

Pool Proof Attendance System with Fingerprint and Sensors based on IOT

Submitted in Partial fulfilment of the Requirements for the Award of
Bachelor of Engineering in Electronics and Communication Engineering.

By

Bodapati Nanda Kishore (38130033)

Diddi Sai Sri Ram (38130056)



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

May 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING
BONAFIDE CERTIFICATE**

This is to certify that Project Report is the bonafide work of B. Nanda Kishore, D Sai Sriram who carried out the Project entitled **"Pool Proof Attendance System with Fingerprint and Sensors based on IOT"**. Under the guidance of **Mrs. F.V. Jaya Sudha**, during the period JAN 2022 to MARCH 2022.

INTERNAL GUIDE
DEPARTMENT

Mrs.F. V Jaya Sudha

HEAD OF THE

Dr.T. RAVI, M.E., Ph.D.,

Submitted for Viva Voce held on

Internal Examiner

External Examiner

DECLARATION

We (B. Nandakishore and D. Sai Sriram) Hereby declare that the Project report entitled **POOL PROOF ATTENDANCE SYSTEM WITH FINGERPRINT AND SENSORS BASED ON IOT** is done under the guidance of **Mrs.F.V. Jaya Sudha** is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in **ELECTRONICS AND COMMUNICATION ENGINEERING.**

DATE:

PLACE: CHENNAI

Signature 1

Signature 2

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to board management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY for their kind encouragement in doing this internship and for completing it successfully. I am grateful to them.

We convey our thanks to **Dr. N.M. NANDHITHA, M.E., Ph.D.**, Dean, school of Electrical and Electronics engineering and **Dr. T. RAVI, M.E., Ph.D.**, Head of the Department, Electronics and Communication engineering for providing me necessary support and details at the right time during the progressive reviews.

We would like to express my sincere and deep sense of gratitude to our Project Guide **Mrs.F.V. Jaya Sudha** for her valuable guidance, suggestions and constant encouragement paved the way for the successful completion of my Project.

We wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Electronics and Communication Engineering who were helpful in many ways for the completion of the project.

Abstract:

The proposed research work defined for the acknowledgement of a biometric based attendance system for institution faculty, as well as its significance. We mainly emphasize the problem the faculty face due to the lack of any existing system which does not use any alert or notification mechanism like sending an SMS to the faculty's mobile phone to prompt them to mark their attendance. The essential purpose of this system is to create automated notifications based on the biometric fingerprint authentication for attendance. The system avoids attendance oriented discrepancies and overheads on a central repository.

Introduction:

Real-time online attendance method is helpful for workers who do a lot of activities outside the office or workers with multi-schedule. The attendance system using an online biometric fingerprint system will reduce the problems caused by manual system usage such as lags in data management. The purpose of the research is to construct an online presence system that combines fingerprint modules and GPS.

The ZFM-20 fingerprint module is used as the system's main input as well as a security tool as an entrance to get access to the entire system. A GPS module is applied to determine the user's location and sends it to the smartphone. The Arduino module in the system will send a text message to the parties concerned about the user's location data automatically. Each module works well and testing the entire system showed the system worked reliably according to the initial scenario.

The User can access the report using SMS, website, and application on the Android smartphone. The fingerprint sensor can determine the fingerprint stored in the database with an average response time of 1.39 seconds, and GPS can determine latitude and longitude with an average error of 0.007352% and 0.0003% respectively.

Literature survey:

Title: Portable Biometric Attendance System Using IOT

Author: Sebastian Chennattu; Aditya Kelkar; Aaron Anthony; Sushma Nagdeote.

Content: The identified fingerprint ID is then stored in memory and is made ready for transfer. The data is then sent to the database on a web server where the Database Management System (DBMS) updates the attendance tables accordingly. This database can be downloaded or viewed by the respective faculty.

Title: Fingerprint attendance system for classroom needs

Author: Basheer K P Mohamed; C V Raghu

Content: Fingerprint attendance system aims to automate the attendance procedure of an educational institution using biometric technology. This will save time wasted on calling out names and it gives a fool-proof method of attendance marking. A hand-held device is used to mark attendance without the intervention of the teacher. The device can be passed and students can mark attendance during the lecture time.

Title: Implementation of IOT based Attendance Management System on Raspberry Pi

Author: V Ruhitha; V N Prudhvi Raj; G Geetha

Content: In this paper we are presenting the implementation of an attendance monitoring system based on facial recognition. In this method the image database of the classroom is created first and the database is used to train the system. Then in the recognition phase the camera is fixed at the focal point in the classroom and it will capture the images, then from the captured images student's faces are recognized & compared with the database and lastly the attendance is marked based on the faces identified.

Title: Enhanced attendance Management System: A Biometrics System of Identification Based on Fingerprint

Author: Happy N. Monday; Ifeanyi D. Dike; Jian P. Li; David Agomuo; Grace U. Nneji; Abel Ogungbile

Content: This paper presents an enhanced attendance management system using a fingerprint system in a university environment. It was developed using the waterfall methodology. This system consists of two procedures; enrollment and identification. During the enrollment, the fingerprint of a person is captured and its unique features extracted and stored in the database along with the user's data as template for the subject. During identification, the fingerprint of the person is again captured and the extracted feature is compared with the template in the database in a ratio 1: N-templates, to identify a match (a user) before attendance is made.

Existing system:

In existing system there is no proper maintenance of attendance system, in this system students won't stay in class room also they have an attendance.

Proposed system:

Our project aims at building a biometrics based attendance registry system for educational institutions such as companies. We have tried to bring together concepts of hardware and software engineering to give an end consumer product which can replace the current manner of attendance marking.

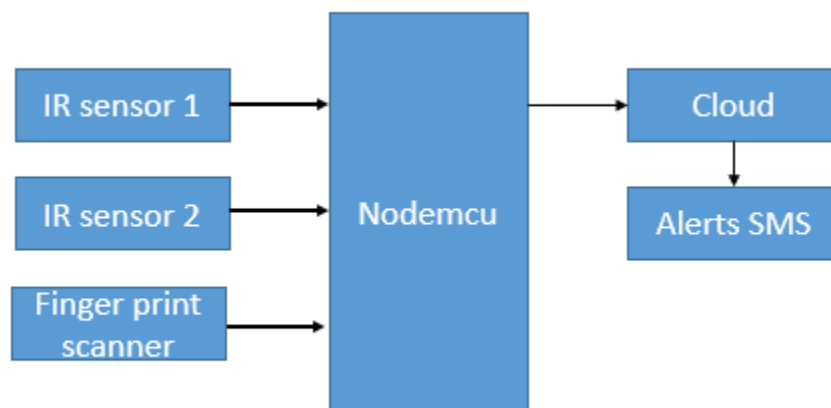
Most of the institutional authorities are troubled with the cumbersome method of maintaining manual attendance of their employees. The manual process of signing on a paper is prolonged and insecure. An efficient attendance monitoring system needs to be enforced at such places..

