiBasicTest creates a WizFiClient socket and connects with the assigned AP; and then waits for the user's command.

② WizFiClient socket connects with the assigned server when the user's 'connect' command ('c' or 'C') comes in through the serial monitor.

③ After connecting with the server, check if there are any received data from the server.

④ Display the received data on the serial monitor and resend it to the server.

⑤ WizFiClient socket disconnects with the assigned server when the user's 'disconnect' command ('d' or 'D') comes in through the serial monitor.

Below is the image of serial monitor if the example has been processed successfully and is in the status of waiting for the user's command.

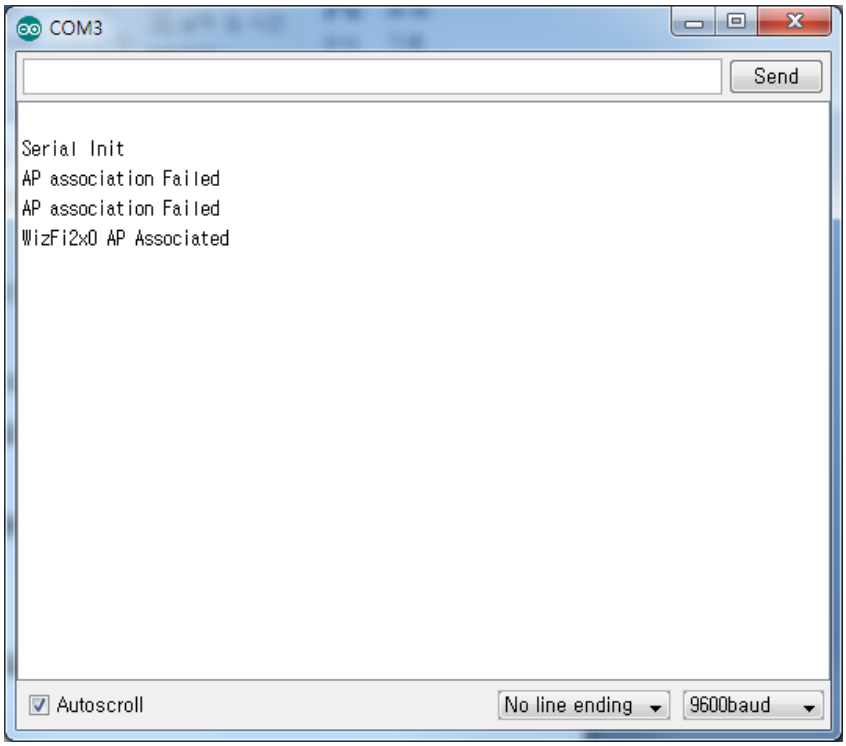


Figure 7 Result of WizFiBasicTest example

5.1.2. Declarative

5.1.2.1. Including header files

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SPI.h>
#include <WizFi2x0.h>
#include <WizFiClient.h>
#include <TimerOne.h>
```

5.1.2.2. Global variable

```
#define SSID        "AP_SSID"       // SSID of your AP
#define Key         "1234567890"    // Key or Passphrase
unsigned char SIP[4] = {192, 168, 123, 119};
unsigned int ServerPort = 5000;
WizFi2x0Class myWizFi;
WizFiClient myClient;
```

```
    TimeoutClass ConnectInterval;
```

5.1.2.3. Timer1_ISR callback function

```
    void Timer1_ISR()
    {
        uint8_t i;
        myWizFi.ReplyCheckTimer.CheckIsTimeout();
    }
```

5.1.3. setup() function

```
void setup() {
  byte key, retval, i;

  Serial.begin(9600);
  Serial.println("₩r₩nSerial Init");

   myClient =  WizFiClient(SIP, ServerPort);

  // initalize WizFi2x0 module:
  myWizFi.begin();

  // Timer1 Initialize
  Timer1.initialize(1000); // 1msec
  Timer1.attachInterrupt(Timer1_ISR);

  myWizFi.SendSync();
  myWizFi.ReplyCheckTimer.TimerStart(1000);

  Serial.println("Send Sync data");

  while(1)
  {
    if(myWizFi.CheckSyncReply())
    {
      myWizFi.ReplyCheckTimer.TimerStop();
      Serial.println("Rcvd Sync data");
      break;
    }
```

```
    if(myWizFi.ReplyCheckTimer.GetIsTimeout())
    {
      Serial.println("Rcving Sync Timeout!!");
      return;
    }
  }

  ///////////////
  // AP association
  while(1)
  {
    retval = myWizFi.associate(SSID, Key, WEP_SECURITY, true);

    if(retval == 1){
      Serial.println("WizFi2xo AP Associated");
      Wifi_setup = true;
      break;
    }else{
      Serial.println("AP association Failed");
    }
  }

}
```

```
    Serial.begin(9600);
```

Reset the serial port baud rate to 9600bps.

```
    myClient = WizFiClient(SIP, ServerPort);

    // initalize WizFi2x0 module:
    myWizFi.begin();
```

Client 소켓을 SIP와 ServerPort를 이용해서 생성하고, WizFi2x0Class 객체를 생성하고 초기화합니다.

Use SIP and ServerPort to create client socket and create WizFi2x0Class object and initilize.

```
    // Timer1 Initialize
```

```
Timer1.initialize(1000); // 1msec
Timer1.attachInterrupt(Timer1_ISR);
```

Set the interrupt period of Timer1 as 1msec and register callback function of Timer1 interrupt to Timer1_ISR.

