GiGA IoTMakers

Arduino SDK 매뉴얼

Document info

Document name: "Arduino SDK 매뉴얼"

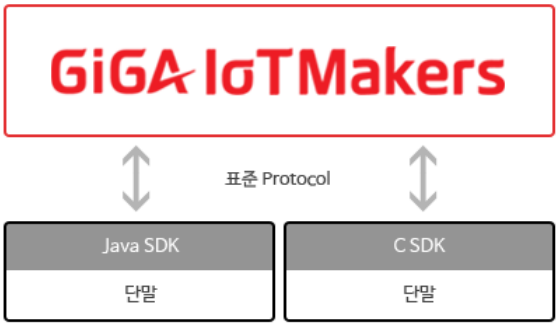
Document date: 2016-05-02

Manual release version: 2.0.0

목차

1 개요

IoTMakers SDK(Software Development Kit)는 IoT 디바이스(이하 디바이스)가 IoTMakers 플랫폼에 연동할 때에 필요한 API(Application Programming Interface)를 제공한다. 디바이스는 제공되는 API 를 통하여 접속 및 장비인증, 수집데이터 전송하고, 플랫폼으로부터 제어데이터를 수신한다.



[그림 1] IoTMakers SDK 연동 구조

디바이스가 IoTMakers 플랫폼에 연동하기 위해서는 IoTMakers 포털에서 디바이스 등록과정을 거쳐야 한다. 자세한 내용은 IoTMakers 포털(http://iotmakers.kt.com)에서 확인한다.

이 문서는 IoTMakers SDK 를 사용하는 프로그래머에게 프로그래밍 가이드와 레퍼런스를 제공한다.

2 IoTMakers SDK 시작하기

IoTMakers Arduino SDK 는 라이브러리 소스 형태로 제공되어, 사용자가 경량의 아두이노 환경에서 IoTMakers 플랫폼에 쉽게 연동할 수 있도록 도와준다. 좀더 자세히는 TCP 프로토콜을 사용하여 플랫폼에 접속하여 단말 인증을 수행하고, 센서를 통하여 수집된 데이터를 전송하고, 반대로 플랫폼에서 전송된 단말제어 메시지를 수신/처리한다.

이후 IoTMakers SDK 는 IoTMakers 아두이노 라이브러리로 이해하도록 하자

2.1 시작하기 전에

2.1.1 아두이노 보드(Board)

아두이노는 사양에 따라 여러 종류의 보드가 있다. IoTMakers SDK 는 가장 보편적인 아두이노 우노(Uno)이상의 사양에서 작동한다. http://www.arduino.cc/en/Main/Products 에서 다양한 아두이노 보드를 확인할 수 있다.

2.1.2 아두이노 통합개발환경(IDE)

사용자는 이미 아두이노와 그 개발환경을 숙지하고 있다고 가정한다. 참고로 아두이노 공식 튜토리얼 사이트는 https://www.arduino.cc/en/Tutorial/HomePage 이다.

아두이노 IDE 는 1.6.4 이후 버전을 권장한다. 이 문서를 작성하는 시점의 최신 IDE 버전은 1.6.8 이다. 아두이노 IDE 는 https://www.arduino.cc/en/Main/Software 에서 내려받을 수 있다.

2.1.3 아두이노 라이브러리

IoTMakers SDK 는 아두이노 라이브러리 형태로 제공되기 때문에 그 관리방법을 숙지하고 있어야 한다. 사용자 라이브러리는 통상적으로 사용자 홈폴더의 아두이노 폴더에 위치시킨다. IoTMakers 라이브러리도 사용자의 아두이노 폴더에 저장한다. http://www.arduino.cc/en/Guide/Libraries 에서 자세한 내용을 확인할 수 있다.

2.1.4 아두이노 네트워크 쉴드(Shield)

일반적으로 아두이노 보드는 네트워크를 자체적으로 지원하지 않기 때문에 보드와 호환되는 추가적인 하드웨어 모듈(Shield)가 필요하다.
https://store.arduino.cc/category/68 에서 다양한 호환 쉴드를 확인할 수 있다. IoTMakers SDK 는 아두이노 우노와 그 호환 와이파이 쉴드에서 사용하기를 권장한다.

사용자는 이더넷 또는 와이파이 쉴드를 사용하여 인터넷에 접속해 보고 정상작동 여부를 미리 확인해 두어야 한다. 참고로, IoTMakers SDK 의 examples 코드에 와이파이 쉴드를 사용하기 위한 Shield_Wrapper.cpp 코드를 작성해 두었다.

무선 와이파이 쉴드를 사용할 경우에는 주변에 있는 와이파이의 무선접속 아이디와 비밀번호를 미리 확인해 둔다.

유선 이더넷 쉴드를 사용하는 경우에는 인터넷에 연결된 랜케이블을 확보해 둔다.

2.2 IoTMakers SDK 설치

IoTMakers 포털에 로그인 후에 해당 SDK 를 내려 받는다. 내려 받은 SDK 를 사용자의 아두이노 라이브러리 폴더에 저장 후에 압축을 푼다.

예를 들면,

a) MS Windows
My Documents₩Arduino₩libraries₩

b) Mac OS X
~/Documents/Arduino/libraries/

c) Ununtu
~/Documents/Arduino/libraries

libraries/IoTMakers/ 의 폴더 구조는 다음과 같다.

파일 및 폴더명	내용
IoTMakers/src	IoTMakers SDK 소스 폴더
IoTMakers/src/IoTMakers.h	IoTMakers 클래스 헤더
IoTMakers/src/IoTMakers.cpp	IoTMakers 클래스 구현체
IoTMakers/examples	IoTMakers SDK 예제 폴더