This project also utilizes the fundamentals of Internet of Things (IoT) for data transfer, storage and display. The system can be interpreted based on its hardware and software aspects.

The hardware section consists of a pod, which is a portable device that can be circulated among employees in the company. It features an LCD screen that displays various functional options and a, which can scan, store and identify student fingerprints.

The identified fingerprint ID is then stored in memory and is made ready for transfer. The data is then sent to the database on a web server where the Database Management System (DBMS) updates the attendance tables accordingly. This database can be downloaded or viewed by the respective authorities.

Block diagram:



Hardware required:

Nodemcu

Finger print

Buzzer

Display

Software required:

Arduino ide

Embedded C

IOT:

The structure of this chapter is the following: Section 1.1 defines the motivation for this work. Section 1.2 shows the research objectives and questions. Section 1.3 presents the steps required to achieve our objectives. Section 1.4 shows the structure of the report.

1.1 Background

The Future Internet goal is to provide an infrastructure to have an immediate access to information about the physical world and its objects. Physical objects can be applicable to different application domains, such as e-health, warehouse management, etc. Each application domain may have different types of physical devices. Each physical device can have its own specifications, which is required to use in order to interact with it. To achieve the future Internet goal, a layered vision is required that can facilitate data access. Internet of Things (IoT) is a vision that aims to integrate the virtual world of information to the real world of devices through a layered architecture.

The term „Internet of Things“ consists of two words, namely *Internet* and *Things*. *Internet* refers to the global network infrastructure with scalable, configurable capabilities based on interoperable and standard communication protocols. *Things* are physical objects or devices, or virtual objects, devices or information, which have identities, physical attributes and virtual personalities, and use intelligent interfaces [1]. For instance, a virtual object can represent an abstract unit of sensor nodes that contains metadata to identify and discover its corresponding sensor nodes. Therefore, IoT refers to the *things* that can provide information from the physical environment through the Internet.

Middleware is an interface between the hardware layer and the application layer, which is responsible for interacting with devices and information management [2]. The role of a middleware is to present a unified programming model to interact with devices. A middleware is in charge of masking the heterogeneity and distribution problems that we face when interacting with devices [3].

1.2 Motivation

IoT-based systems are in charge of providing knowledge from an environment to a non-expert user. IoT-based systems can be used in different environments, so it needs to be able to address many heterogeneous devices. Thus, a major concern within developing an IoT-based system is how to handle the interaction with the heterogeneous devices for non-expert users. This concern can be addressed by a middleware layer between devices and non-expert users. This layer is responsible for hiding the diversity of devices from the user perspective, and provides access transparency to the devices for the end users.

The idea of creating abstractions of devices has been addressed in the literature. The middleware we found in the literature can provide satisfaction by facilitating the interaction with devices, but they do not support low-level device configuration [4].

2.1. IoT definition

In this section, we explain some of the IoT definitions. Also, we explain the layered architecture for IoT.

The Internet of Things (IoT) has increasingly gained attention in industry to interact with different types of devices. IoT can have influence on industry and society by integrating physical devices into information networks [8]. IoT impacts can be on different perspectives, namely for private and business users. From the perspective of a private user, IoT has an effect on both working and personal fields, such as smart homes and offices, e-health and assisted living. From the aspect of a business user the impacts would be in fields such as automation and industrial manufacturing, logistics, business process management, intelligent transportation of people and goods [9]

IoT integrates physical things into information networks. IoT covers the overall infrastructure, including software, hardware and services, which is used to support these information networks. The integrated physical things can exchange data about the physical properties and information that they sense in their environment. To identify devices, we can use identification technologies like for example RFID, which allow each device to be uniquely identified [10].

International Telecommunication Union (ITU)¹ defines IoT as “A global infrastructure for the Information Society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies”[11]

Internet of Things-Architecture ² (IoT-A) defines it as “The idea of a globally interconnected continuum of devices, objects and things in general emerged with the RFID technology, and this concept has considerably been extended to the current vision that envisages a plethora of heterogeneous objects interacting with the physical environment.”[12]

IoT has a layered architecture designed to answer the demands of various industries, enterprises and society. Fig. 2.1 shows a generic layered architecture for IoT that consist of five layers, which are discussed, in the following:[13]

Edge Technology layer

This is a hardware layer that consists of embedded systems, RFID tags, sensor networks and all of the other sensors in different forms. This hardware layer can perform several functions, such as collecting information from a system or an environment, processing information and supporting communication.

Access Gateway layer

This layer is concerned with data handling, and is responsible for publishing and subscribing the services that are provided by the *Things*, message routing, and hovelling the communication between platforms.

Middleware layer

This layer has some critical functionalities, such as aggregating and filtering the received data from the hardware devices, performing information discovery and providing access control to the devices for applications.

Application layer

This layer is responsible for delivering various application services. These services are provided through the middleware layer to different applications and users in IoT-based systems. The application services can be used in different industries such as, logistics, retail, healthcare, etc.

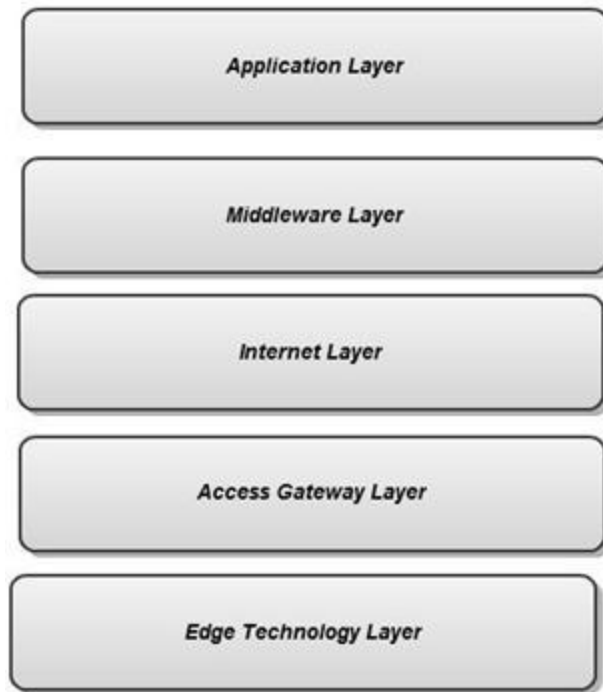


Figure 2-1 Layered architecture of the Internet of Things (IoT).

2.2 Challenges

IoT-based systems can be used for different purposes and areas, so we have to face different challenges. In this section, we explain some of the challenges that need to be considered in research activities [10]:

☛ Edge technology

At the hardware level, more research efforts are required to develop the technology of embedded devices, sensors, actuators and (passive and active) identification, since an IoT-based system must be able to gather sufficient information about the real world by employing a large variety of devices and environments. Thus, more work is still required to connect heterogeneous devices and deploy them in IoT applications, and to provide support for new devices.

Networking technologies

In IoT, things are connected through different kinds of networks, i.e. mobile, wired and wireless networks. These networks support bi-directional communication at different levels among the real world objects, applications and

services that are employed by the IoT applications. This highly distributed structure should provide interconnection with low energy consumption, while distributed data can cause privacy issues.

Middleware system

In IoT, we have heterogeneous devices and networks. Their heterogeneity can potentially increase with new technologies. To facilitate the use of these devices by IoT applications, we should shield their heterogeneity. Therefore, we need to develop a secure, scalable and semantically enriched service-oriented middleware to cope with the heterogeneity of devices.

Platform services

They support a superior management of all involved devices in an integrated way, ensuring scalability, high availability, and the safe and secure execution of the requested functionality from devices.

In continuation, we focus on the middleware challenges, because we are looking for the functionalities that a middleware can provide for the application layer in the IoT-based systems.

2.3 IoT_based Middleware

Bandyopadhyay, S. et. al. have studied the middleware systems that have been applied in IoT-based systems [1]. They classify the required functionality of middleware to manage interaction with a variety of devices in four functional components, namely (1) interface protocols, (2) device abstraction, (3) central control, context detection & management, and (4) application abstraction (shown in Fig. 2.2). In the following, we explain these components in detail.

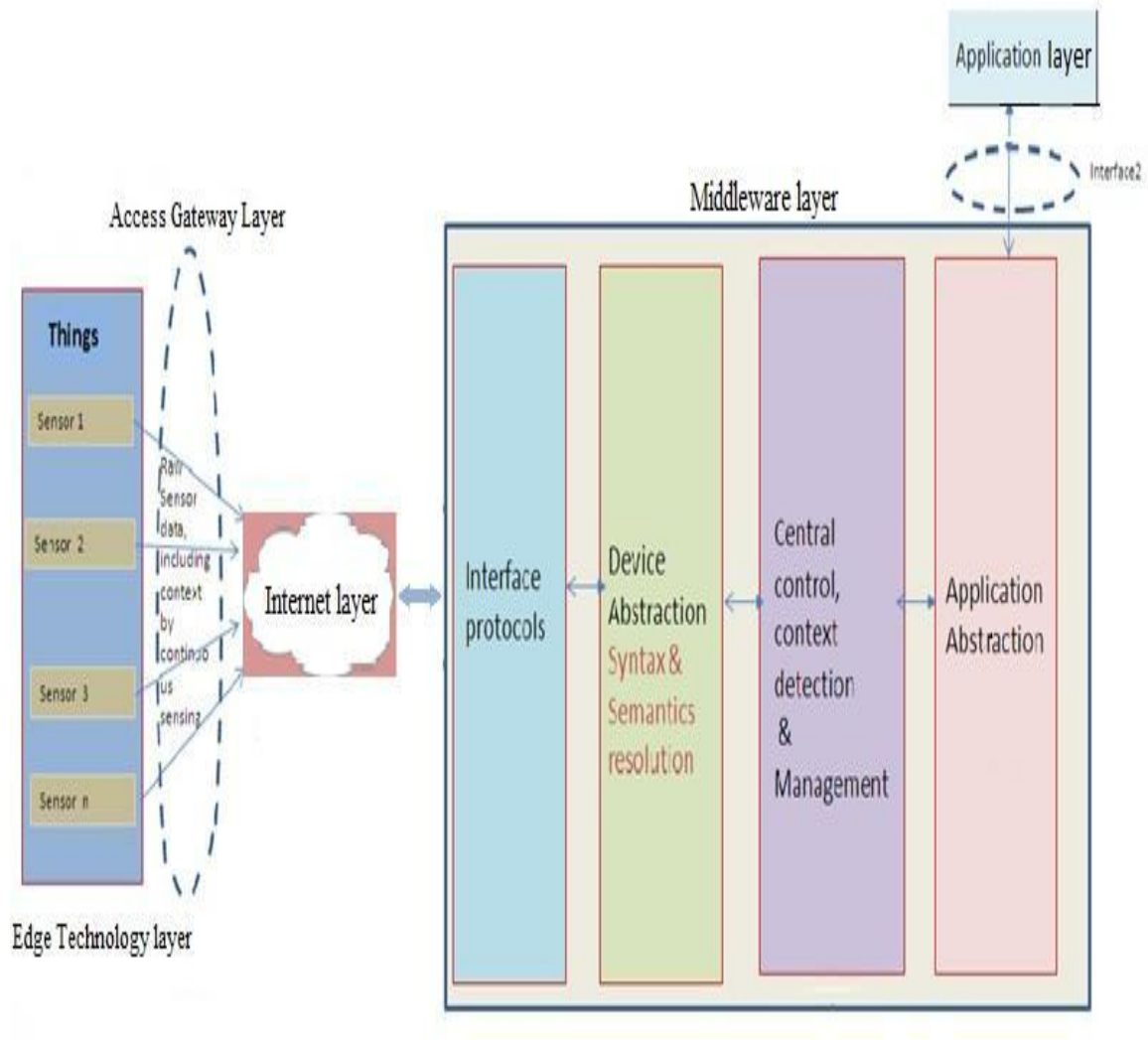


Figure 2.2 Functional components of a middleware for IoT-based systems

2.3.1 Interface protocols

This component is in charge of providing technical interoperability. Interoperability in the context of Interface protocols means: the ability of two systems to interoperate by using the same communication protocols. According to ETSI (European Telecommunications Standards Institute) [14] technical interoperability is defined as the association of hardware or software components, systems and platforms that enable machine-to-machine communication to take place. This kind of interoperability is often centered on (communication) protocols and the infrastructure needed for those protocols

to operate[14].

The Interface Protocol component defines protocols for exchanging information among different networks that may work based on different communication protocols, in order to

allow technical interoperability. This component is responsible for handling basic connectivity in the physical and data link, network, transport, and sometimes the application layer of the TCP/IP stack.

To cope with the heterogeneity of devices, we can use a wrapper for each device to translate the protocol supported by the device to a common protocol. This wrapper can be placed either on the device side or the middleware side. If we want to have direct interaction with devices, we should place the wrapper in the middleware side. Devices usually have limited capability of computational process, so this would be a reason to implement wrapper on the middleware side. In contrast, in case of indirect interaction with devices we can develop an intermediary wrapper between the middleware and the devices. The interface protocol component is responsible for allowing the middleware to support both direct and indirect interactions.

2.3.2 Device Abstraction (DA)

This component is responsible for providing an abstract format to facilitate the interaction of the application components with devices. This abstraction provides syntactic and semantic interoperability, which are defined by ETSI [14] as follows:

Syntactic interoperability is associated with data formats. The messages transferred by communication protocols must have a well-defined syntax and encoding format, which can be represented by using high-level transfer syntaxes such as, HTML and XML.

Semantic interoperability is usually associated with the meaning of the content of a message which is understandable for humans. Thus, interoperability on this level means that there is a common understanding among people on the meaning of the content (information) being exchanged among them. However, since DA does not communicate directly with humans, semantic interoperability in the context of DA is in charge of providing this common understanding for applications.

Semantic interoperability relies on semantic models which tend to be domain specific. For example, one way to provide semantic interoperability in Service Oriented (SOA) [15] based middleware is by using Devices Profile for Web Services (DPWS) [16] . In this context, each device type refers to a distinguished service type

DA components provide two general functionalities: (1) to ask devices to perform some functionality and (2) to define and configure devices DPWS uses the XML format that is shown in Code 2.1.

```

<dpws:ThisModel>
  <dpws:Manufacturer>ACME Manufacturing</dpws:Manufacturer>
  <dpws:ModelName xml:lang="en-GB" >
    ColourBeam 9
  </dpws:ModelName>
  <dpws:ModelName xml:lang="en-US" >
    ColorBeam 9
  </dpws:ModelName>
</dpws:ThisModel>
</wsx:MetadataSection>
<wsx:MetadataSection
Dialect="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01/ThisDevice"
>
  <dpws:ThisDevice>
    <dpws:FriendlyName xml:lang="en-GB" >
      ACME ColourBeam Printer
    </dpws:FriendlyName>
    <dpws:FriendlyName xml:lang="en-US" >
      ACME ColorBeam Printer
    </dpws:FriendlyName>
  </dpws:ThisDevice>

```

Code 2.1 XML code to define a device

2.3.3 Central control, Context detection & Management (CCM)

Context characterizes the situation of an entity, which can be a place, a person or an object that is relevant to the user, applications and their interactions [1]. The CCM functional component is responsible to support context-aware computation that is a computational style that takes into account the context of the entities that interact with the system. A middleware for IoT-based systems must be context-aware to work in smart environments [1]. Smart environments refer to a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network [17]. Context-awareness includes two functionalities:

- 1) Context detection, which consists of collecting data from resources, and selecting the information that can have an impact on the computation.

- 2) Context processing, to use the gathered information to perform a task or make a decision.

2.3.4 Application Abstraction

This functional component provides an interface for both high-level applications and end-users to interact with devices. For instance, this interface can be a RESTful interface or can be implemented with some query-based language.

A. Malatras. et. al. propose a SOA-based middleware to perform data management and data monitoring services [18]. This middleware uses a RESTful interface to facilitate the interaction of high-level applications with sensors, which can communicate with Wireless Sensor Network (WSN) through the HTTP operations: (1) GET to issue a query on an existing resource, (2) DELETE to remove an existing resource, (3) POST to create a new resource, and (4) PUT to update an existing resource. For instance, a client application gets the result of a domain task by sending a GET request through the URL: „http://{hostname}/REST/{version}/DomainTaskResult/id’. In this URI that reference to an „id” leads to a unique result.

In the TinyDB middleware [19], an end-user can use a query language to interact with devices. For instance, a user can send the following query to get the id of a sensor

(nodeid) nearby and temperatures that are sensed by this sensor during the past 10 seconds before executing the query:

```
SELECT nodeid, temp
FROM sensors
SAMPLE PERIOD 1s FOR 10s
```

2.4 Conclusion

In this chapter, we defined IoT and identified common IoT layers. Furthermore, we discussed a reference middleware architecture for IoT-based systems. This architecture has been proposed by Bandyopadhyay, S. et. al based on a study on the existing middleware frameworks for IoT-based systems [1].

Chapter 3 Review of IoT-based Middleware

The term Internet of Things (IoT) refers to a wide set of applications and research areas such as, distributed computing and knowledge management. Much of the IoT research we found in the literature is related to the field of pervasive computing. World Wide Web Consortium (W3C)³ defines pervasive computing as a vision about our future computing lifestyle. In this vision, computer systems will be seamlessly integrated into our everyday life, anticipating our needs and providing relevant services and information for us in an anytime-and-any-anywhere fashion. Pervasive Computing is all about making the human's life easier by exploiting the available computing technologies [20]. This motivates scientists to develop new technologies for everyday people's lives.

Since the vision of IoT is almost similar to Pervasive computing, we reviewed the literature that addresses both IoT and Pervasive systems. To select the middleware to discuss we had the following approaches:

- 1) We looked at typical IoT middleware such as ISMP [21], ASPIRE [22] GSN [23]. We selected HYDRA [24] to review, because it is the most popular and well-documented middleware in comparison to the mentioned middleware. Also, based on the study of Bandyopadhyay, S. et. al [3], those middleware does not support the discussed functional components in Chapter 3, while HYDRA supports them.
- 2) We looked at typical Pervasive system middleware such as [25], [26], [27]. We selected the following middleware to review:

AURA [28] because this middleware focuses on elaboration and manipulation of the gathered data from devices. We aim to provide ease of configuration and

deployment for end users and developers. To meet this goal, to facilitate the gathered data manipulation, we reviewed AURA, which was the only middleware among the reviewed paper that considered the data manipulation.

TinyDB [19] focuses on gathering data from devices. Since in an IoT-based system we need to gather data from the environment through different devices, we reviewed TinyDB, which is a popular middleware.

WiseMID [29] is the only middleware among the reviewed middleware that is specific for energy saving purposes. As saving the energy of devices is an important issue, we reviewed this middleware.

In this chapter, we mentioned middleware in more detail. Also, we analyze the discussed middleware with respect to the functional components mentioned in Chapter 2.

NODE MCU

NodeMCU is an open source [IoT](#) platform.^{[4][5]} It includes [firmware](#) which runs on the [ESP8266 Wi-Fi SoC](#) from [Espressif Systems](#), and hardware which is based on the ESP-12 module.^{[6][7]} The term "NodeMCU" by default refers to the firmware rather than the development kits. The firmware uses the [Lua](#) scripting language. It is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open source projects, such as lua-cjson^[8] and [SPIFFS](#)

NodeMCU was created shortly after the [ESP8266](#) came out. On December 30, 2013, [Espressif Systems](#)^[6] began production of the ESP8266.^[10] The ESP8266 is a Wi-Fi SoC integrated with a [Tensilica](#) Xtensa LX106 core,^[citation needed] widely used in IoT applications (see [related projects](#)). NodeMCU started on 13 Oct 2014, when Hong committed the first file of nodemcu-firmware to GitHub.^[11] Two months later, the project expanded to include an open-hardware platform when developer Huang R committed the [gerber](#) file of an ESP8266 board, named devkit v0.9.^[12] Later that month, Tuan PM ported [MQTT](#) client library from [Contiki](#) to the ESP8266 SoC platform,^[13] and committed to NodeMCU project, then NodeMCU was able to support the MQTT IoT protocol, using Lua to access the MQTT broker. Another important update was made on 30 Jan 2015, when Devsaurus ported the u8glib^[14] to NodeMCU project,^[15] enabling NodeMCU to easily drive LCD, Screen, OLED, even VGA displays.

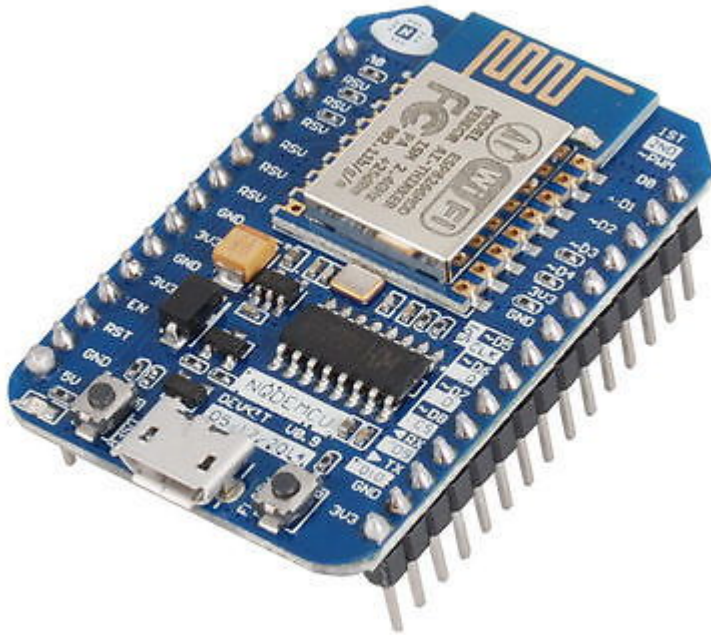
In summer 2015 the creators abandoned the firmware project and a group of independent contributors took over. By summer 2016 the NodeMCU included more than 40 different modules. Due to resource constraints users need to select the modules relevant for their project and build a firmware tailored to their needs.

ESP8266 Arduino Core[edit]

As Arduino.cc began developing new MCU boards based on non-AVR processors like the ARM/SAM MCU and used in the Arduino Due, they needed to modify the Arduino IDE so that it would be relatively easy to change the IDE to support alternate toolchains to allow Arduino C/C++ to be compiled for these new processors. They did this with the introduction of the Board Manager and the SAM Core. A "core" is the collection of software components required by the Board Manager and the Arduino IDE to compile an Arduino C/C++ source file for the target MCU's machine language. Some ESP8266 enthusiasts developed an Arduino core for the ESP8266 WiFi SoC, popularly called the "ESP8266 Core for the Arduino IDE".^[16] This has become a leading software development platform for the various ESP8266-based modules and development boards, including NodeMCUs.

Introduction

NodeMCU is an open source [LUA](#) based firmware developed for ESP8266 wifi chip. By exploring functionality with ESP8266 chip, NodeMCU firmware comes with ESP8266 Development board/kit i.e. NodeMCU Development board.

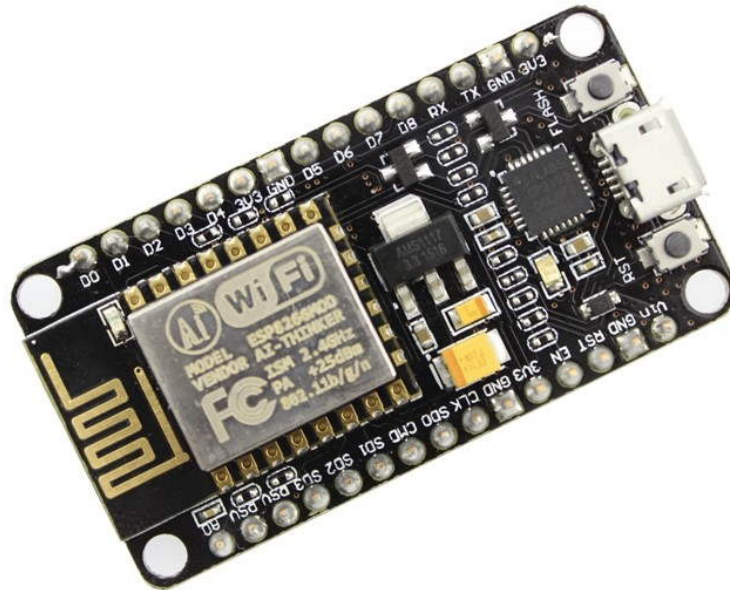


NodeMCU Development Board/kit v0.9 (Version1)

Since NodeMCU is an open source platform, their hardware design is open for edit/modify/build.

NodeMCU Dev Kit/board consist of ESP8266 wifi enabled chip. The **ESP8266** is a low-cost [Wi-Fi](#) chip developed by Espressif Systems with TCP/IP protocol. For more information about ESP8266, you can refer [ESP8266 WiFi Module](#).

There is Version2 (V2) available for NodeMCU Dev Kit i.e. **NodeMCU Development Board v1.0 (Version2)**, which usually comes in black colored PCB.



NodeMCU Development Board/kit v1.0 (Version2)

For more information about NodeMCU Boards available in market refer [NodeMCU Development Boards](#)

NodeMCU Dev Kit has **Arduino like** Analog (i.e. A0) and Digital (D0-D8) pins on its board.

It supports serial communication protocols i.e. UART, SPI, I2C etc.

Using such serial protocols we can connect it with serial devices like I2C enabled LCD display, Magnetometer HMC5883, MPU-6050 Gyro meter + Accelerometer, RTC chips, GPS modules, touch screen displays, SD cards etc.

How to start with NodeMCU?

NodeMCU Development board is featured with wifi capability, analog pin, digital pins and serial communication protocols.

To get started with using NodeMCU for IoT applications first we need to know about how to write/download NodeMCU firmware in NodeMCU Development Boards. And before that where this NodeMCU firmware will get as per our requirement.

There are online NodeMCU custom builds available using which we can easily get our custom NodeMCU firmware as per our requirement.

To know more about how to build custom NodeMCU firmware online and download it refer [Getting started with NodeMCU](#)

How to write codes for NodeMCU?

After setting up ESP8266 with Node-MCU firmware, let's see the IDE (Integrated Development Environment) required for development of NodeMCU.

NodeMCU with ESPlorer IDE

[Lua](#) scripts are generally used to code the NodeMCU. Lua is an open source, lightweight, embeddable scripting language built on top of C programming language.

For more information about how to write Lua script for NodeMCU refer [Getting started with NodeMCU using ESPlorerIDE](#)

NodeMCU with Arduino IDE

Here is another way of developing NodeMCU with a well-known IDE i.e. Arduino IDE. We can also develop applications on NodeMCU using the Arduino development environment. This makes it easier for Arduino developers than learning a new language and IDE for NodeMCU.

For more information about how to write Arduino sketch for NodeMCU refer [Getting started with NodeMCU using ArduinoIDE](#)

Difference in using ESPlorer and Arduino IDE

Well, there is a programming language difference we can say while developing applications for NodeMCU using ESPlorer IDE and Arduino IDE.

We need to code in C/C++ programming language if we are using Arduino IDE for developing NodeMCU applications and Lua language if we are using ESPlorer IDE.

Basically, NodeMCU is Lua Interpreter, so it can understand Lua script easily. When we write Lua scripts for NodeMCU and send/upload it to NodeMCU, then they will get executed sequentially. It will not build a binary firmware file of code for NodeMCU to write. It will send Lua script as it is to NodeMCU to get executed.

In the Arduino IDE when we write and compile code, the ESP8266 toolchain in the background creates a binary firmware file of code we wrote. And when we upload it to NodeMCU then it will flash all NodeMCU firmware with newly generated binary firmware code. In fact, it writes the complete firmware.

That's the reason why NodeMCU does not accept further Lua scripts/code after it is getting flashed by Arduino IDE. After getting flashed by Arduino sketch/code it will be no more Lua interpreter and we get errors if we try to upload Lua scripts. To start again with Lua script, we need to flash it with NodeMCU firmware.

Since Arduino IDE compiles and uploads complete firmware, it takes more time than ESPlorer IDE.

FINGERPRINT SENSOR:

A fingerprint in its narrow sense is an impression left by the friction ridges of a human finger. The recovery of fingerprints from a crime scene is an important method of forensic science. Fingerprints are easily deposited on suitable surfaces (such as glass or metal or polished stone) by the natural secretions of sweat from the glands that are present in epidermal ridges. These are sometimes referred to as "Chanced Impressions".

In a wider use of the term, fingerprints are the traces of an impression from the friction ridges of any part of a human or other primate hand. A print from the sole of the foot can also leave an impression of friction ridges.

Deliberate impressions of fingerprints may be formed by ink or other substances transferred from the peaks of friction ridges on the skin to a relatively smooth surface such as a fingerprint card.^[2] Fingerprint records normally contain impressions from the pad on the last joint of fingers and thumbs, although fingerprint cards also typically record portions of lower joint areas of the fingers.

Human fingerprints are detailed, nearly unique, difficult to alter, and durable over the life of an individual, making them suitable as long-term markers of human identity. They may be employed by police or other authorities to identify individuals who wish to conceal their identity, or to identify people who are incapacitated or deceased and thus unable to identify themselves, as in the aftermath of a natural disaster. Fingerprint analysis, in use since the early 20th century, has led to many crimes being solved. This means that many criminals consider gloves essential. In 2015, the identification of sex by use of a fingerprint test has been reported.

Biology

A friction ridge is a raised portion of the epidermis on the digits (fingers and toes), the palm of the hand or the sole of the foot, consisting of one or more connected ridge units of friction ridge skin.^[1] These are sometimes known as "epidermal ridges" which are caused by the underlying interface between the dermal papillae of the dermis and the interpapillary (rete) pegs of the epidermis. These epidermal ridges serve to amplify vibrations triggered, for example, when fingertips brush across an uneven surface, better transmitting the signals to sensory nerves involved in fine texture perception. These ridges may also assist in gripping rough surfaces and may improve surface contact in wet conditions.

Types

Before computerization, manual filing systems were used in large fingerprint repositories. Manual classification systems were based on the general ridge patterns of several or all fingers (such as the presence or absence of circular patterns). This allowed the filing and retrieval of paper records in large collections based on friction ridge patterns alone. The most popular systems used the pattern class of each finger to form a key (a number) to assist lookup in a filing system. Classification systems include the Roscher system, the Juan

Vucetich system, and the Henry Classification System. The Roscher system was developed in Germany and implemented in both Germany and Japan, the Vucetich system (developed by a Croatian-born Buenos Aires Police Officer) was developed in Argentina and implemented throughout South America, and the Henry system was developed in India and implemented in most English-speaking countries.

In the Henry system of classification, there are three basic fingerprint patterns: loop, whorl, and arch,^[11] which constitute 60–65%, 30–35%, and 5% of all fingerprints respectively.^[citation needed] There are also more complex classification systems that break down patterns even further, into plain arches or tented arches,^[10] and into loops that may be radial or ulnar, depending on the side of the hand toward which the tail points. Ulnar loops start on the pinky-side of the finger, the side closer to the ulna, the lower arm bone. Radial loops start on the thumb-side of the finger, the side closer to the radius. Whorls may also have sub-group classifications including plain whorls, accidental whorls, double loop whorls, peacock's eye, composite, and central pocket loop whorls.

Other common fingerprint patterns include the tented arch, the plain arch, and the central pocket loop.

The system used by most experts, although complex, is similar to the Henry System of Classification. It consists of five fractions, in which R stands for right, L for left, i for index finger, m for middle finger, t for thumb, r for ring finger and p (pinky) for little finger. The fractions are as follows: $R_i/R_t + R_r/R_m + L_t/R_p + L_m/L_i + L_p/L_r$. The numbers assigned to each print are based on whether or not they are whorls. A whorl in the first fraction is given a 16, the second an 8, the third a 4, the fourth a 2, and 0 to the last fraction. Arches and loops are assigned values of 0. Lastly, the numbers in the numerator and denominator are added up, using the scheme:

$$(R_i + R_r + L_t + L_m + L_p)/(R_t + R_m + R_p + L_i + L_r)$$

and a 1 is added to both top and bottom, to exclude any possibility of division by zero. For example, if the right ring finger and the left index finger have whorls, the fractions would look like this:

$0/0 + 8/0 + 0/0 + 0/2 + 0/0 + 1/1$, and the calculation: $(0 + 8 + 0 + 0 + 0 + 1)/(0 + 0 + 0 + 2 + 0 + 1) = 9/3 = 3$.

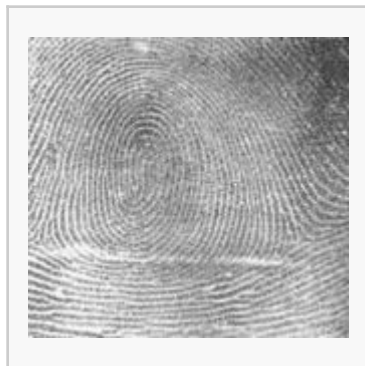
Using this system reduces the number of prints that the print in question needs to be compared to. For example, the above set of prints would only need to be compared to other sets of fingerprints with a value of 3.



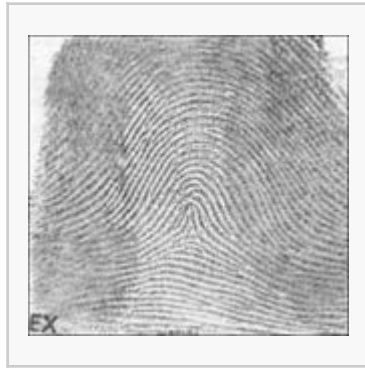
Arch



Loop
(Right Loop)



Whorl



Arch
(Tented Arch)

ELECTRONIC RECORDING

There has been a newspaper report of a man selling stolen watches sending images of them on a mobile phone, and those images included parts of his hands in enough detail for police to be able to identify fingerprint patterns.

Recent studies found that the improving cameras with increasing resolution of smartphones might have a high impact on users' security: The back-facing camera of a device can be used to capture an image of the user's index finger, which on smartphones using biometric means of authentication is often used to authenticate a user against the smartphone.

At the 31st Chaos Communication Congress, hardware hacker star bug presented how DSLRs with high resolution and equipped with a long focus lens can be used to capture images of hands, or more specifically, fingers in order to use them for spoofing.

FOOTPRINTS

Friction ridge skin present on the soles of the feet and toes (plantar surfaces) is as unique in its ridge detail as are the fingers and palms (palmar surfaces). When recovered at crime scenes or on items of evidence, sole and toe impressions can be used in the same manner as finger and palm prints to effect identifications. The footprints of infants, along with

the thumb or index finger prints of mothers, are still commonly recorded in hospitals to assist in verifying the identity of infants. It is not uncommon for military records of flight personnel to include bare foot inked impressions. Friction ridge skin protected inside flight boots tends to survive the trauma of a plane crash (and accompanying fire) better than fingers.

Fingerprint sensor module R305 (connected across CON2) has UART interfaces with direct connections to the MCU or to the PC through max 232/USB serial adapter. The user can store fingerprint data in the module and configure it in 1:1 or 1: N mode. For identification FP modules can directly interface with 3v3 or 5v microcontrollers.

The module supports USART communication protocol. Here, USART protocol is used for communicating with the microcontroller. Transmit pin of the module is connected to the receive pin of the microcontroller. Receive pin should be connected to the transmit pin of the microcontroller. VIN is 5V and GND is connected to ground.



Fingerprint Sensor Module R305

Fingerprint scanner is low cost, low power consumption, small size, and excellent performance professional optical technology. The user can store the fingerprint data in the module and can configure it in 1:1 or 1: N mode for identifying the person. Good image processing capabilities can successfully capture images up to resolution 500 dpi.

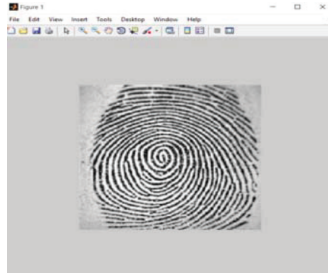
This is a fingerprint sensor module with TTL UART interface for direct connections to microcontroller UART or to PC through MAX232 / USB-Serial adapter. The user can store the fingerprint data in the module and can configure it in 1:1 or 1: N mode for identifying the person. The FP module can directly interface with 3.3 or 5v Microcontrollers. A level converter (like MAX232) is required for interfacing with PC serial port.

A. MINUTIAE BASED FINGERPRINT RECOGNITION

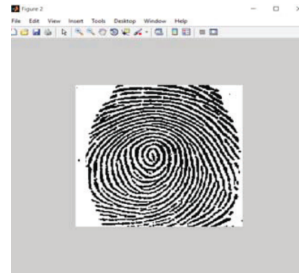
The fingerprint image undergoes pre-processing stages like binarization which uses fixed threshold to convert a grayscale image to a binary image and then proceeds to thinning process to reduce the thickness of all ridge edge lines to a single pixel width after which an initial code is generated, prior to the secured final code. The code block consists of five sub-blocks placed within the header and trailer. The five sub block patterns are five minutiae characteristics which are chosen of suitable length for example consider 14 bytes:

- **Type:** It specifies the termination and bifurcation points. Three bytes are allocated for this parameter.
- **Orientation:** Each minutiae point faces a particular direction. It is either clockwise (CW) or counter clockwise (CC). Two bytes are allocated for this parameter. Let gradient x be (gx) and gradient y be (gy)
- **Spatial Frequency:** indicates the distance of the ridges in the neighbourhood of the minutiae point. It's measured in pixels and only one byte is allocated for this parameter.
- **Curvature:** Is the rate of change of ridge orientation. It is also measured in pixels and one byte is allocated for this parameter.
- **Position:** indicates its x, y location. It is calculated in relative to the core or delta points One byte is allocated for this parameter. An initial code string of 14 bytes is generated depending on these features and it is saved in the database. Later this code is passed through the one way

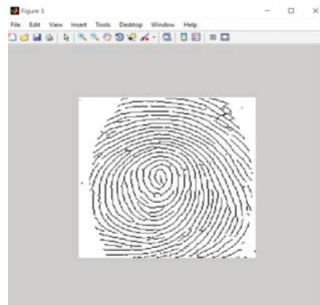
hash MD5 algorithm to generate a secured multipurpose code. The Fig.2 gives the matlab results when a minutiae matching algorithm is applied to a fingerprint image from the Fingerprint Verification Competition (FVC) 2002 database.



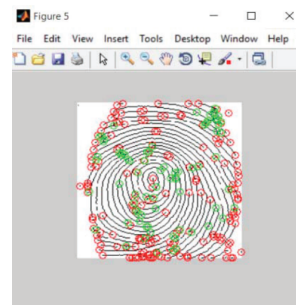
(a)



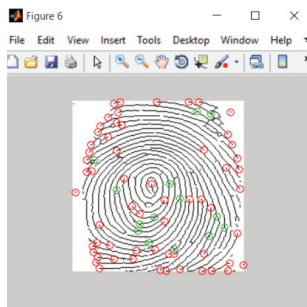
(b)



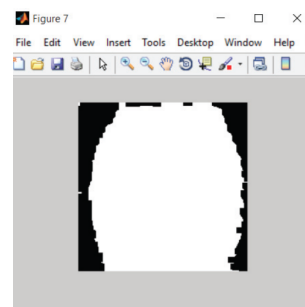
(c)



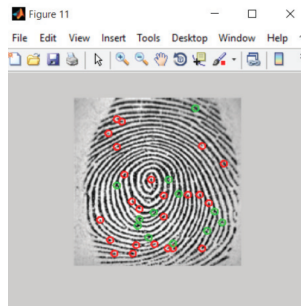
(d)



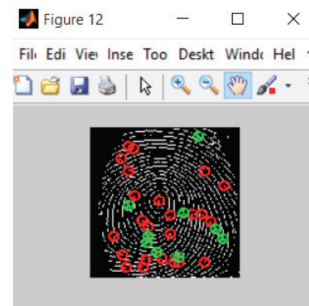
(e)



(f)



(g)



(h)

Infrared Sensors:

Infrared (IR) light is electromagnetic radiation with a wavelength longer than that of visible light, measured from the nominal edge of visible red light at 0.7 micrometers, and extending

conventionally to 300 micrometers. These wavelengths correspond to a frequency range of approximately 430 to 1 THz, and include most of the thermal radiation emitted by objects near room temperature. Microscopically, IR light is typically emitted or absorbed by molecules when they change their rotational-vibration movements. Sunlight at zenith provides an irradiance of just over 1 kilowatt per square meter at sea level. Of this energy, 527 watts is infrared radiation, 445 watts is visible light, and 32 watts is ultraviolet radiation.

The basic idea is to send infrared light through IR-LEDs, which is then reflected by any object in front of the sensor. Then all you have to do is to pick-up the reflected IR light. For detecting the reflected IR light, we are going to use a very original technique: we are going to use another IR-LED, to detect the IR light that was emitted from another led of the exact same type. This is an electrical property of Light Emitting Diodes (LEDs) which is the fact that a led produces a voltage difference across its leads when it is subjected to light. As if it was a photo-cell, but with much lower output current. In other words, the voltage generated by the leds can't be - in any way - used to generate electrical power from light, it can barely be detected. That's why Op Amps are mostly used for accurate detection of low voltages.

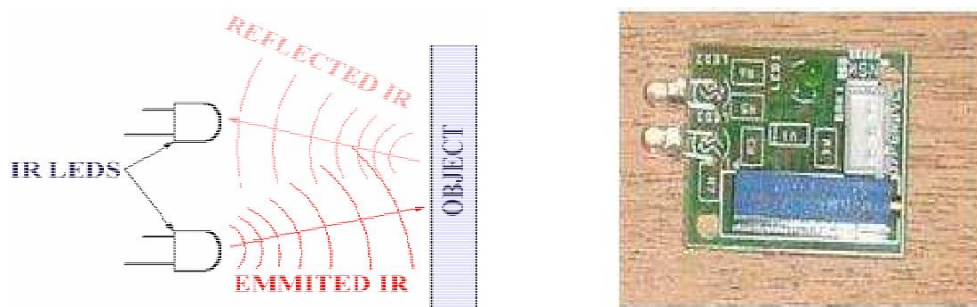


Fig 4.14.1: IR Sensors and its working

Arduino ide:

Introduction

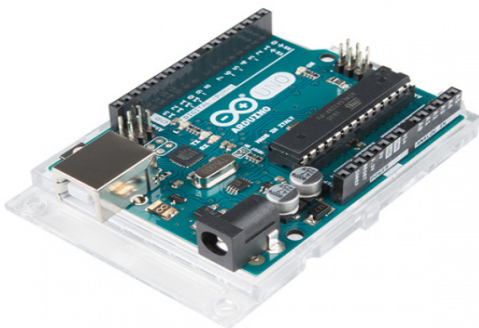
This tutorial will walk you through downloading, installing, and testing the [Arduino software](#) (also known as the Arduino IDE - short for Integrated Development Environment). Before you jump to the page for your operating system, make sure you've got all the right equipment.



Required Materials

To follow along with this tutorial, you will need the following materials. You may not need everything though depending on what you have. Add it to your cart, read through the guide, and adjust the cart as necessary.

- A computer (Windows, Mac, or Linux)
- An Arduino-compatible microcontroller (anything from [this guide](#) should work)
- A USB A-to-B cable, or another appropriate way to connect your Arduino-compatible microcontroller to your computer (check out this [USB buying guide](#) if you're not sure which cable to get).



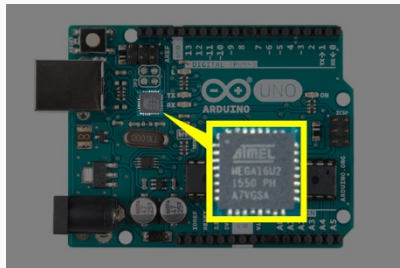
Arduino Uno - R3

USB Cable A to B - 6 Foot

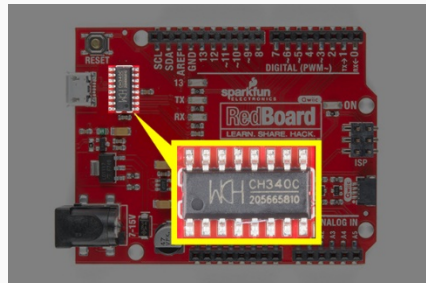
What is an Arduino?

What is this 'Arduino' thing anyway? This tutorial dives into what an Arduino is and along with Arduino projects and widgets.

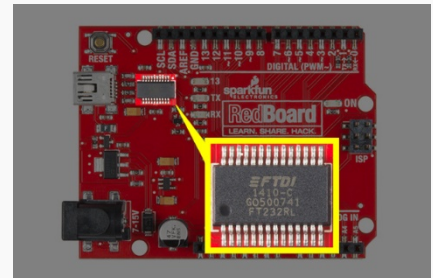
Note: There are several variants that use the Arduino Uno R3 footprint. Depending on the design, you may need to install additional drivers for your USB-to-serial converter before you are able to upload code to your microcontroller. For example, the RedBoard uses an FTDI while the RedBoard Plus uses the CH340.



*Atmega16U2 on the
Arduino Uno R3*



*FTDI on the
RedBoard*



*CH340 on the
RedBoard Qwiic*

Both use different drivers compared to the Arduino Uno R3. Make sure to look closely at your board and its respective hookup guide to determine the USB-to-serial converter that is on board. You will probably have either an FTDI or CH340 populated on the board.

How to Install FTDI Drivers

How to install drivers for the FTDI Basic on Windows, Mac OS X, and Linux.

How to Install CH340 Drivers

How to install CH340 drivers (if you need them) on Windows, Mac OS X, and Linux.

Other boards using a different architecture like the [RedBoard Turbo \(SAMD21\)](#) have a built-in USB communication, eliminating the need to have a separate piece of hardware. For more information, check out the blog post on drivers: [What Drives your SparkFun Inventor's Kit?](#).

Downloading the Arduino IDE

You can [download the Arduino IDE](#) from their website. They have [installation instructions](#), but we will also go over the installation process as well. Make sure you download the version that matches your operating system.

[CLICK FOR ARDUINO IDE DOWNLOAD PAGE](#)

The installation procedure is fairly straightforward, but it does vary by OS. Here are some tips to help you along.

Troubleshooting Tips

- We recommend using a computer with a full desktop operating system like Windows 7/10 (**avoid** Windows 8 if you can), Mac OSX, and certain flavors Linux (check the [Arduino FAQ page for compatibility](#)).
 - If you are not a technical or computer savvy individual and you have your choice of computers, I highly recommend using a **Windows 7 or 10** computer. You will usually run into the least issues, if any, with these operating systems.
- We do **NOT** recommend using a Chromebook, