

## Chapter 270. 이더넷 - 오늘의 날씨

---

이전 장에서 이더넷 쉴드를 사용하여 아두이노를 웹 클라이언트와 웹 서버로 사용하는 방법에 대해 알아보았다. 웹 클라이언트로 사용하는 경우 특정 웹 사이트에 접속하여 정보를 읽어오는 것이 가능하기는 하지만 HTML(Hyper Text Markup Language)로 표현되는 웹 페이지는 데이터 이외의 정보들이 많이 포함되어 있어 아두이노로는 그 내용을 알아내기가 쉽지 않다. 하지만 일부 웹 서버의 경우 공개 API(Open API)를 통해 정보를 쉽게 알아낼 수 있는 방법을 제공하고 있다. API는 Application Programming Interface의 약어로 컴퓨터 프로그래밍에서 프로그램 작성을 쉽게 할 수 있도록 제공되는 함수와 도구들을 가리키는 말이다. 이러한 API를 웹을 통해 확장한 것을 공개 API 또는 웹 API라고 부르며, 웹 서비스를 제공하는 회사에서 회사의 특정 웹 서비스에 접근, 데이터 요청 및 획득 등의 방법을 API 형식으로 공개하여 사용자가 쉽게 정보를 얻을 수 있도록 만들어져 있다. 일반적인 웹 페이지는 실제로 필요한 정보 이외에도 페이지 레이아웃 등을 위해 많은 정보가 추가되어 있는 반면 공개 API를 통해 제공되는 내용에는 실제 정보만이 포함되어 있어 해석하고 사용하기 편리하다. 이 장에서는 날씨 정보와 관련된 공개 API를 제공하고 있는 [openweathermap.org](http://openweathermap.org)를 통해 현재 날씨를 알아내는 방법에 대해 알아본다.

### 1. 공개 API - 날씨 정보

먼저 [openweathermap.org](http://openweathermap.org)에 접속해서 현재 날씨를 알고 싶은 도시를 입력해 보자.