```
myWizFi.SendSync();
myWizFi.ReplyCheckTimer.TimerStart(1000);
while(1)
{
        if(myWizFi.CheckSyncReply())
        {
                myWizFi.ReplyCheckTimer.TimerStop();
                break;
        }
        if(myWizFi.ReplyCheckTimer.GetIsTimeout())
        {
                return;
        }
}
```

To check the communication status between the Arduino board and WizFi2x0 module, send the Sync byte and check the reply.

```
Retval = myWizFi.associate(SSID, Key, WEP_SECURITY, true);
```

Attempt connection with AP. **The EncryptionType in red font above must be modified according to the AP security type.**

5.1.4. loop() function

loop() function calls the code in the function infinitely; the codes in loop() function of WizFiBasicTest sketch is as below.

```
void loop()
{
    uint8_t retval, i;
    byte rcvdBuf[129];
    byte cmd;
    byte TxBuf[100];
```

```
        memset(rcvdBuf, 0, 129);
        if(Wifi_setup){
            myWizFi.RcvPacket();
            if(myClient.available()){
                if(myClient.read(rcvdBuf)){
            Serial.print("CID[");
            Serial.print((char)myClient.GetCID());
            Serial.print("]");
            Serial.println((char *)rcvdBuf);
            myClient.write(rcvdBuf);
          }
          if(Disconnect_flag){
            retval = myClient.disconnect();
            Disconnect_flag = false;
            if(retval == 1)
            Serial.println("Disconnected! ");
            else
            Serial.println("Disconnection Failed");
          }
        }else{
          if(Connect_flag){
            retval = myClient.connect();
            Connect_flag = false;
            if(retval == 1)
                    Serial.println("Connected! ");
            else
                    Serial.println("Connection Failed");
          }
            }
            CheckConsoleInput();
        }
  }
```

```
    myWizFi.RcvPacket();
```

RcvPakcet() function of WizFi2x0Class reads the data from the WizFi2x0 module and read the user's data and system control messages.

Note) must be called within loop().

```
    if(myClient.available()){
    if(myClient.read(rcvdBuf)){
      Serial.print("CID[");
      Serial.print((char)myClient.GetCID());
      Serial.print("]");
      Serial.println((char *)rcvdBuf);
      myClient.write(rcvdBuf);
    }
  }
```

Figure 8 Example of checking the socket connection and data sending and receiving

This is a sample of resending the data which was the received data printed on the serial console window while the client socket was connected to the server. 'myClient.available()' checks the connection with server and 'myClient.read(rcvdBuf)' checks whether there is received data or not. The received data is copied to rcvdBuf and the received byte number is returned. Call the myClient.write(rcvdBuf) function to resend the received data to the server.

```
    retval = myClient.connect();
```

myClient.connect() requests connection to the server.

This example is processed upon user's command.

```
    retval = myClient.disconnect();
```

myClient.disconnect() requests disconnection with the server.

This example is processed upon user's command.