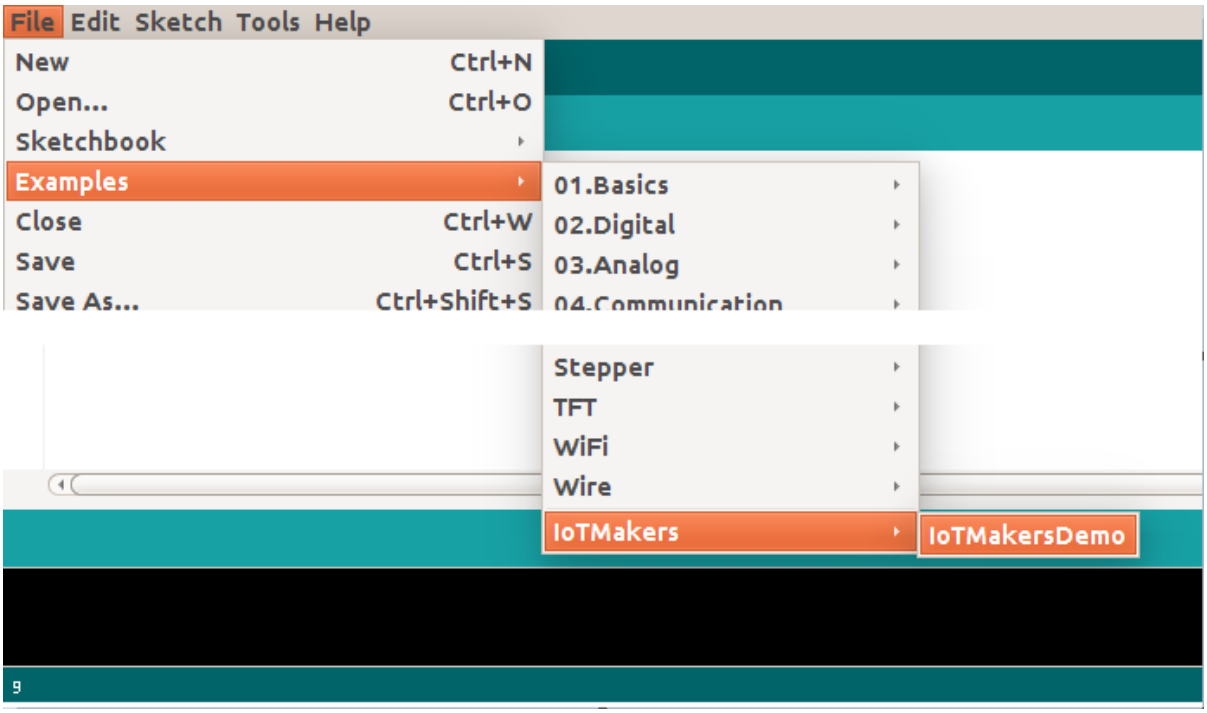
IoTMakers/examples/IoTMakersDemo	IoTMakersDemo 예제

그리고, 아두이노 IDE 를 다시 시작한다.

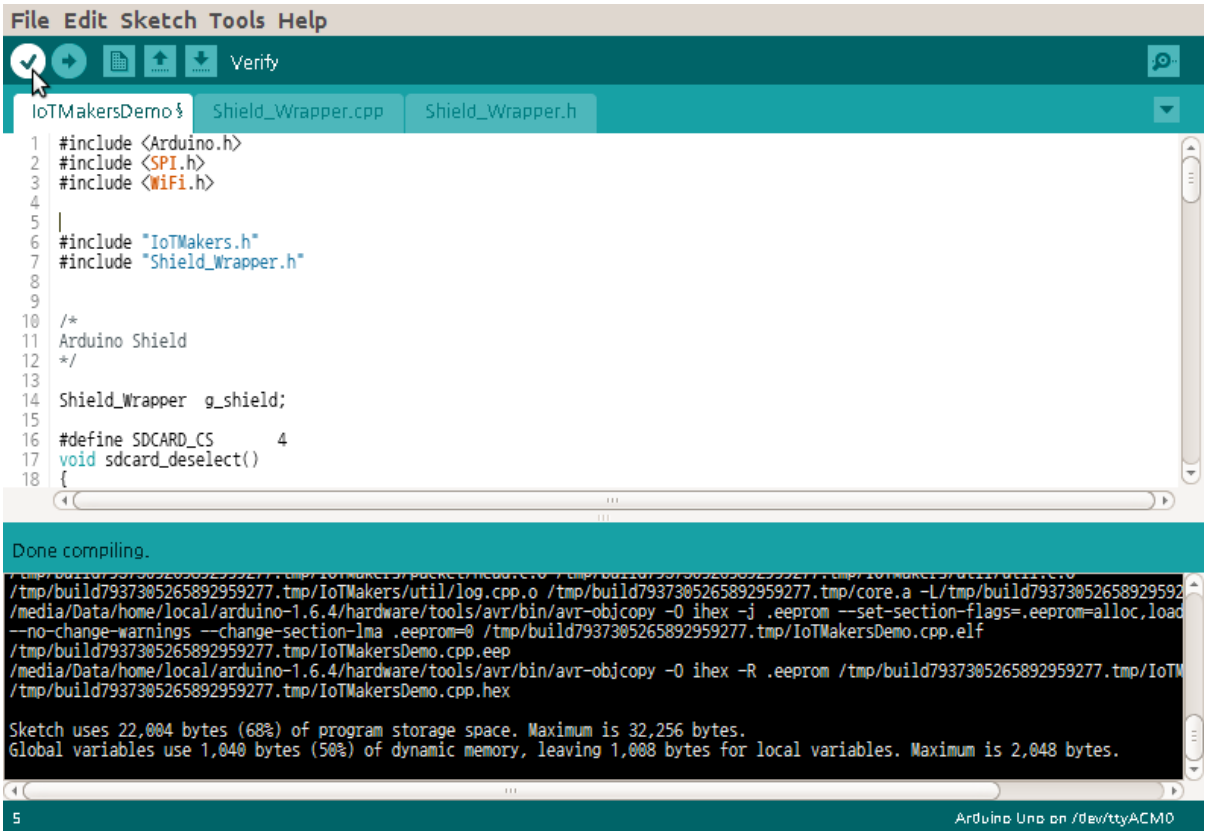
2.3 IoTMakersDemo

IoTMakers SDK 를 사용자의 libraries 폴더에 복사한 후에 아두이노 IDE 를 재시작하면, 바로 IoTMakers 를 사용할 수 있다. IoTMakers SDK 에서 기본 제공하는 예제 코드를 불러와서 빌드해 보자.

IoTMakersDemo 의 위치는 IDE 에서 File/Examples/IoTMakers/IotMakersDemo 이다.



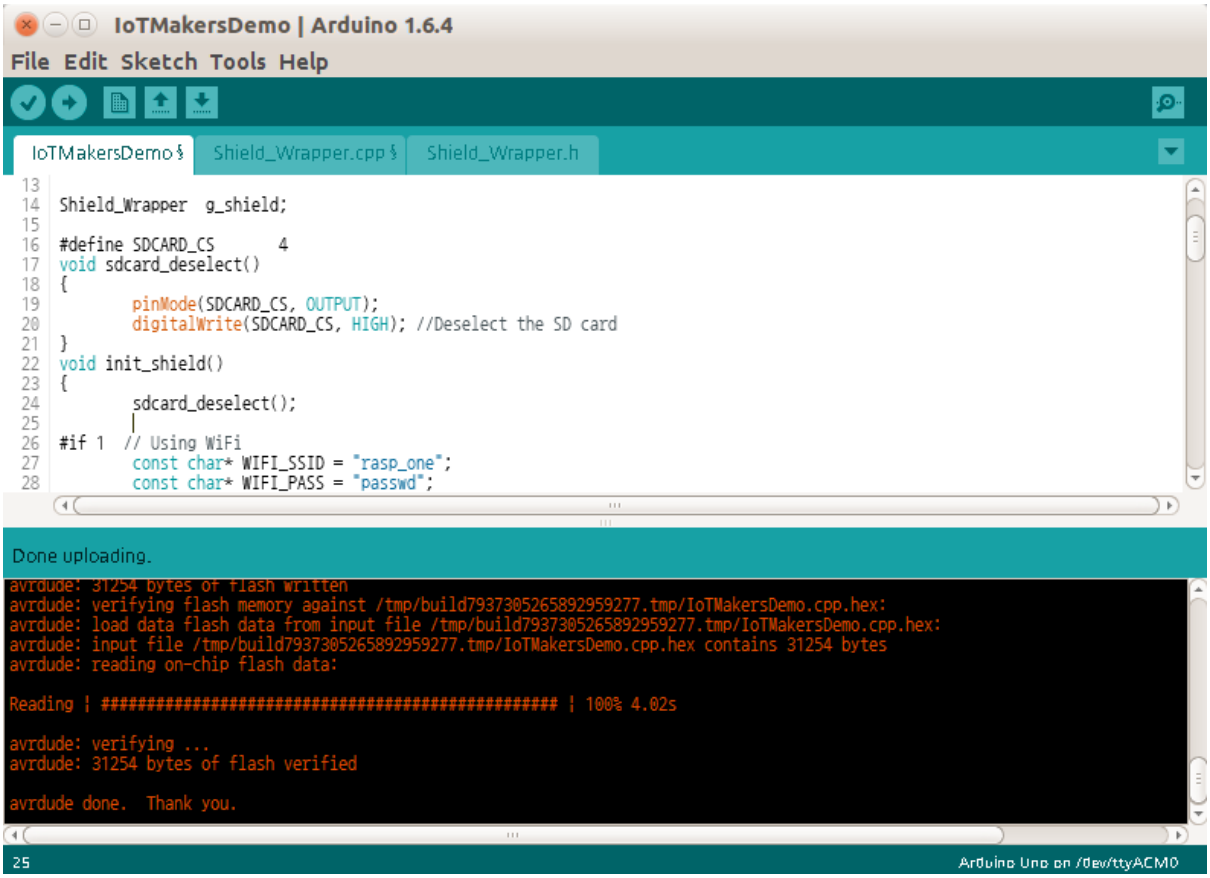
아래 그림과 같이 Verify 버튼을 클릭하여 일단 빌드를 확인한다.



무선랜에 접속을 확인하기 위하여 WIFI_SSID 와 WIFI_PASS 를 사용자의 공유기에 설정된 값으로 설정한다.

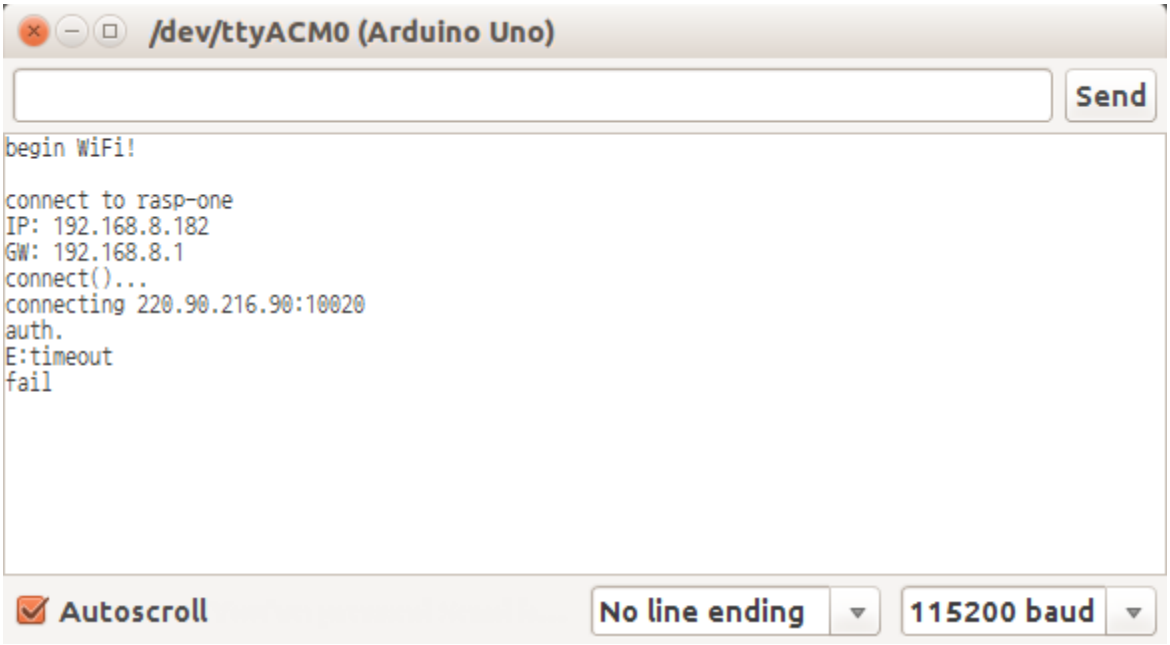
```
void init_shield()
{
        sdcard_deselect();

        const char* WIFI_SSID = "rasp_one";
        const char* WIFI_PASS = "passwd";
        g_shield.begin(WIFI_SSID, WIFI_PASS);

// 생략
```

다시 Verify 버튼을 클릭하여 일단 빌드를 하고, 아래와 같이 Upload 버튼을 클릭하여 아두이노에 전송한다.