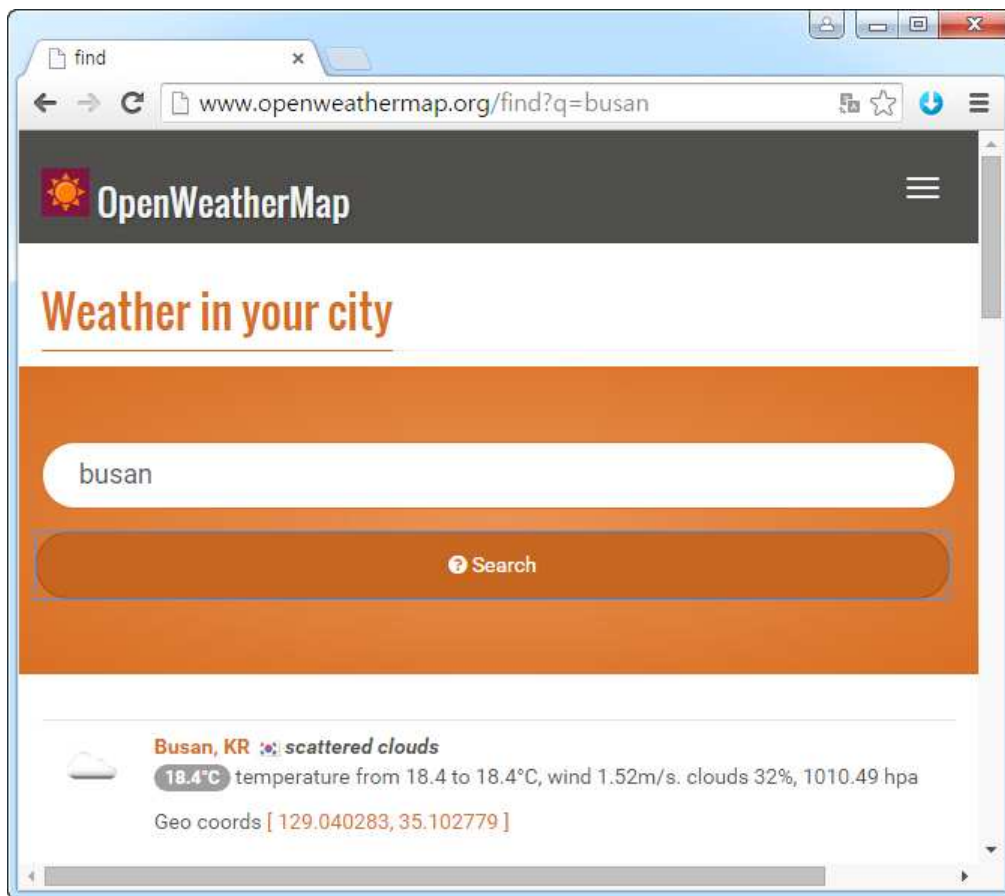


그림 1. openweathermap.org에서 제공하는 부산의 현재 날씨 정보

'busan'을 입력하였을 때 아래쪽에 현재 날씨와 위치 정보가 표시되고 있음을 볼 수 있다. 웹 페이지를 'find.html'이라는 이름으로 저장하고 텍스트 편집기로 내용을 살펴보자. 20KByte를 넘는 문서에는 다양한 내용이 포함되어 있지만 우리가 알고 싶은 내용은 그 중 극히 일부일 뿐이다. 우리가 원하는 내용만을 얻어올 수 있는 방법이 없을까? 공개 API를 사용하는 것이 해결책 중 하나이며 openweathermap.org에서는 날씨 정보를 위한 공개 API를 제공하고 있다. 날씨 정보는 도시 이름을 사용하는 방법, 도시 ID를 사용하는 방법, 도시의 위도와 경도를 사용하는 방법 등 다양한 방법을 통해 제공되고 있다. 이 중 도시 이름을 이용하여 날씨 정보를 알아내기 위해서는 주소 창에 그림 2와 같이 입력하면 된다.

```
http://api.openweathermap.org/data/2.5/weather?q="busan"  
http://api.openweathermap.org/data/2.5/weather?q="busan"&mode=xml
```

그림 2. 공개 API 사용 형식

그림 2에서 알 수 있듯이 모드를 지정하지 않으면 JSON(JavaScript Object Notation) 형식의 데이터를 얻을 수 있으며, XML 형식의 데이터를 얻기 위해서는 'mode=xml'을 추가해 주어야 한다. XML 형식으로 반환되는 날씨 정보는 그림 3과 같다.



그림 3. XML 형식의 날씨 정보<sup>1)</sup>

어딘가 익숙하지 않은가? XML은 eXtended Markup Language의 약어로 구조화된 정보 표현을 위해 널리 사용되는 방식으로 HTML 역시 XML의 한 종류이다. XML에서는 계층적으로 정보를 구조화해서 나타내며 각 항목을 태그를 사용하여 표현한다. 그림 3를 살펴보면 날씨 정보가 2가지 방식으로 표현되고 있음을 알 수 있다. 첫 번째는 여는 태그와 닫는 태그의 쌍으로 표시되는 경우이고, 두 번째는 하나의 태그로만 표시되는 경우이다. 정보의 계층 구조에 따라 첫 번째 방식은 상위 정보를 나타내기 위해, 두 번째 방식은 하위 정보를 나타내기 위해 사용되고 있다. 그림 4는 태그 표시 방법을 단순화하여 나타낸 것이다.

1) 브라우저의 종류에 따라 표시되는 방법이 약간씩 다를 수 있다.

|                        |                 |
|------------------------|-----------------|
| <wind>                 | ← 상위 정보 (여는 태그) |
| <speed value="1.52" /> | ← 하위 정보         |
| </wind>                | ← 상위 정보 (닫는 태그) |

그림 4. 태그 표시 방법

이 장에서는 XML 형식의 날씨 정보를 얻어오고 이를 해석(parsing)하여 현재 날씨 정보를 표시하도록 한다.

## 2. XML 데이터 얻기

현재 부산의 날씨를 XML 형식으로 얻어 시리얼 모니터로 출력해 보자. 스케치 1은 이전 장에서 웹 클라이언트로 사용한 코드와 거의 동일하며, 반환되어 오는 정보에서 차이가 있을 뿐이다.