```
    CheckConsoleInput();
```

CheckConsoleInput() function is used for the user's command flag setting.

5.2. WizFiBasicServerTest

5.2.1. Introduction

WizFiBasicServerTest sketch is processed as the following steps below.

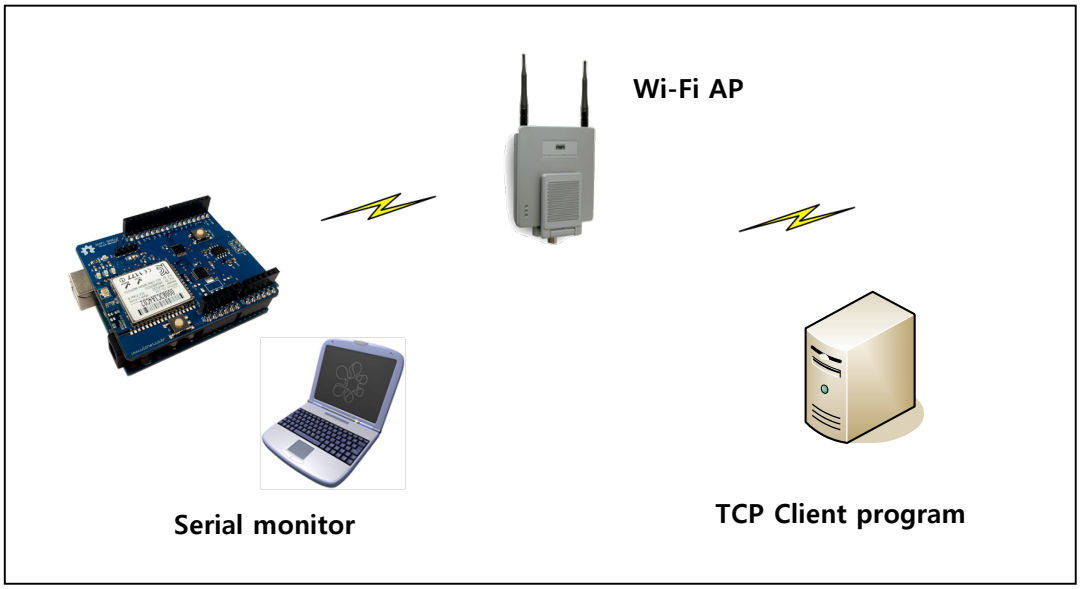


Figure 9 Network diagram of WizFiBasicServerTest

① **WizFiBasicServerTest connects with the assigned AP.**

② **Once the TCP connection from client is done, 4 WizFiClient sockets for data communication are declared.**

③ **Create a WizFiServer socket and wait for the connection attempt from the client socket to the assigned port number.**

④ **If each WizFiClient sockets are connected to the peer's system, check if there is received data. If there is received data, print the value to the serial monitor and resend the received data to the peer's system. This operation is repeated.**

The figure below is an example of the serial window after step③ is done successfully.

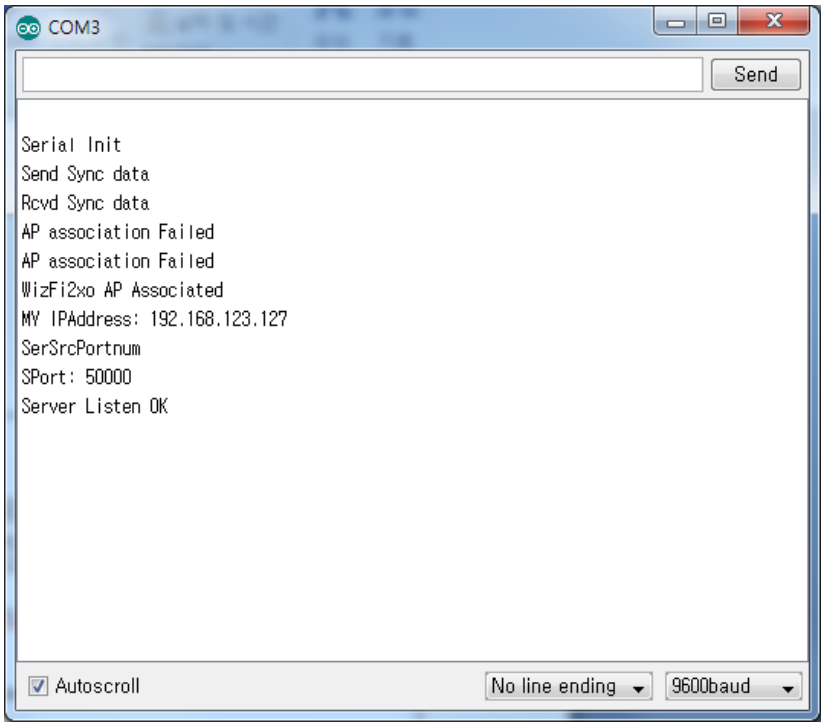


Figure 10 Normal operation of WizFiBasicServerTest

5.2.2. Declarative

5.2.2.1. Including header files

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SPI.h>
#include <WizFi2x0.h>
#include <WizFiClient.h>
#include <WizFiServer.h>
#include <TimerOne.h>
```

5.2.2.2. Variables

```
#define SSID                "AP_SSID"       // SSID of your AP
#define Key                 "1234567890"    // Key or Passphrase
unsigned char SIP[4] = {192, 168, 123, 119};
unsigned int ServerPort = 5000;
unsigned int SrcPort = 5000; //WizFiServer Listen port #

WizFi2x0Class myWizFi;
WizFiClient myClient[4]; //the number of Client : 4
```

```
WizFiServer myServer(SrcPort);
TimeoutClass ConnectInterval;
```

5.2.2.3. Timer1_ISR callback function

```
void Timer1_ISR()
{
    uint8_t i;
    myWizFi.ReplyCheckTimer.CheckIsTimeout();
}
```

5.2.3. setup() fucntion

The full composition of the setup() function is as below.

```
void setup() {
  byte key, retval, i;
  byte retry_count = 0;
  byte tmpstr[64];

  Serial.begin(9600);
  Serial.println("₩r₩nSerial Init");

  for(i=0; i<4; i++)
    myClient[i] =  WizFiClient();

  // initalize WizFi2x0 module:
  myWizFi.begin();

  ConnectInterval.init();
  // Timer1 Initialize
  Timer1.initialize(1000); // 1msec
  Timer1.attachInterrupt(Timer1_ISR);

  myWizFi.SendSync();
  myWizFi.ReplyCheckTimer.TimerStart(1000);

  Serial.println("Send Sync data");
  while(1)
  {
    if(myWizFi.CheckSyncReply())
    {
      myWizFi.ReplyCheckTimer.TimerStop();
      Serial.println("Rcvd Sync data");
      break;
    }
    if(myWizFi.ReplyCheckTimer.GetIsTimeout())
    {
      Serial.println("Rcving Sync Timeout!!");
      return;
    }
  }

  ////////////////////////////////////////////////////////
  // AP association
  while(1){
    byte tmpstr[32];
```

```
    retval = myWizFi.associate(SSID, Key, NO_SECURITY, true);

    if(retval == 1){
      myWizFi.GetSrcIPAddr(tmpstr);
      Serial.println("WizFi2xo AP Associated");
      Serial.print("MY IPAddress: ");
      Serial.println((char *)tmpstr);
      Wifi_setup = true;
      break;
    }else{
      Serial.println("AP association Failed");
    }
  }

  if(myServer.begin())
    Serial.println("Server Listen OK");
  else
    Serial.println("Server Listen Failed");
}
```

```
    for(i=0; i<4; i++)
        myClient[i] =  WizFiClient();


      // initalize WizFi2x0 module:
      myWizFi.begin();
```

In this example, 4 WizFiClient class are created to communicate with the peer's client class. Then, call myWizFi.begin() and initilize the myWizFi class. The process is similar to the WizFiBasicTest.

```
  if(myServer.begin())
    Serial.println("Server Listen OK");
  else
    Serial.println("Server Listen Failed");
```

The code above is for creating the server socket which will wait to connect with the assigned port number. '1' will return if creating the server socket is successful and '0' will return if failed.