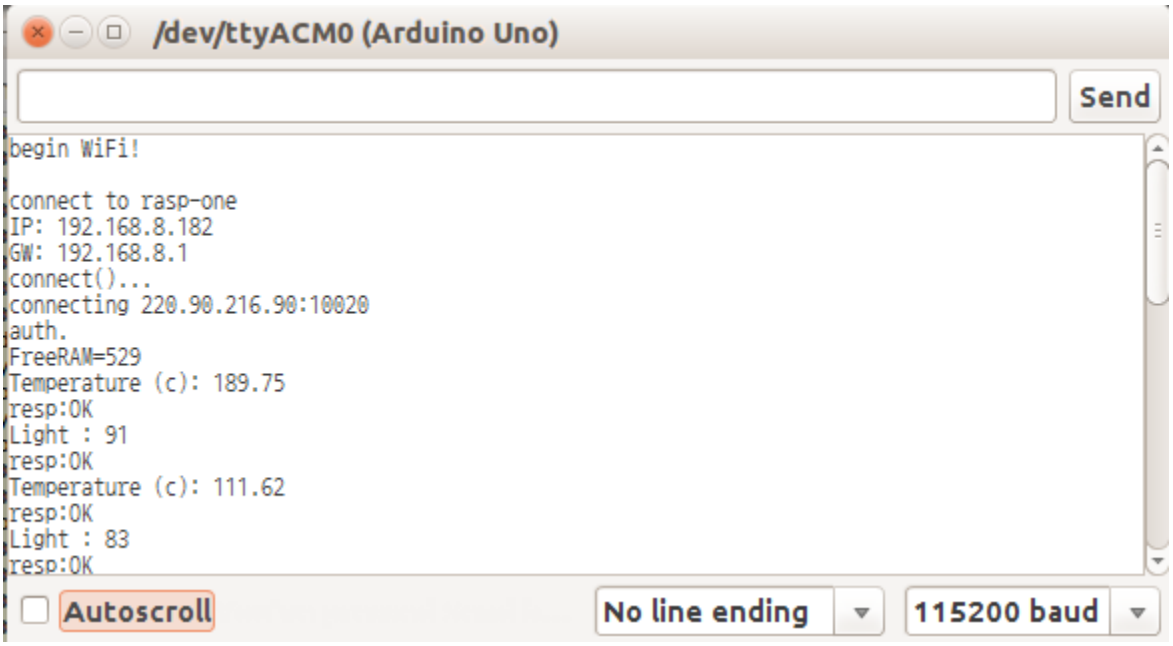


Upload 를 마친 후에 시리얼 모니터로 출력을 확인한다. 이 단계에서는 공유기에 접속해서 디바이스에 ip 를 할당받고(예, 192.168.8.182), IoTMakers 플랫폼(220.90.216.90:10020) 접속까지 성공해야 한다.

여기서, 올바른 디바이스 정보를 입력하지 않았으므로 auth 는 fail 이다.



진행과정에서, 정상적인 디바이스 정보를 지정하면 아래 그림과 같이 데이터를 전송한다. 실제 센서가 부재하므로 데이터 값은 실제 값이 아니다. 참고만 하자.



참고로, 유선의 이더넷 쉴드를 사용할 경우에는 다음의 안내 대로 코드를 수정해준다.

a. Shield_Wrapper.cpp 파일 수정

Shield_Wrapper.cpp 소스의 모든 내용을 지우고 arduinio/libraries/IoTMakers/examples/IoTMakersDemo/Shield_Wrapper.cpp_Ethernet 파일의 코드를 복사하여 사용한다.

b. IoTMakersDemo.ino 파일 수정

헤더파일 변경

```
#include <WiFi.h>
 라인을 지우고 같은 자리에
#include <Ethernet.h>
를 추가한다
```

쉴드 접속 소스코드 부분 변경

```
      const char* WIFI_SSID = "WIFI_SSID";
```

```
      const char* WIFI_PASS = "WIFI_PASS";
      g_shield.begin(WIFI_SSID, WIFI_PASS);
```

라인을 지우고 같은 자리에

```
      const byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
      // Set the static IP address to use if the DHCP fails to
assign
      const IPAddress ip(192, 168, 0, 177);
      g_shield.begin(mac, ip);
```

를 추가한다.

3 IoTMakers Class 메소드 목록

주요 메소드는 다음과 같다.

분류	API명	내용
생성자	IoTMakers()	IoIMakers클래스의 객체를 생성
초기화	init()	IoTMakers 포털에 등록한 디바이스 정보로 초기화
	set_strdata_handler()	장비제어데이터 핸들러 등록(문자열타입)
	set_dataresp_handler()	수집데이터 전송결과 처리 핸들러 등록
	set_dataresp_handler()	오류 처리 핸들러 등록
접속/장비인증	connect()	IoTMakers에 TCP접속을 시도
	disconnect()	IoTMakers에 TCP접속 종료
	auth_device()	디바이스 인증 시도
데이터 전송	send_numdata()	태그스트림이름과 숫자값을 전송
	send_strdata()	태그스트림이름과 문자열값을 전송
내부처리	loop()	접속유지 및 제어데이터 수신

참고로, IoTMakers 플렛폼에서 보내는 디바이스 제어데이터를 처리하기 위한 사용자 콜백(callback)함수의 타입은 다음과 같다.

콜백함수 형	설명
typedef void (*IMCbTagidNumDataHndl)(char *tagid, double val)	제어:숫자형 태그 데이터 처리 함수 타입
typedef void (*IMCbTagidStrDataHndl)(char *tagid, char *val)	제어:문자형 태그데이터 처리 함수 타입
typedef void (*IMCbDataRespHndl)(unsigned long long trxid, char *respCode)	수집:수집데이터 전송결과 처리 함수 타입

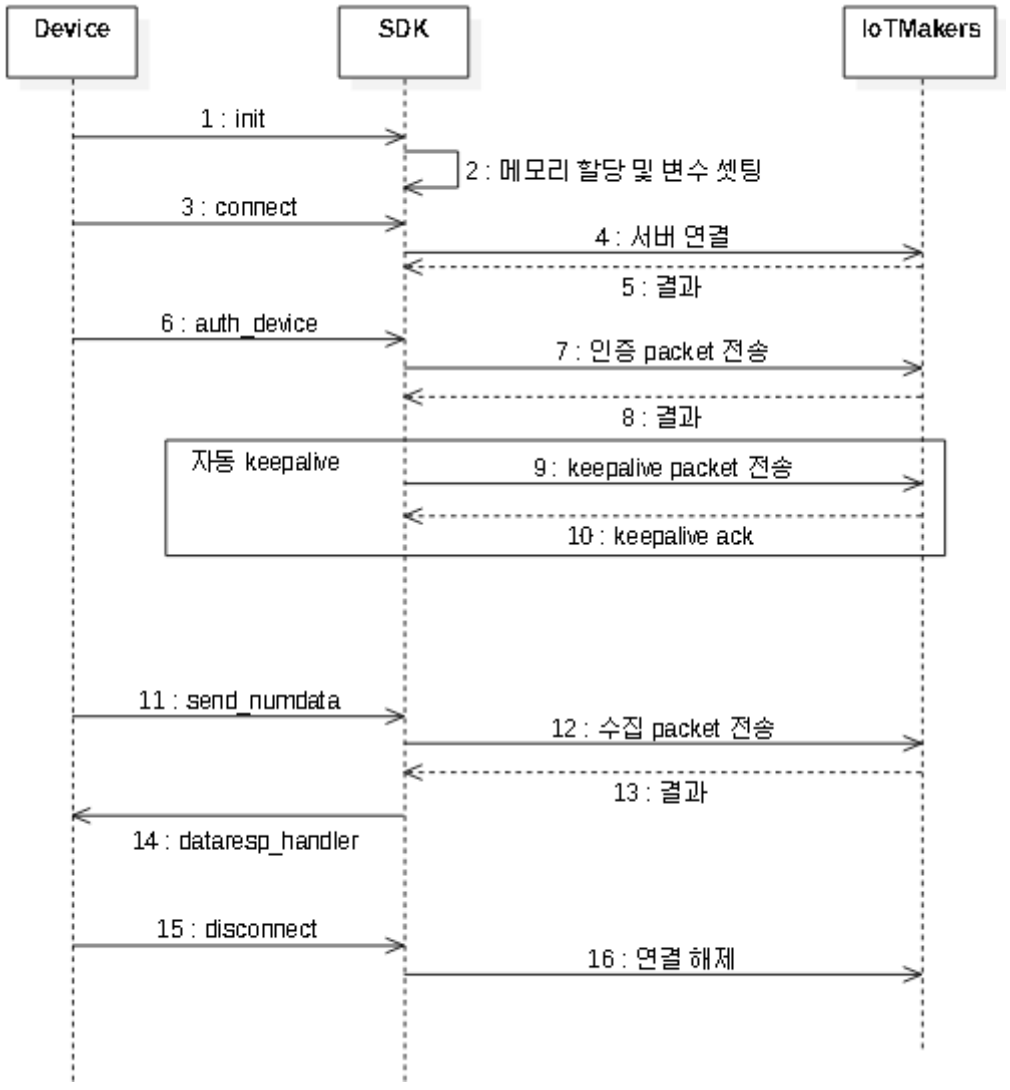
4 디바이스 연동(TCP 방식)

디바이스 연동을 하기 위하여 a) 디바이스 아이디, b) 디바이스 패스워드, 그리고 c) Gateway 연결 ID 가 필요하다. 이 정보는 IoTMakers 포털의 사용자 계정에서 새 디바이스를 등록한 후에 확인할 수 있다.

4.1 수집데이터 전송 시나리오

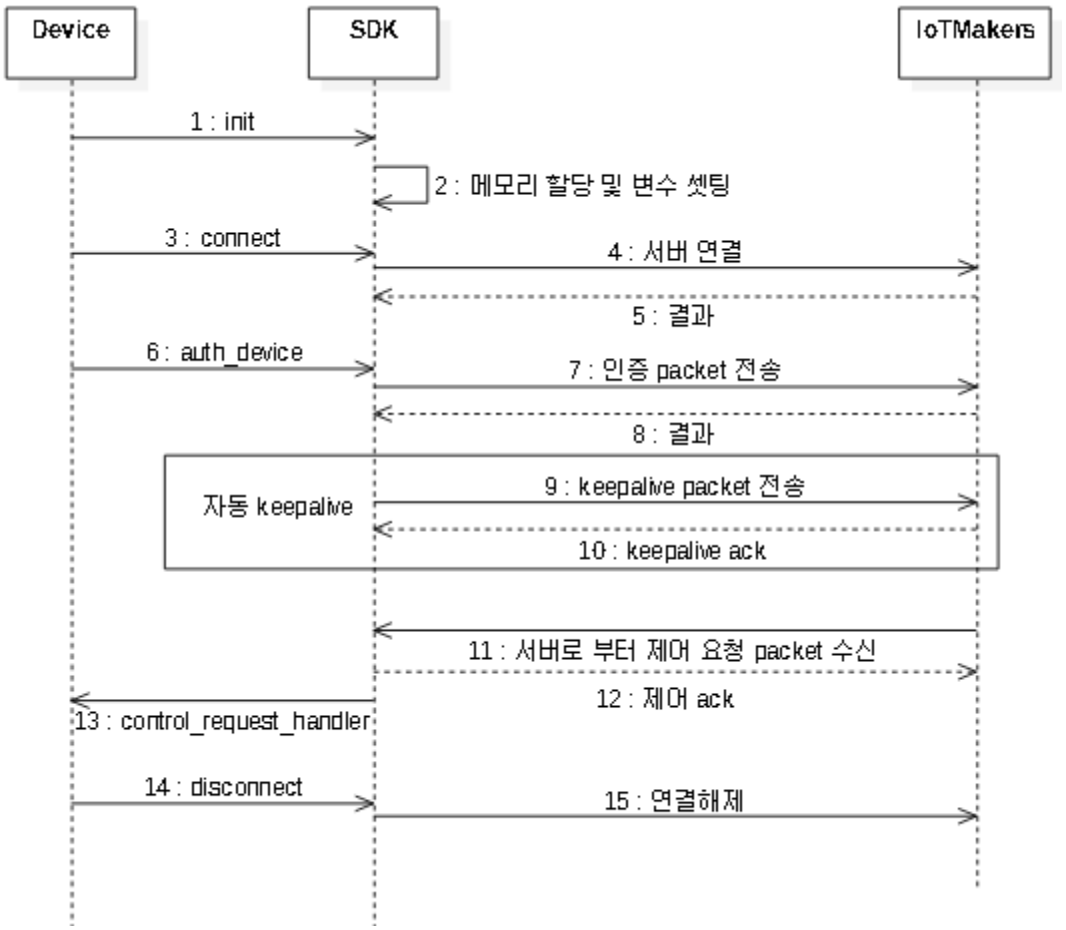
TCP Connection 연결후에, IoTMakers 플렛폼에 디바이스 채널의 인증을 시도한다. 인증시 디바이스아이디, 디바이스패스워드, Gateway 연결 Id 를 넘겨준다. 인증 성공시에는 현재 채널에 대한 인증번호(authNum)를 받는다.

