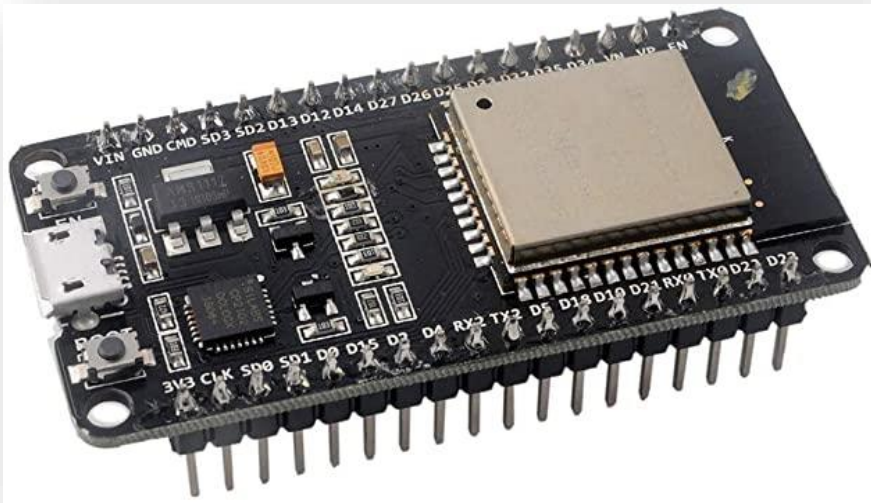


# Using the ESP32 and REST APIs

Making sense of IOTs (Internet of Things)



# Quick Links

Click on Icon to go directly to the desired section:

## IOT - Using an API

In this example we will now introduce you to an application of this using the ESP32 as an Internet connected device (IoT - Internet of Things).

API's (Application Programming Interface) are used throughout the Internet to provide data sets that can be used to inform, guide or entertain.

For example, there are APIs for Bitcoin, Weather and Cat Facts for example.

Let's take a moment and explore a simple example.

**Cat Facts:**

This API provides a short random fact about cats. Our application will query the API and send the response to our serial monitor for display.



## Preparing our ESP32 IOT Code

With our code snippet in hand, we are ready to begin the process of preparing our first IOT app.

Open the following Arduino program into your Arduino IDE setup for the ESP32

"api\_activity\_template.ino"

With this program loaded we will need a few bits of information to complete the process:

- Your wireless SSID - for this example we are using the ESP32 that has only WiFi and Bluetooth
- Your wireless SSID Password
- The URL of the API response site use: <https://catfact.ninja/fact>



## Serialization/Deserialization

- When sending data to and from over the Internet we often have data streams that contain a variety of forms, text, numbers, images, etc. In the course of this we would like to send the data with integrity, efficiently and to preserve the structure/order of its original form.
- Serialization is a way that we can do this. Simply we break the data down into smaller parts, register the structure and then send this data along.
- Deserialization receives the data with its schema that describes how the data is sent as well as its structure and restores all the values to their original form.
- Our API data is serialized when we receive it and hence the structure that we see.



- Now we need to deserialize the data stream restore the original structure and data types.

## HTTPClient to connect to the Internet

- 1 This section
  - Checks for internet to run new page requests
- 2 HTTPClient
  - http.begin() (URL of IOT API):
    - Note the following into this field:
      - <https://catfact.ninja/fact>
      - The statement will look like this when complete:  
`http.begin("https://catfact.ninja/fact");`
  - Check to see if page request is successful
    - If successful put data into String variable call "output"
    - If connection fail after user



## ArduinoJson

Fortunately, we have a resource tool that we can use to do this for us.

[ArduinoJson](https://arduinojson.org/)

We will use this site and its tools to deserialize our API response and use it to create an IOT device that will report real time data from the internet.

For our purposes we will use the [ArduinoJson](https://arduinojson.org/) Assistant.

Note the following settings:

- Processor - ESP32
- Mode - Deserialise
- Input type - String



## Displaying your results

- 1 Looking to the right you can see a connected section // json code goes here... please the code from the [ArduinoJson](https://arduinojson.org/) Assistant that was generated from our API reflect investigation
- 2 `StaticJsonDocument<128>` due, is the object that contains the data
- 3 The data that we can use has been parsed as:
  - `const char* fact`
  - `int length`
- 4 We are now ready to send the data to the serial monitor



# IOT – Using an API

In this example we will now introduce you to an application of the using the ESP32 as an Internet connected device (IOT – Internet of Things).

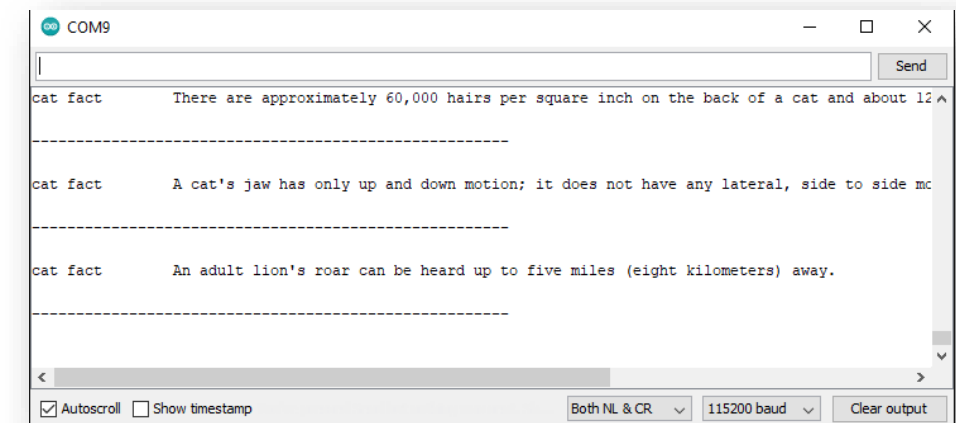
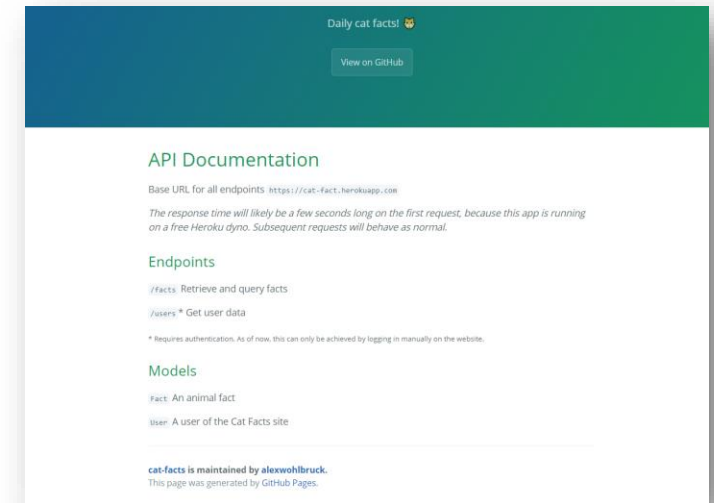
API's (Application Programming Interface) are used throughout the internet to provide data sets that can be used to inform, guide or entertain.

For example, there are APIs for Bitcoin, Weather and Cat Facts for example.

Let's take a moment and explore a simple example.

Cat Facts:

This API provides a short random fact about cats. Our application will query the API and send the response to our serial monitor for display.

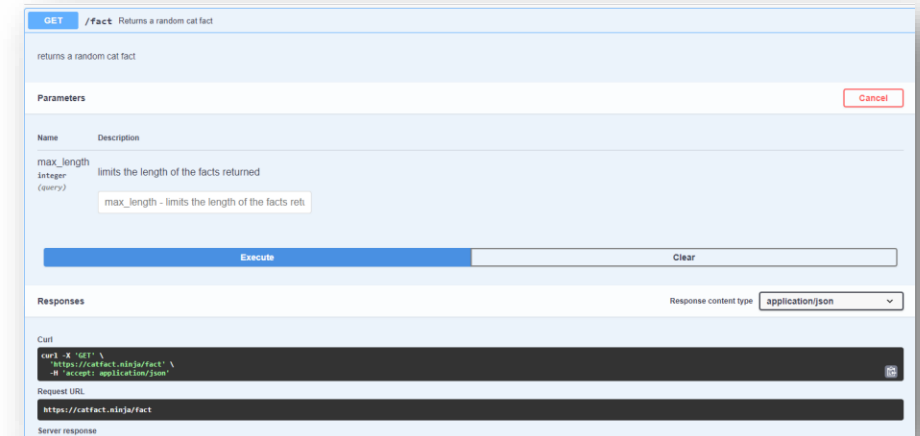
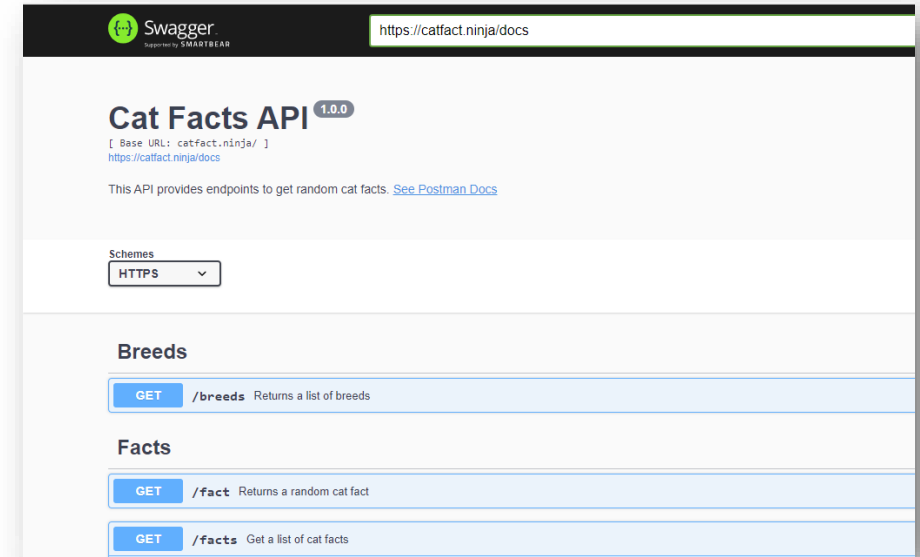
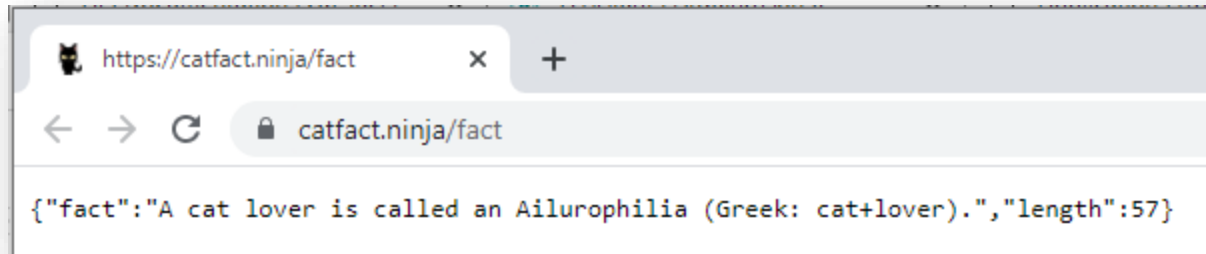


# What do we need?

- First, we want to make sure that our API has some documentation.
- We need to do an internet search and find a web site that looks interesting and clean.

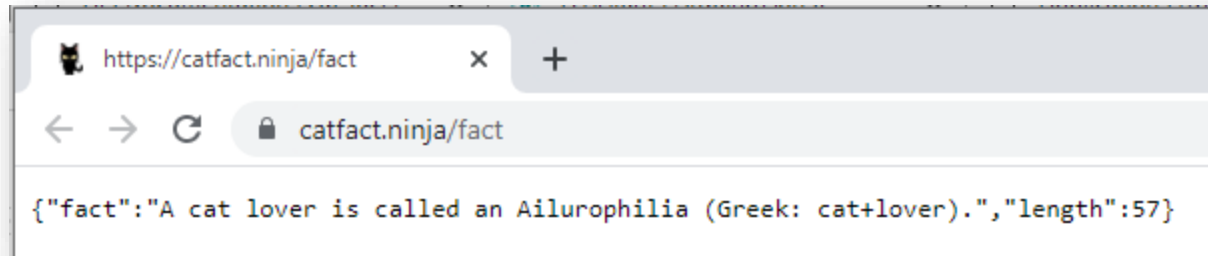
<https://catfact.ninja/docs>

- Then we look for information on the URL (Universal Resource Locator) which will tell us the address to post in our browser for a response. In this case we are given a general URL – <https://catfact.ninja/fact>
- Now that we have this, we can paste this URL in our browser and see the result.



# Seems simple enough, well err...

- Well, this seems simple enough but if we look further, we can see that there is a form to the response.

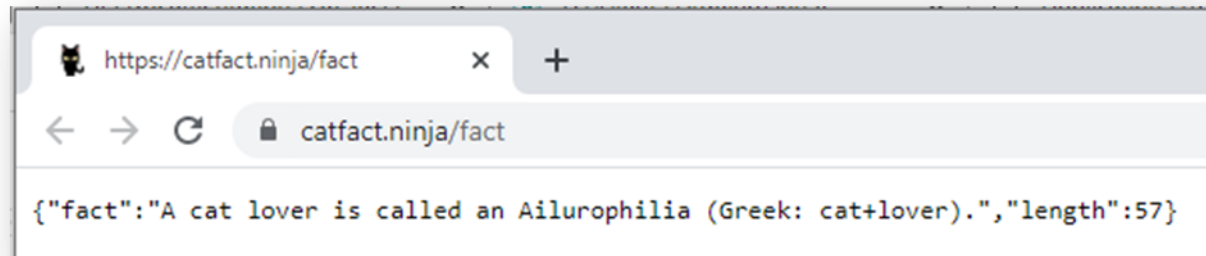


A screenshot of a web browser window. The address bar shows the URL `https://catfact.ninja/fact`. Below the address bar, the page content displays a JSON object: `{"fact": "A cat lover is called an Ailurophilia (Greek: cat+lover).", "length": 57}`.

- `{"fact": "A cat lover is called and Ailurophilia (Greek: cat+lover).", "length": 57}`, so notice we have a field name, `: "fact"`, and `"length"` followed `:` by text and `:` a number. Given this simple example we might be able to convert this to a string – `fact$`, and `length` as an integer. However, this means that in every example API response we will need see the return format and to formulate how to decode the information.
- We do an easier way to approach this. [JSON](#) – JavaScript Object Notation. In fact, due to this problem as well as many other similar instances where we want to send data over the internet there are tools sets designed to capture this data and convert into an easier form of `char`, `int`, `float`, etc. types that we can use in our C++ program.

# Serialization/Deserialization

- When sending data to and from over the internet we often have data streams that contain a variety of forms, text, numbers, images, etc. In the course of this we would like to send the data with integrity, efficiently and to preserve the structure/order of its original form.
- Serialization is a way that we can do this. Simply we break the data down into smaller parts, register the structure and then send this data along.
- Deserialization receives the data with a schema that describes how the data is sent as well as its structure and restores all the values to their original form.
- Our API data is serialized when we receive it and hence the structure that we see.



- Now we need to deserialize the data stream restore the original structure and data types.

# ArduinoJson

Fortunately, we have a resource tool that we can use to do this for us.

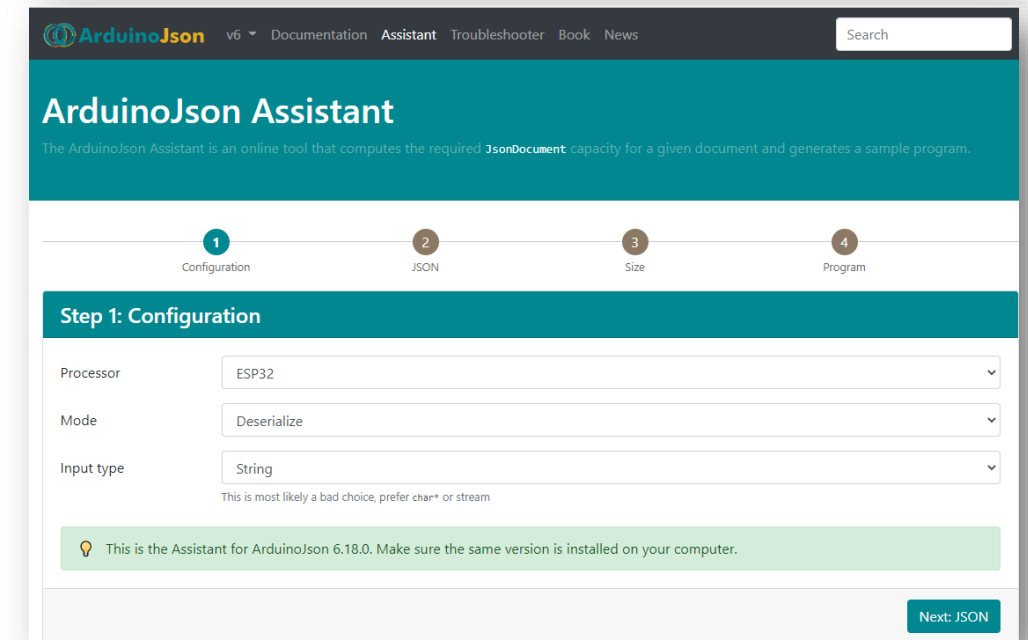
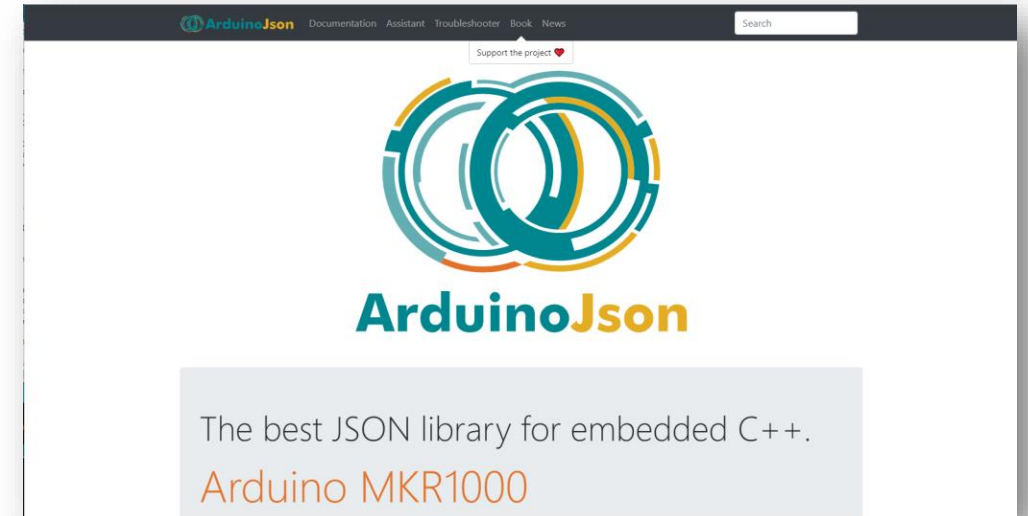
[ArduinoJson](https://arduinojson.org)

We will use this site and its tools to deserialize our API response and use it to create an IOT device that will report real time data from the internet.

For our purposes we will use the [ArduinoJson](https://arduinojson.org) Assistant.

Note the following settings:

- Processor – ESP32
- Mode – Deserialize
- Input Type - String





# ArduinoJson

Next, we will hit the “next: JSON” button in the lower right corner of the page.

Here we will page our response from the API so for instance:

```
{"fact": "The first cat show was in 1871 at the Crystal Palace in London.", "length": 63}
```

Here we will page our response from the API so for instance:

### Step 2: JSON

Examples: [OpenWeatherMap](#), [Reddit](#)

**Input**

```
{"fact": "The first cat show was in 1871 at the Crystal Palace in London.", "length": 63}
```

Input length: 86

ArduinoJson v6 Documentation Assistant Troubleshooter Book News

Search

## ArduinoJson Assistant

The ArduinoJson Assistant is an online tool that computes the required `JsonDocument` capacity for a given document and generates a sample program.

1 Configuration

2 JSON

3 Size

4 Program

### Step 1: Configuration

Processor

Mode

Input type   
This is most likely a bad choice, prefer `char*` or `stream`

This is the Assistant for ArduinoJson 6.18.0. Make sure the same version is installed on your computer.

Next: JSON

Next: JSON



# ArduinoJson

Notably if you want you can see the structure by pressing on Prettify, this will restructure the stream into it's intended data structure.

Next press “[Next: Size](#)” in the lower Right Corner.

This will bring us to a general summary of the amount data and type of structures detected in the stream.

Next press “[Next: Program](#)” in the lower right-hand portion of the page.

This will leave us with the program snippet that we will use to convert our API response into workable code.

Using the Copy button copy and store this into a text file.

### Step 2: JSON

Examples: [OpenWeatherMap](#), [Reddit](#)

Input

```
{
  "fact": "The first cat show was in 1871 at the Crystal Palace in London.",
  "length": 63
}
```

Input length: 95

[Previous](#) [Prettify](#) [Next: Size](#)

### Step 3: Size

Data structures	32	Bytes needed to stores the JSON objects and arrays in memory <a href="#">1</a>
Strings	76	Bytes needed to stores the strings in memory <a href="#">1</a>
Total (minimum)	108	Minimum capacity for the JsonDocument.
Total (recommended)	128	Including some slack in case the strings change, and rounded to a power of two

► Tweaks (advanced users only)

[Previous](#) [Next: Program](#)

### Step 4: Program

```
// String input;

StaticJsonDocument<128> doc;

DeserializationError error = deserializeJson(doc, input);

if (error) {
  Serial.print(F("deserializeJson() failed: "));
  Serial.println(error.f_str());
  return;
}

const char* fact = doc["fact"]; // "The first cat show was in 1871 at the Crystal Palace in London."
int length = doc["length"]; // 63
```

[Copy](#)

See also [Deserialization Tutorial](#) [DynamicJsonDocument](#) [deserializeJson\(\)](#)

# Preparing our ESP32 IOT Code

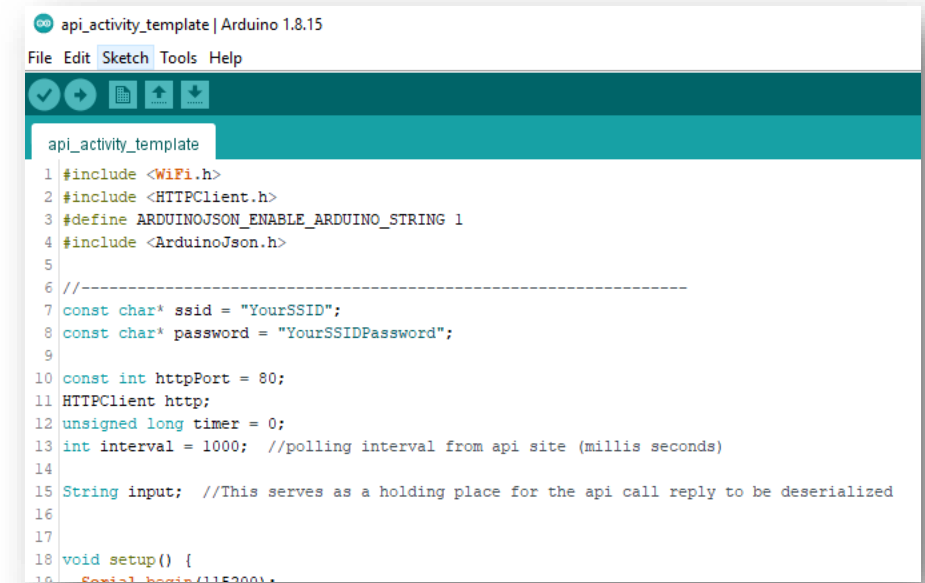
With our code snippet in hand, we are ready to begin the process of preparing our first IOT app.

Open the following Arduino program into your Arduino IDE setup for the ESP32

“api\_activity\_template.ino”

With this program loaded we will need a few bits of information to complete the process:

- Your wireless SSID – for this example we are using the ESP32 that has only WiFi and Bluetooth
- Your wireless SSID Password
- The URL of the API response site  
use: <https://catfact.ninja/fact>



```
api_activity_template | Arduino 1.8.15
File Edit Sketch Tools Help

api_activity_template
1 #include <WiFi.h>
2 #include <HTTPClient.h>
3 #define ARDUINOJSON_ENABLE_ARDUINO_STRING 1
4 #include <ArduinoJson.h>
5
6 //-----
7 const char* ssid = "YourSSID";
8 const char* password = "YourSSIDPassword";
9
10 const int httpPort = 80;
11 HTTPClient http;
12 unsigned long timer = 0;
13 int interval = 1000; //polling interval from api site (millis seconds)
14
15 String input; //This serves as a holding place for the api call reply to be deserialized
16
17
18 void setup() {
19   Serial.begin(115200);
```

# Setting up your Wireless Network

Notice on the right our `#include` statements:

- `<WiFi.h>` – this library connects us to your wifi network.
- `<HTTP Client.h>` - we need this library to access web pages on the internet.
- `ARDUINOJSON_ENABLE_ARDUINO_STRING 1` and `<ArduinoJson.h>` provides the library to deserialize the API response the to setup the types and data structure to use the response.
- Replace the “YourSSID” with your networks SSID
- Replace the “YourSSIDPassword” with your networks access password.

```
api_activity_template
1 #include <WiFi.h>
2 #include <HTTPClient.h>
3 #define ARDUINOJSON_ENABLE_ARDUINO_STRING 1
4 #include <ArduinoJson.h>
5
6 //-----
7 const char* ssid = "YourSSID";
8 const char* password = "YourSSIDPassword";
9
10 const int httpPort = 80;
11 HTTPClient http;
12 unsigned long timer = 0;
13 int interval = 1000; //polling interval from api site (millis seconds)
```

## **CAUTION!**

This will be available to anyone that uses this program...

# Decoding the Code

## ① This section

- Sets the HTTP port 80 (default)
- Create a task timer (timer) to schedule new page requests
- Sets the interval (interval) for the page requests (seconds)

## ② Void setup()

- Begin Serial Monitor
- Begin WiFi Session
- Check WiFi for successful connection

```
10 const int httpPort = 80;
11
12 unsigned long timer = 0;
13 int interval = 1000; //polling interval from api site (millis seconds)
14
15 String input; //This serves as a holding place for the api call reply to be deser
16
17
18 void setup() {
19     Serial.begin(115200);
20     WiFi.begin(ssid, password);
21     Serial.println("Connecting");
22     while(WiFi.status() != WL_CONNECTED)
23     {
24         delay(500);
25         Serial.print(".");
26     }
27     Serial.println("");
28     Serial.print("Connected to WiFi network with IP Address: ");
29     Serial.println(WiFi.localIP());
30
31     Serial.println("Timer set to 10 seconds (timerDelay variable), it will take 10 s
32 }
33
```

# HttpClient to connect to the Internet

## ① This section

- Checks for interval to run new page requests

## ② HttpClient

- `http.begin("URL of IOT API");`
  - Paste the following into this field
  - <https://catfact.ninja/fact>
  - The statement will look like this when complete:

`http.begin(https://catfact.ninja/fact);`

- Check to see if page request is successful
  - If successful put data into String variable call "input"
- If connection fail alter user

```
34 void loop()
35 {
36   if ((millis() - timer) >= interval)
37   {
38     timer = millis();
39
40     HttpClient http;
41
42     http.begin("URL of IOT API");
43     int httpCode = http.GET();
44
45     if (httpCode > 0)
46     { //Check for the returning code
47       input = http.getString();
48     }
49     else
50     {
51       Serial.println("Error on HTTP request");
52     }
53     http.end();
54 }
```

# Deserialization

- ① Looking to the right you can see a commented section //Json code goes here... place the code from the **ArduinoJson** Assistance that was generated from our API catfact investigation
- ② This is an example of the code that will be inserted.

```
53 http.end();
54
55
56 //This section comes from ArdinoJson Assistant
57 //Step 2 JSON page check for the returning code https://arduinojson.org/v6/assistant/# cut and paste return code from api call
58
59 //This is generated by the ArduinoJson Assistant
60 //Structure is contained within the objects that are created
61 //They do not have to used may be comment out or deleted since
62 //members are committed to memory StaticJsonDocument<1024> doc
63
64 //Json code goes here....
65
66 StaticJsonDocument<128> doc;
67
68 DeserializationError error = deserializeJson(doc, input);
69
70
71 if (error) {
72   Serial.print(F("deserializeJson() failed: "));
73   Serial.println(error.f_str());
74   return;
75 }
76
77 const char* fact = doc["fact"]; // "The first cat show was in 1871 at the Crystal Palace in London."
78 int length = doc["length"]; // 63
```

# Displaying your results

- ① Looking to the right you can see a commented section //Json code goes here... place the code from the **ArduinoJson** Assistance that was generated from our API catfact investigation
- ② StaticJsonDocument<128> doc, is the object that contains the data.
- ③ The data that we can use has been posted as:
  - `const char* fact`
  - `int length`
- ④ We are now ready to send the data to the serial monitor

```
53 http.end();
54
55
56 //This section comes from ArdinoJson Assistant
57 //Step 2 JSON page check for the returning code https://arduinojson.org/v6/assistant/# cut and paste
58
59 //This is generated by the ArduinoJson Assistant
60 //Structure is contained within the objects that are created
61 //They do not have to used may be comment out or deleted since
62 //members are committed to memory StaticJsonDocument<1024> doc
63
64 StaticJsonDocument<128> doc;
65
66 DeserializationError error = deserializeJson(doc, input);
67
68 if (error) {
69   Serial.print(F("deserializeJson() failed: "));
70   Serial.println(error.f_str());
71   return;
72 }
73
74 const char* fact = doc["fact"]; // "The first cat show was in 1871 at the Crystal Palace in London."
75 int length = doc["length"]; // 63
76
77 //Variables are temporary and have persistence as long as this sample
78 //cycle lasts, consider what needs or is being done with the variables
79 Serial.print("cat fact");
80 Serial.print("\t");
81 Serial.println(fact);
82 Serial.println();
83 Serial.println("-----");
84 Serial.println();
85 }
```



# Enjoy Using the ESP32 and APIs