5.2.4. loop() function

The entire composition of loop() function is as following.

```
    void loop()
    {
        uint8_t retval, i;
        byte rcvdBuf[129];
        byte cmd;
        byte TxBuf[100];
```

```
        memset(rcvdBuf, 0, 129);
      if(Wifi_setup){
           myWizFi.RcvPacket();
           for(i=0; i<4; i++){
             if(myClient[i].available()){
               if(myClient[i].read(rcvdBuf)){
                 Serial.print("CID[");
    Serial.print((char)myClient[i].GetCID());
                 Serial.print("]");
                 Serial.println((char *)rcvdBuf);
                 myClient[i].write(rcvdBuf);
               }
             }
           }
      }
}
```

The operation of loop() function and WizFiBasicTest is similar.

5.3. WizFiLimitedAPTest

5.3.1. Introduction

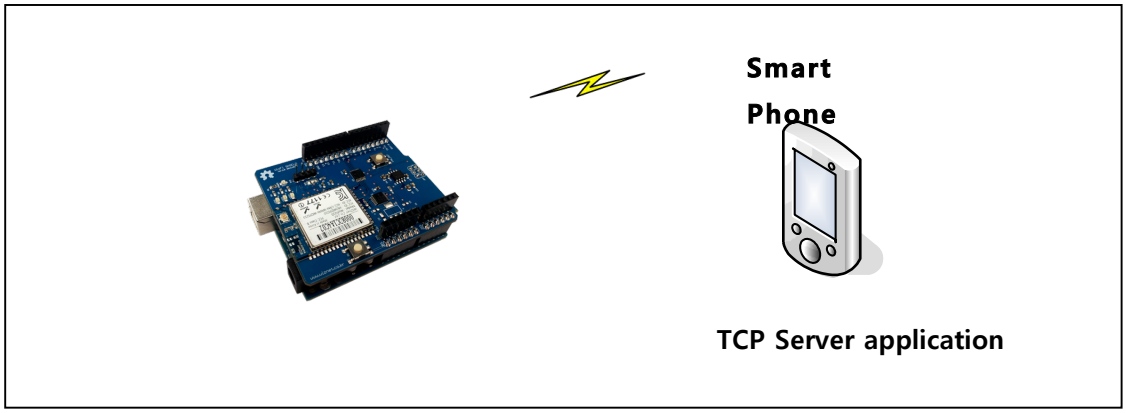


Figure 11 Network diagram WizFiLimitedAPTest

The WizFiLimitedAPTest sketch is done in the following order.

① The WizFiLimitedAPTest calls the SetOperatingMode() function to operate in "LIMITED_AP" mode.

② Call the associate(...) function to operate the WizFi210 module as AP.

③ If the initialization as AP is successful, a WizFiClient socket is created and waits for the user's command.

④ When the server connection command ('c' or 'C') is received through the serial console, the socket attempts connection with the server.

⑤ If the WizFiClient socket is connected to the Peer system, the received data will be printed to the serial port and re-send it to the peer system. This operation is repeated.

Below image is the serial monitor waiting for the user's server connection command when the WizFiLimitedAPTest is done correctly.

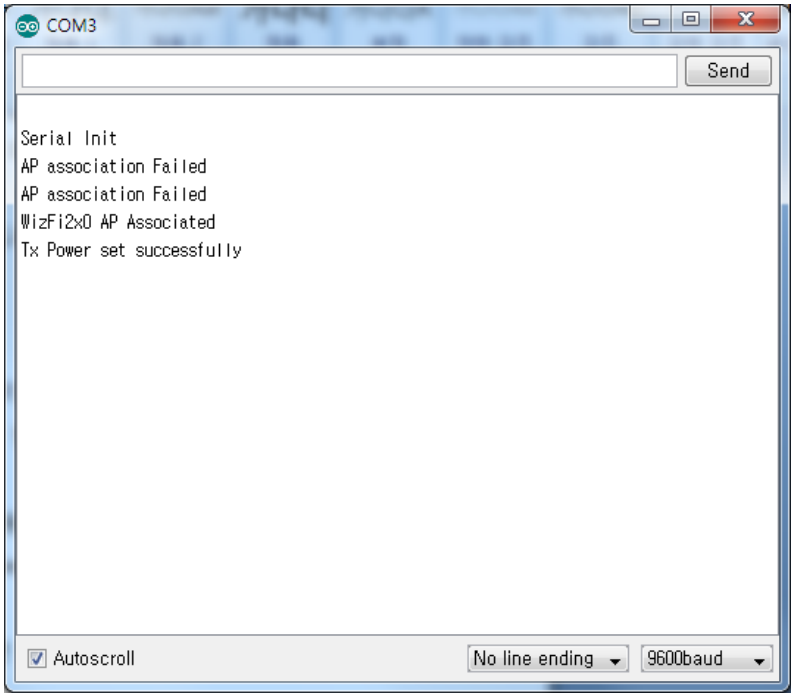


Figure 12 Serial monitor output of WizFiLimitedAPTest

5.3.2. Declarative

5.3.2.1. Including headers

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SPI.h>
#include <WizFi2x0.h>
#include <WizFiClient.h>
#include <WizFiServer.h>
#include <TimerOne.h>
```

5.3.2.2. Variables

```
#define SSID                "AP_SSID"       // SSID of your AP
#define Key                 "1234567890"    // Key or Passphrase
unsigned char SIP[4] = {192, 168, 123, 119};
unsigned int ServerPort = 5000;
unsigned int SrcPort = 5000; //WizFiServer Listen port #

WizFi2x0Class myWizFi;
```

```
WizFiClient myClient[4]; //the number of Client : 4
WizFiServer myServer(SrcPort);
TimeoutClass ConnectInterval;
```

5.3.2.3. Timer1_ISR callback function

```
void Timer1_ISR()
{
    uint8_t i;
    myWizFi.ReplyCheckTimer.CheckIsTimeout();
}
```

5.3.3. setup() function

The full composition of the setup() function is as below.

```
void setup() {
  byte key, retval, i;
  byte retry_count = 0;
  byte tmpstr[64];

  Serial.begin(9600);
  Serial.println("₩r₩nSerial Init");

  for(i=0; i<4; i++)
    myClient[i] =  WizFiClient();

  // initalize WizFi2x0 module:
  myWizFi.begin();

  ConnectInterval.init();
  // Timer1 Initialize
  Timer1.initialize(1000); // 1msec
  Timer1.attachInterrupt(Timer1_ISR);

  myWizFi.SendSync();
  myWizFi.ReplyCheckTimer.TimerStart(1000);

  Serial.println("Send Sync data");
  while(1)
  {
    if(myWizFi.CheckSyncReply())
    {
      myWizFi.ReplyCheckTimer.TimerStop();
      Serial.println("Rcvd Sync data");
      break;
    }
    if(myWizFi.ReplyCheckTimer.GetIsTimeout())
    {
      Serial.println("Rcving Sync Timeout!!");
      return;
    }
  }

  ////////////////////////////////////////////////////////
  // AP association
  while(1){
    byte tmpstr[32];
```

```
    retval = myWizFi.associate(SSID, Key, NO_SECURITY, true);

    if(retval == 1){
      myWizFi.GetSrcIPAddr(tmpstr);
      Serial.println("WizFi2xo AP Associated");
      Serial.print("MY IPAddress: ");
      Serial.println((char *)tmpstr);
      Wifi_setup = true;
      break;
    }else{
      Serial.println("AP association Failed");
    }
  }

  if(myServer.begin())
    Serial.println("Server Listen OK");
  else
    Serial.println("Server Listen Failed");
}
```

```
    for(i=0; i<4; i++)
        myClient[i] =  WizFiClient();


      // initalize WizFi2x0 module:
      myWizFi.begin();
```

In this example, 4 WizFiClient class are created to communicate with the peer's client class. Then, call myWizFi.begin() and innitiaize the myWizFi class. The process is similar to the WizFiBasicTest.

```
  if(myServer.begin())
    Serial.println("Server Listen OK");
  else
    Serial.println("Server Listen Failed");
```

The code above is for creating the server socket which will wait to connect with the assigned port number. '1' will return if creating the server socket is successful and '0' will return if failed.

5.3.4. loop() function

The entire composition of loop() function is as following.

```
    void loop()
    {
          uint8_t retval, i;
          byte rcvdBuf[129];
          byte cmd;
          byte TxBuf[100];


          memset(rcvdBuf, 0, 129);
```

```
    if(Wifi_setup){
        myWizFi.RcvPacket();
        for(i=0; i<4; i++){
          if(myClient[i].available()){
            if(myClient[i].read(rcvdBuf)){
              Serial.print("CID[");
  Serial.print((char)myClient[i].GetCID());
              Serial.print("]");
              Serial.println((char *)rcvdBuf);
              myClient[i].write(rcvdBuf);
            }
          }
        }
    }
}
```

The operation of loop() function and WizFiBasicTest is similar.

The difference is that the socket connections are 4 and checking all of these and communicate with connected 4 sockets.

5.4. WizFiLimitedAPTest

5.4.1. Introduction

LimitedAP Mode

Wi-Fi devices such as smart phones can connect with WizFi210 using Limited AP mode of WizFi210. However, since WizFi210 is designed for Embedded usage, the lack of program memory can result to not providing all functions as a normal AP. The LimitedAP mode is designed to provide limited but necessary functions for connection between the module and a Wi-Fi device.

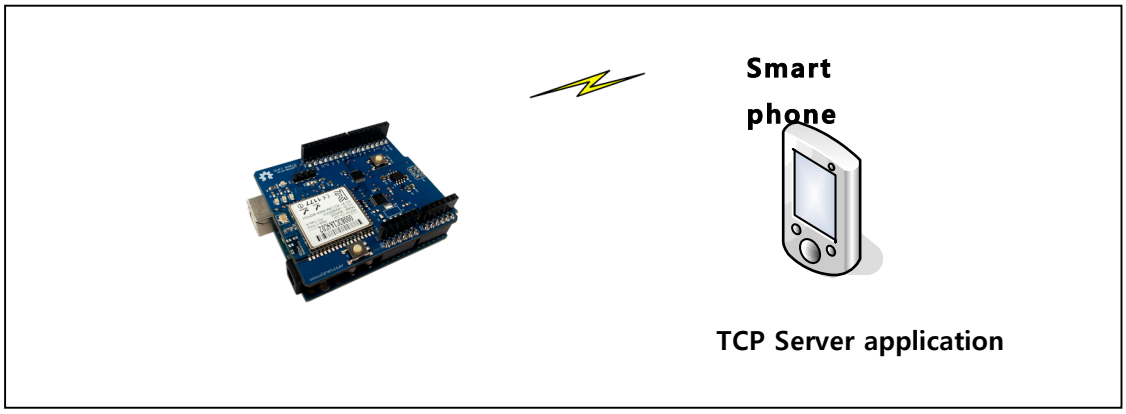


Figure 13 Network diagram of WizFiLimitedAPTest

WizFiLimitedAPTest sketch is done in the following order.

① **WizFiLimitedAPTest calls the SetOperatingMode() function to set the operation mode as "LIMITED_AP" mode.**

② **Call the associate(...) function to operate WizFi210 as AP.**

③ **After the initialization as AP is successful, create a WizFiClient socket to wait for the user's command.**

④ **Once the server connect command character('c' or 'C') is received through the serial monitor, the WizFiClient socket will attempt to connect with the server.**

⑤ **When the WizFiClient socket is connected to the peer's system, it checks whether or not there are any received data. If there is received data, the received data will print out to the serial port and be resent to the peer's system. This operation is repeated.**

The below image is the serial window waiting for the user's server connect

command.

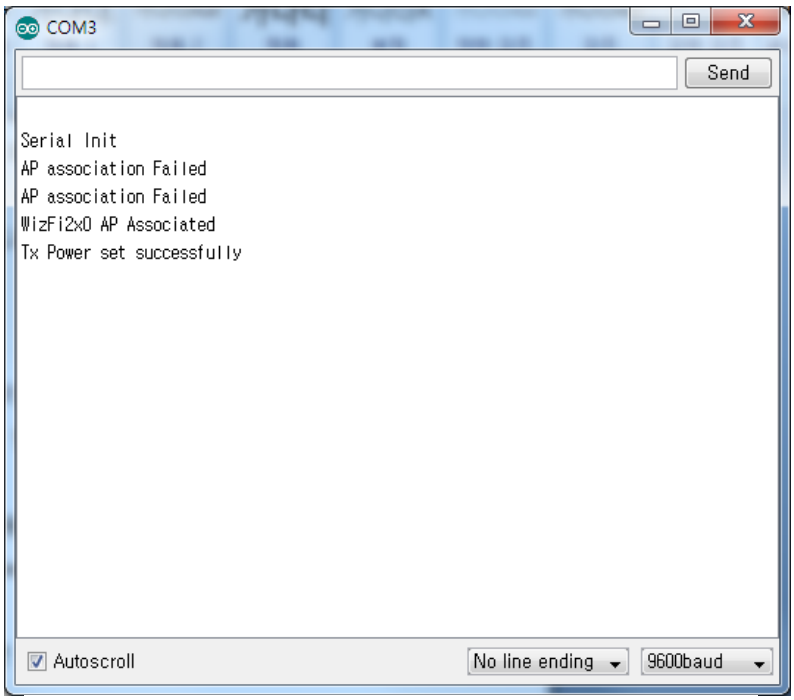


Figure 14 Serial monitor output of WizFiLimitedAPTest

5.4.2. Declarative

5.4.2.1. Including headers

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SPI.h>
#include <WizFi2x0.h>
#include <WizFiClient.h>
#include <TimerOne.h>
```

5.4.2.2. Variables

```
#define SSID        "LimitedAp"     // replace with SSID of your AP
#define Key         "1234567890"    // Key or Passphrase
unsigned char SIP[4] = {192, 168, 123, 119};
unsigned int ServerPort = 5000;
unsigned int SrcPort = 5000; //WizFiServer Listen port #


WizFi2x0Class myWizFi;
```

```
WizFiClient myClient;
TimeoutClass ConnectInterval;
```

When operating in LimitedAP mode, the value entered in the SSID string will be the SSID value shown from the Wi-Fi device. Also, the value entered in the Key character string will be the security key for connecting to the WizFi210 module set as LimitedAP.

5.4.2.3. Timer1_ISR callback function

```
void Timer1_ISR()
{
    uint8_t i;
    myWizFi.ReplyCheckTimer.CheckIsTimeout();
}
```

5.4.3. setup() function

```
void setup() {
  byte key, retval, i;
  byte retry_count = 0;
  byte tmpstr[64];

  Serial.begin(57600);
  Serial.println("₩r₩nSerial Init");

  // initalize WizFi2x0 module:
  myWizFi.begin();
  myClient = WizFiClient(SIP, ServerPort);

  // Timer1 Initialize
  Timer1.initialize(1000); // 1msec
  Timer1.attachInterrupt(Timer1_ISR);

  myWizFi.SetSrcIPAddr((byte *)"192.168.55.1");
  myWizFi.SetSrcSubnet((byte *)"255.255.255.0");
  myWizFi.SetSrcGateway((byte *)"192.168.55.1");

  myWizFi.SetOperatingMode(LIMITEDAP_MODE);

  myWizFi.SendSync();
  myWizFi.ReplyCheckTimer.TimerStart(1000);

  Serial.println("Send Sync data");

  while(1)
  {
    if(myWizFi.CheckSyncReply()){
      myWizFi.ReplyCheckTimer.TimerStop();
      Serial.println("Rcvd Sync data");
      break;
    }
    if(myWizFi.ReplyCheckTimer.GetIsTimeout()){
      Serial.println("Rcving Sync Timeout!!");
      return;
    }
  }
  ////////////////////////////////////////////////////////
  // AP association
  while(1)
  {
    retval = myWizFi.associate(SSID, Key, WEP_SECURITY, false);

    if(retval == 1){
```

```
      Serial.println("WizFi2x0 AP Associated");
      Wifi_setup = true;
      break;
    }else{
      Serial.println("AP association Failed");
    }
  }
  retval = myWizFi.SetTxPower(5);
  if(retval == 1){ Serial.println("Tx Power set successfully"); }
  else{ Serial.println("Tx Power parameter is invalid"); }
  delay(1000);
}
```

WizFi2x0 operates as a Gateway during LimitedAP mode. Therefore, the IP address, Subnet mask, and Gateway address of the WizFi2x0 module must be manually set.

```
myWizFi.SetSrcIPAddr((byte *)"192.168.55.1");
myWizFi.SetSrcSubnet((byte *)"255.255.255.0");
myWizFi.SetSrcGateway((byte *)"192.168.55.1");
```

Please note that the setting for the Source IP Address and Source Gateway Address is identically set to "192.168.55.1."

```
myWizFi.SetOperatingMode(LIMITEDAP_MODE);
```

Set the operation mode of the WizFi2x0 module after the IP address setting.

```
retval = myWizFi.associate(SSID, Key, WEP_SECURITY, false);
```

Call the associate(…) function to initialize the WizFi2x0 module as "Limited AP."

```
retval = myWizFi.SetTxPower(5);
```

The above code is for setting the RF output level of the WizFi2x0 module to '5.'

5.4.4. loop() function

The entire composition of the loop() function is as below.

```
void loop(){
  uint8_t retval, i;
  byte rcvdBuf[129];
  byte TxBuf[100];

  memset(rcvdBuf, 0, 129);
  if(Wifi_setup){
    myWizFi.RcvPacket();
    if(myClient.available()){
```

```
      if(myClient.read(rcvdBuf)){
        Serial.print("CID[");
        Serial.print((char)myClient.GetCID());
        Serial.print("]");
        Serial.println((char *)rcvdBuf);
        Serial.flush();
        myClient.write(rcvdBuf);
      }
      if(Disconnect_flag){
        retval = myClient.disconnect();
        Disconnect_flag = false;
      }
    }else{
      if(Connect_flag){
        retval = myClient.connect();
        Connect_flag = false;
      }
    }
    CheckConsoleInput();
  }
}
```

The operation of loop() function is similar to WizFiBasicTest.

5.5. WizFiUDPClientTest

5.5.1. Introduction

The WizFiUDPClientTest shows how to create a WizFiUDP class and how to send/receive data when the peer's IP address and port number is known. Please refer to section 5.6 "WizFiUDPServerTest" when the peer's IP address and port number is unknown and only the user's port number is known.

WizFiUDPClientTest sketch is done in the following order.

① One WizFiUDP class is created.

② Call the associate(…) function to connect with AP.

③ After connecting to the AP, create a UDP socket based on the data saved in the WizFiUDP class.

④ Start the counter of Timer(ConnectInterval) which works in unit of 100ms.

⑤ If Timeout occurs, it means that 100ms is elapsed. Then, send a character string stating "Time elapsed..I'm alive" to UDP.

⑥ Repeat step ④ and ⑤.

The below image is the serial window during the WizFiUDPClientTest.

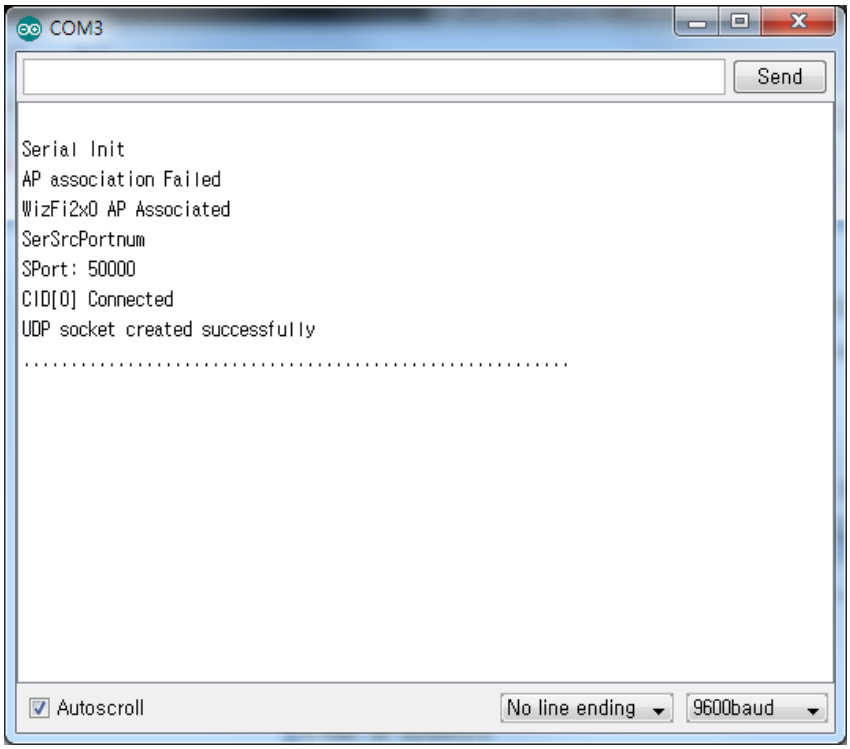


Figure 15 Serial monitor output of WizFiUDPClientTest

5.5.2. Declarative

5.5.2.1. Including headers

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SPI.h>
#include <WizFi2x0.h>
#include <WizFiUDP.h>
#include <TimerOne.h>
```

Include the 'WizFiUDP.h' header file to use the UDP socket.

5.5.2.2. Variables

```
#define SSID                "AP_SSID"       // SSID of your AP
#define Key                 "1234567890"    // Key or Passphrase

unsigned char  SIP[4]               = {192, 168, 123, 170};
unsigned int ServerPort = 50000;

WizFi2x0Class myWizFi;
WizFiUDP myUDP;
TimeoutClass ConnectInterval;
```

Define myUDP which is a WizFiUDP type class.

5.5.2.3. Timer1_ISR Callback function

```
void Timer1_ISR()
{
  uint8_t i;

  myWizFi.ReplyCheckTimer.CheckIsTimeout();
  ConnectInterval.CheckIsTimeout();
}
```

Since the UDP packet will be sent every 100ms, call the CheckIsTimeout() member function inside the Timer1_ISR Callback() function in order to check the elapsed time of the ConnectInterval.

5.5.3. setup() function

```
void setup()
{
  byte key, retval, i;
  byte retry_count = 0;
  byte tmpstr[64];

  Serial.begin(9600);
  Serial.println("₩r₩nSerial Init");

  // initalize WizFi2x0 module:
  myWizFi.begin();
  myUDP =  WizFiUDP(SIP, ServerPort, 50000);

  ConnectInterval.init();

  // Timer1 Initialize
  Timer1.initialize(1000); // 1msec
  Timer1.attachInterrupt(Timer1_ISR);



  myWizFi.SendSync();
  myWizFi.ReplyCheckTimer.TimerStart(1000);

  //Serial.println("Send Sync data");

  while(1)
  {
    if(myWizFi.CheckSyncReply())
```

```
        {
          myWizFi.ReplyCheckTimer.TimerStop();
          //Serial.println("Rcvd Sync data");
          break;
        }
        if(myWizFi.ReplyCheckTimer.GetIsTimeout())
        {
          //Serial.println("Rcving Sync Timeout!!");
          return;
        }
      }

      ////////////////////////////////////////////////////////////  // AP association
      while(1)
      {
        retval = myWizFi.associate(SSID, Key, WEP_SECURITY, true);

        if(retval == 1){
          Serial.println("WizFi2x0 AP Associated");
          Wifi_setup = true;
          break;
        }else{
          Serial.println("AP association Failed");
        }
      }

      if(myUDP.open())
        Serial.println("UDP socket created successfully");
      else
        Serial.println("UDP socket creation failed");

      ConnectInterval.TimerStart(100);
    }
```

```
myUDP =  WizFiUDP(SIP, ServerPort, 50000);
```

Set the peer's IP address, port number, and user's port number to create myUDP.

```
if(myUDP.open())
```

Call open() function to create a UDP socket.

5.5.4. loop() function

```
void loop()
{
  uint8_t retval, i;
  byte rcvdBuf[129];
  byte cmd;
  byte TxBuf[100];

  memset(rcvdBuf, 0, 129);

  if(Wifi_setup)
  {
    myWizFi.RcvPacket();

    if(myUDP.available())
    {
      if(ConnectInterval.GetIsTimeout())
      {
        myUDP.write((byte *)"Time elapsed..I'm alive");
        ConnectInterval.TimerStart();
        Serial.println("Time elapsed..I'm alive ");
      }
    }
  }
}
```

```
myUDP.write((byte *)"Time elapsed..I'm alive");
```

Call write() function to create a UDP packet.

5.6. WizFiUDPServerTest

The WizFiUDPServerTest uses only the user's port number when creating a WizFiUDP class. Most parts are similar to the WizFiUDPClientTest. However, the WizFiUDPServerTest is designed to print the peer's IP address and port number in the serial monitor when a data is received, and resend that data to the peer or a particular IP address and port number.

```
myUDP = WizFiUDP((uint8_t *)NULL, 0, SrcPort);
```

```
void loop()
{
  uint8_t retval, i;
  byte rcvdBuf[129];
  byte cmd;
  byte TxBuf[100];
  byte tmpIP[16];
  uint16_t tmpPort;

  memset(rcvdBuf, 0, 129);

  if(Wifi_setup)
  {
    myWizFi.RcvPacket();
    if(myUDP.available())
    {
      if(myUDP.read(rcvdBuf))
      {
          Serial.print("CID[");
          Serial.print((char)myUDP.GetCID());
          Serial.print("]");
          Serial.println((char *)rcvdBuf);
```

```
        myUDP.GetCurrentDestInfo(tmpIP, &tmpPort);
        Serial.println((char *)tmpIP);
        sprintf((char *)tmpIP, "Portnum: %u", tmpPort);
        Serial.println((char *)tmpIP);
        myUDP.write(rcvdBuf);
        myUDP.SetCurrentDestInfo((byte *)"192.168.123.170", 5500);
        myUDP.write(rcvdBuf);
    }
   }
  }
}
```