디바이스에서 수집된 데이터를 IoTMaker 에 전송한다. 태그스트림아이디(tagId)와 수집데이터(val) 쌍을 전송한다. 데이터타입은 숫자형과 문자형을 전송할 수 있다.



4.2 제어데이터 수신 시나리오

IoTMakers 플렛폼에서 제어데이터를 디바이스로 전송하는 경우이다. 디바이스가 제어데이터 수신하면 즉시 응답하고, 장비제어 콜백를 통하여 제어처리한다. C SDK 는 내부적으로 loop() 안에서 제어데이터를 수신하고, 데이터를 추출하여 사용자가 정의한 콜백 함수를 호출한다.



5 IoTMakers 메소드 상세설명

IoTMakers SDK 에서 제공하는 메소드를 설명한다.

5.1 init(deviceId, devicePasswd, gatewayId, client)

IoTMakers 클라이언트 인스턴스를 초기화한다.

int init(const char *deviceId, const char * devicePasswd, const char * gatewayId, Client& client);		
param	deviceId	디바이스 아이디
	devicePasswd	디바이스 패스워드
	gatewayId	등록된 디바이스의 Gateway연결ID
	client	아두이노 쉴드의 클라이언트 객체
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
// 생략
Shield_Wrapper    g_shield;
// 생략

/*
IoTMakers
*/
IoTMakers g_im;

const char deviceID[]   = "XXXXXXXX458541656289";
const char devicePasswd [] = "XXXXXXXXt";
const char gatewayId []  = "OPEN_TCP_001PTL001_100XXXXXXX";

void init_iotmakers()
{
      Client* client = g_shield.getClient();
      if ( client == NULL )    {
            Serial.println(F("No client from shield."));
            while(1);
      }

      g_im.init(deviceID, devicePasswd, gatewayId, *client);
// 생략
```

5.2 init(ip, port, deviceId, devicePasswd, gatewayId, client)

IoTMakers 클라이언트 인스턴스를 초기화한다. IoTMakers 플랫폼의 ip 주소와 port 를 명시적으로 지정한다.

int init(const char *ip, int port, const char *deviceId, const char * devicePasswd, const char * gatewayId, Client& client);		
param	ip	IoTMakers서버 아이피
	port	IoTMakers서버 포트
	deviceId	디바이스 아이디
	devicePasswd	디바이스 패스워드
	gatewayId	등록된 디바이스의 Gateway연결ID
	client	아두이노 쉴드의 클라이언트 객체
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
// 생략
Shield_Wrapper    g_shield;
// 생략

/*
IoTMakers
*/
IoTMakers g_im;

const char deviceID[]   = "XXXXXXXX458541656289";
const char devicePasswd [] = "XXXXXXXXt";
const char gatewayId []  = "OPEN_TCP_001PTL001_100XXXXXXX";

void init_iotmakers()
{
      Client* client = g_shield.getClient();
      if ( client == NULL )    {
            Serial.println(F("No client from shield."));
            while(1);
      }

      g_im.init(deviceID, devicePasswd, gatewayId, *client);

// 생략
```

5.3 set_numdata_handler()

IoTMakers 에서 전송한 디바이스 제어데이터(숫자형)를 처리하기 위한 콜백함수를 등록한다.

void set_numdata_handler(IMCbTagidNumDataHndl cb_proc)		
param	cb_proc	사용자 정의 콜백함수(아래 코드예시 참조) 콜백함수에는 통상 디바이스 제어를 위한 코드를 작성
return	N/A	

코드예시)

```
void mycb_numdata_handler(char *tagid, double numval)
{
  Serial.print(tagid);Serial.print(F("="));Serial.println(numval);
}

void init_iotmakers()
{
	// 생략
	g_im.init(deviceID, authnRqtNo, extrSysID, *client);

	g_im.set_numdata_handler(mycb_numdata_handler);
	g_im.set_strdata_handler(mycb_strdata_handler);
	g_im.set_dataresp_handler(mycb_resp_handler);

	// 생략
}
```

5.4 set_strdata_handler()

IoTMakers 에서 전송한 디바이스 제어데이터(숫자형)를 처리하기 위한 콜백함수를 등록한다.

void im_set_strdata_handler(IMCbTagidNumDataHndl cb_proc)		
param	cb_proc	사용자 정의 콜백함수(아래 코드예시 참조) 콜백함수에는 통상 디바이스 제어를 위한 코드를 작성
return	N/A	

코드예시)

```
void mycb_strdata_handler(char *tagid, char *strval)
{
  Serial.print(tagid);Serial.print(F("="));Serial.println(strval);
}

void init_iotmakers()
{
       // 생략
       g_im.init(deviceID, authnRqtNo, extrSysID, *client);

       g_im.set_numdata_handler(mycb_numdata_handler);
       g_im.set_strdata_handler(mycb_strdata_handler);
       g_im.set_dataresp_handler(mycb_resp_handler);

       // 생략
}
```

5.5 set_dataresp_handler()

수집데이터 전송 후 응답 결과를 처리하기 위한 콜백함수를 등록한다.

void im_set_dataresp_handler(IMCbDataRespHndl cb_proc)		
param	cb_proc	사용자 정의 콜백함수(아래 코드예시 참조) 콜백함수 매개변수에는 트랜잭션아이디와 응답코드가 온다.
return	N/A	

코드예시)

```
void mycb_resp_handler(long long trxid, char *respCode)
{
      if ( strcmp(respCode, "100")==0 )
            Serial.println("resp:OK");
      else
            Serial.println("resp:Not OK");
}

void init_iotmakers()
{
      // 생략
      g_im.init(deviceID, authnRqtNo, extrSysID, *client);

      g_im.set_numdata_handler(mycb_numdata_handler);
      g_im.set_strdata_handler(mycb_strdata_handler);
      g_im.set_dataresp_handler(mycb_resp_handler);

      // 생략
}
```

5.6 connect()

IoTMakers 플랫폼에 접속을 시도한다.

int connect()		
param		
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
void init_iotmakers()
{
	// 생략
	g_im.init(deviceID, authnRqtNo, extrSysID, *client);
	// 생략

	// IoTMakers 서버 연결
	Serial.println(F("connect()..."));
	while ( g_im.connect() < 0 ){
		Serial.println(F("retrying."));
		delay(3000);
	}
	// 생략
}
```

5.7 disconnect()

IoTMakers 플랫폼의 접속을 해제한다.

int disconnect()		
param		
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
void init_iotmakers()
{
	// 생략

	// IoTMakers 서버 연결 해제
	Serial.println(F("disconnect()..."));
	g_im.disconnect();

	// 생략
}
```

5.8 auth_device()

IoTMakers 플랫폼에 디바이스 인증을 시도한다. 인증시 SDK 는 디바이스 아이디(devId), 디바이스패스워드(devPasswd), Gateway 연결 Id(gwId)를 플랫폼에 넘겨준다. 인증 성공시에는 현재 채널에 대한 인증번호(authNum)를 받아 내부적으로 처리한다..

int auth_device ()		
param		
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
void init_iotmakers()
{
	// 생략
	g_im.init(deviceID, authnRqtNo, extrSysID, *client);
	// 생략

	Serial.println(F("auth."));
	while ( g_im.auth_device() < 0 ) {
		Serial.println(F("fail"));
		while(1);
	}

	// 생략
}
```

5.9 loop()

IoTMakers 내부작업을 수행한다. 먼저 디바이스와 IoTMakers 플랫폼과의 연결유지를 위하여 Keepalive 메시지를 매 30 초간격으로 전송한다. 그리고, IoTMakers 플랫폼으로부터 전송된 제어 데이터가 있는지 확인한다. 제어데이터가 수신되었을 경우 사용자가 등록한 핸들러(콜백함수)를 호출한다. 수행할 작업이 없으면 즉시 리턴한다.

void loop()		
param	N/A	
return	N/A	

코드예시)

```
void loop()
{
	static unsigned long tick = millis();

	// 3초 주기로 무엇인가를 수행
	if ( ( millis() - tick) > 3000 )
	{
		// 생략
		tick = millis();
	}

	// IoTMakers 서버 수신처리 및 keepalive 송신
	g_im.loop();
}
```

5.10 send_numdata()

숫자형의 수집데이터를 태그스트림의 이름으로 전송한다. 태그스트림은 IoTMakers 포털에 등록된 디바이스의 상세정보란에서 정의한다.

int send_numdata(char *tagid, double val, long long trxid)		
param	tagid	해당 디바이스에 등록된 태그스트림이름
	val	태그스트림 값 반드시 double형을 사용해야 한다.
	trxid	이 데이터에 대한 트랜잭션아이디를 명시한다. 0값을 사용하면, sdk가 자동생성한다.
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
int send_light()
{
	int tmpVal = analogRead(A1);
```

```
        int light = tmpVal/4;

        Serial.print(F("Light : ")); Serial.println(light);
        if ( g_im.send_numdata("light", (double)light) < 0 ){
              Serial.println(F("fail"));
              return -1;
        }
        return 0;
}
```

5.11 send_strdata()

문자형의 수집데이터를 태그스트림의 이름으로 전송한다. 태그스트림은 IoTMakers 포털에 등록된 디바이스의 상세정보란에서 정의한다.

int send_strdata(char *tagid, char *val, long long trxid)		
param	tagid	해당 디바이스에 등록된 태그스트림이름
	val	태그스트림 값 반드시 '₩0'으로 끝나는 문자배열을 사용해야 한다.
	trxid	이 데이터에 대한 트랜잭션아이디를 명시한다. 0값을 사용하면, sdk가 자동생성한다.
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
int send_swich()
{

      Serial.print(F("Switch : ON"));
      if ( g_im.send_strdata("switch", "ON") < 0 ){
            Serial.println(F("fail"));
            return -1;
      }
      return 0;
}
```

6 IoTMakersDemo

6.1 IoTMakersDemo.ino 소스

아래 소스코드는 IoTMakers 플랫폼에 접속 후에 3 초 간격으로 태그스트림 데이터를 전송하는 예제이다.

deviceID, deviceId 그리고, devicePasswd 는 IoTMakers 포털에서 디바이스를 등록한 후에 디바이스 상세 정보 페이지에서 확인한다.

```
#include <Arduino.h>
#include <SPI.h>
#include <WiFi.h>

#include "IoTMakers.h"
#include "Shield_Wrapper.h"


/*
Arduino Shield
*/

Shield_Wrapper    g_shield;

#define SDCARD_CS 4
void sdcard_deselect()
{
      pinMode(SDCARD_CS, OUTPUT);
      digitalWrite(SDCARD_CS, HIGH); //Deselect the SD card
}
void init_shield()
{
      sdcard_deselect();

      const char* WIFI_SSID = "WIFI_SSID";
      const char* WIFI_PASS = "WIFI_PASS";
      g_shield.begin(WIFI_SSID, WIFI_PASS);

      g_shield.print();
}


/*
IoTMakers
*/
IoTMakers g_im;

const char deviceID[]   = "XXXXXXXX458541656289";
const char devicePasswd [] = "XXXXXXX2t";
const char gatewayId []  = "OPEN_TCP_001PTL001_1000XXXXXX";
```

```
void init_iotmakers()
{
	Client* client = g_shield.getClient();
	if ( client == NULL )	{
		Serial.println(F("No client from shield."));
		while(1);
	}

	g_im.init(deviceID, devicePasswd, gatewayId, *client);

	g_im.set_numdata_handler(mycb_numdata_handler);
	g_im.set_strdata_handler(mycb_strdata_handler);
	g_im.set_dataresp_handler(mycb_resp_handler);

	// IoTMakers 서버 연결
	Serial.println(F("connect()..."));
	while ( g_im.connect() < 0 ){
		Serial.println(F("retrying."));
		delay(3000);
	}

	// Auth
	Serial.println(F("auth."));
	while ( g_im.auth_device() < 0 ) {
		Serial.println(F("fail"));
		while(1);
	}

	Serial.print(F("FreeRAM="));Serial.println(g_im.getFreeRAM());
}

#define PIN_LED		9

void setup()
{
	Serial.begin(115200);
	while ( !Serial )  {
	  ;
	}

	// 9번 핀에 LED를 연결하여 작동상황을 파악
	pinMode(PIN_LED, OUTPUT);
	digitalWrite(PIN_LED, HIGH);

	// 와이파이 쉴드 초기화/접속/IP할당
	init_shield();

	// IoTMakers 객체 초기화/접속/장비인증
	init_iotmakers();

	digitalWrite(PIN_LED, LOW);
```

```
}

void loop()
{
	static unsigned long tick = millis();

	// 3초 주기로 센서 정보 송신
	if ( ( millis() - tick) > 3000 )
	{
		digitalWrite(PIN_LED, HIGH);
		send_temperature();
		send_light();
		digitalWrite(PIN_LED, LOW);

		tick = millis();
	}

	// IoTMakers 서버 수신처리 및 keepalive 송신
	g_im.loop();
}


// 아날로그 A0핀에 온도센서 연결
int send_temperature()
{
	int tmpVal = analogRead(A0);
	float voltage = (tmpVal/1024.0) * 5.0;
	float temperature = (voltage - .5) * 100;

	Serial.print(F("Temperature (c): "));
Serial.println(temperature);
	if ( g_im.send_numdata("temperature", (double)temperature) <
0 ) {
		Serial.println(F("fail"));
		return -1;
	}
	return 0;
}

// 아날로그 A1핀에 조도센서 연결
int send_light()
{
	int tmpVal = analogRead(A1);
	int light = tmpVal/4;

	Serial.print(F("Light : ")); Serial.println(light);
	if ( g_im.send_numdata("light", (double)light) < 0 ){
		Serial.println(F("fail"));
		return -1;
	}
	return 0;
}
```

```
void mycb_numdata_handler(char *tagid, double numval)
{
      // !!! USER CODE HERE
      //Serial.print(tagid);Serial.print(F("="));Serial.println(numv
al);
}

// "led" 태그의 값에 따라서 LED(9번핀)를 켜거나 끔
void mycb_strdata_handler(char *tagid, char *strval)
{
      // !!! USER CODE HERE
      //Serial.print(tagid);Serial.print(F("="));Serial.println(strv
al);

      if ( strcmp(tagid, "led")==0 && strcmp(strval, "on")==0 )
            digitalWrite(PIN_LED, HIGH);
      else if ( strcmp(tagid, "led")==0 && strcmp(strval,
"off")==0 )
            digitalWrite(PIN_LED, LOW);
}
void mycb_resp_handler(long long trxid, char *respCode)
{
      if ( strcmp(respCode, "100")==0 )
            Serial.println("resp:OK");
      else
            Serial.println("resp:Not OK");
}
```

6.2 Shield_Wrapper 샘플

시중에 유통중인 여러 종류의 네트워크 쉴드에 유연하게 대응하기 위하여 쉴드래퍼를 만들어 사용한다.

Shield_Wrapper.h

```
/*
  Shield_Wrapper.cpp - Library for wrapping Ethernet.
 Created by Kiucheol Shin(kiucheol.shin@kt.com), November 20, 2015.
  Released into the public domain.
*/

#ifndef SHIELD_WRAPPER_H
#define SHIELD_WRAPPER_H

#include <Arduino.h>
#include <Client.h>

class Shield_Wrapper
```

```
{
      public:
            Shield_Wrapper();
            void begin(const char* ssid, const char* pass);
            void begin(const byte* mac, const IPAddress ip);
            void disconnect();

            void print();
            Client* getClient();
      private:
            int _status;
};
#endif
```

Shield_Wrapper.cpp (무선 WiFi 쉴드래퍼)

```
/*
  Shield_Wrapper.cpp - Library for wrapping Ethernet.
 Created by Kiucheol Shin(kiucheol.shin@kt.com), November 20, 2015.
  Released into the public domain.
*/

#include <WiFi.h>
#include <SPI.h>
#include "Shield_Wrapper.h"

WiFiClient __client;

Shield_Wrapper::Shield_Wrapper(){
      _status = WL_IDLE_STATUS;
}

void Shield_Wrapper::begin(const char* ssid, const char* pass)
{
      Serial.println(F("begin WiFi!\n"));
      if (WiFi.status() == WL_NO_SHIELD) {
            Serial.println("WiFi shield not present");
            while(true);
      }

      while ( _status != WL_CONNECTED) {
            Serial.print(F("connect to "));Serial.println(ssid);
            // WPA/WAP2
            _status = WiFi.begin((char*)ssid, pass);
            delay(1000);
      }
}

void Shield_Wrapper::disconnect(){
      WiFi.disconnect();
}
```

```
void Shield_Wrapper::print(){
      Serial.print(F("IP: "));Serial.println(WiFi.localIP());
      Serial.print(F("GW: "));Serial.println(WiFi.gatewayIP());
}

Client* Shield_Wrapper::getClient(){
      return (Client*)&__client;
}
```

Shield_Wrapper.cpp (유선 Ethernet 쉴드래퍼)

```
/*
  Shield_Wrapper.cpp - Library for wrapping Ethernet.
 Created by Kiucheol Shin(kiucheol.shin@kt.com), November 20, 2015.
  Released into the public domain.
*/

#include <SPI.h>
#include <Ethernet.h>
#include "Shield_Wrapper.h"

EthernetClient __client;

Shield_Wrapper::Shield_Wrapper(){
      _status = 0;
}

void Shield_Wrapper::begin(const byte* mac, const IPAddress ip)
{
      Serial.println(F("begin Ethernet!\n"));
      // start the Ethernet connection:
      if (Ethernet.begin((uint8_t*)mac) == 0) {
            Serial.println(F("Failed to configure Ethernet using
DHCP"));
            // try to congifure using IP address instead of DHCP:
            Ethernet.begin((uint8_t*)mac, ip);
      }
}

void Shield_Wrapper::connect()
{
}

void Shield_Wrapper::disconnect()
{
}

void Shield_Wrapper::print(){
      Serial.print(F("IP: "));Serial.println(Ethernet.localIP());
      Serial.print(F("GW: "));Serial.println(Ethernet.gatewayIP());
}
```

```
Client* Shield_Wrapper::getClient(){
      return (Client*)&__client;
}
```

7 부록

7.1 응답코드

수집데이터 전송에 대한 응답코드 목록이다.

코드	내용	비고
100	Success	처리성공
200	GeneralError	일반 오류
201	ImplementationError	구현 오류
202	PacketPushError	패킷푸쉬 오류
203	DecryptionError	복호화 오류
204	ParsingError	패킷파싱 오류
205	AuthenticationError	인증 오류
206	AckError	응답 오류
207	CommChAthnError	통신채널 인증오류
208	RequestInfoError	요청정보 오류