### 스케치 1. XML 형식의 날씨 정보 얻기

```
#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetClient.h>
#include <EthernetServer.h>
#include <EthernetUdp.h>
#include <SPI.h>

byte MAC[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xD2, 0x16 };
IPAddress IP(192, 168, 0, 10);
EthernetClient client;
char server_address[] = "api.openweathermap.org";
char server_page[] = "/data/2.5/weather?q=\"busan\"&mode=xml";

void setup(){
    Serial.begin(9600);
```

```
Ethernet.begin(MAC, IP);           // 고정 IP 사용
Serial.print("Local IP : ");
Serial.println(Ethernet.localIP());
delay(2000);                       // 이더넷 쉘드의 초기화 대기

Serial.println("* Connecting to server...");
// 날씨 정보 제공 서버에 접속
if(client.connect(server_address, 80)){
    Serial.println("* Connected to server...");

    // 부산의 현재 날씨 정보 요청
    client.println(String("GET ") + server_page);
    client.println("Connection: close");    // 데이터 수신 종료 후 접속 종료
    client.println();                       // 요청 종료 알림
}
else{
    Serial.println("* Connection to server failed...");
}
}

void loop()
{
    // 웹 서버로부터 수신한 데이터가 존재하는 경우
    if (client.available()) {
        char c = client.read();           // 수신 데이터를 읽기
        Serial.print(c);                 // 시리얼 모니터로 수신 데이터 출력
    }

    // 서버와의 접속이 종료된 경우 클라이언트 종료
    if (!client.connected()) {
        Serial.println();
        Serial.println("* Disconnecting...");
        client.stop();                   // 클라이언트 종료

        while(true);
    }
}
```

```
}
```

그림 5은 스케치 1의 실행 결과로 그림 3과 동일한 내용이 수신되고 있음을 확인할 수 있다.

The screenshot shows the 'COM35 (Arduino Uno)' serial monitor window. The text displayed is an XML document representing weather data for Busan. The XML structure is as follows:

```
<current>
  <city id="0" name="Busan">
    <coord lon="129.08" lat="35.18"/>
    <country>South Korea</country>
    <sun rise="2015-04-17T20:47:41" set="2015-04-18T09:58:26"/>
  </city>
  <temperature value="288.453" min="288.453" max="288.453" unit="kelvin"/>
  <humidity value="47" unit="%"/>
  <pressure value="1009.99" unit="hPa"/>
  <wind>
    <speed value="1.08" name="Calm"/>
    <direction value="180.003" code="S" name="South"/>
  </wind>
  <clouds value="76" name="broken clouds"/>
  <visibility/>
  <precipitation mode="no"/>
  <weather number="803" value="broken clouds" icon="04d"/>
  <lastupdate value="2015-04-18T08:09:16"/>
</current>
```

At the bottom of the window, the settings are: '자동 스크롤' (checked), 'No line ending', and '9600 보드 레이트'.

그림 5. 스케치 1 실행 결과

그림 3과 그림 5를 살펴보면 각 항목은 한 줄에 표현되고 있음을 확인할 수 있다. XML은 텍스트 기반으로 각 행에 하나의 항목을 표시하고 각 행의 끝에는 줄바꿈 문자인 '\n'(LF, Line Feed)가 추가되어 있다.

### 3. XML 데이터 해석

스케치 1은 현재 날씨 정보를 XML 형식의 데이터로 얻어올 수 있도록 해준다. 이 중 온도

(temperature), 습도(humidity), 풍속(speed)을 찾아내어 출력해 보자. 먼저 수신되는 데이터를 버퍼에 저장하고 '\n' 문자를 만나면 이를 처리하도록 한다. 각 항목에 해당하는 단어가 발견되면 'value' 값을 찾아내어 출력하면 된다. 스케치 2는 수신되는 데이터에서 각 항목을 찾아내어 출력하는 예이다.

#### 스케치 2. XML 데이터 해석

```
#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetClient.h>
#include <EthernetServer.h>
#include <EthernetUdp.h>
#include <SPI.h>

byte MAC[] = { 0x90, 0xA2, 0xDA, 0x0D, 0xD2, 0x16 };
IPAddress IP(192, 168, 0, 10);
EthernetClient client;
char server_address[] = "api.openweathermap.org";
char server_page[] = "/data/2.5/weather?q=\"busan\"&mode=xml";

String buffer = ""; // 수신 데이터 저장 버퍼
boolean process_it = false;

void setup(){
  Serial.begin(9600);

  Ethernet.begin(MAC, IP); // 고정 IP 사용
  Serial.print("Local IP : ");
  Serial.println(Ethernet.localIP());
  delay(2000); // 이더넷 설드의 초기화 대기

  Serial.println("* Connecting to server...");
  // 날씨 정보 제공 서버에 접속
  if(client.connect(server_address, 80)){
    Serial.println("* Connected to server...");
```

```
// 부산의 현재 날씨 정보 요청
client.println(String("GET ") + server_page);
client.println("Connection: close");    // 데이터 수신 종료 후 접속 종료
client.println();                       // 요청 종료 알림
}
else{
    Serial.println("* Connection to server failed...");
}
}

void loop()
{
    // 웹 서버로부터 수신한 데이터가 존재하는 경우
    if (client.available()) {
        char c = client.read();          // 수신 데이터를 읽기
        if(c == '\n')
            process_it = true;           // 한 줄 수신 종료
        else
            buffer = buffer + c;
    }

    if(process_it) process_data();        // 수신 데이터 처리

    // 서버와의 접속이 종료된 경우 클라이언트 종료
    if (!client.connected()) {
        Serial.println();
        Serial.println("* Disconnecting...");
        client.stop();                  // 클라이언트 종료

        while(true);
    }
}

void process_data(void)
{

```

```
String key[] = { "temperature", "humidity", "speed" };
String value_string = "";
int index, position, position_end;

// 추출하고자 하는 정보의 키워드 검색
for(index = 0; index < 3; index++){
    position = buffer.indexOf(key[index]);
    if(position != -1) break;
}

// 온도, 습도, 풍속 정보가 포함되어 있는 경우
if(index < 3){
    position = buffer.indexOf("value") + 7; // 값 시작 위치
    position_end = -1;
    for(int i = position; i < buffer.length(); i++){
        if(buffer[i] == '\r'){
            position_end = i;           // 값 종료 위치
            break;
        }
    }

    if(position_end != -1){
        value_string = buffer.substring(position, position_end);
        Serial.println(">> " + buffer);           // 수신된 문자열
        Serial.println(key[index] + " : " + value_string); // 추출된 내용
    }
}

buffer = "";
process_it = false;
}
```

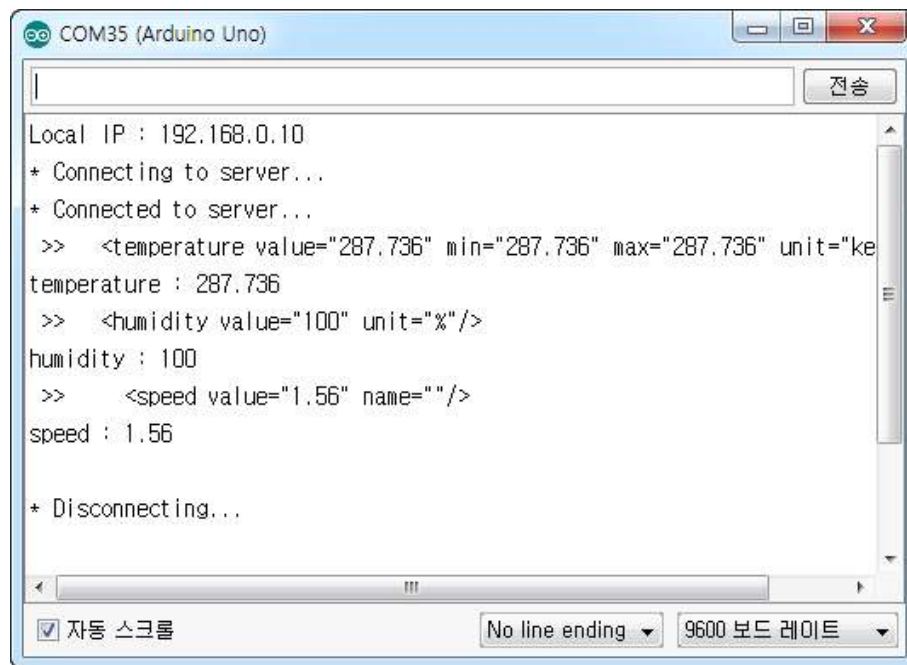


그림 6. 스케치 2 실행 결과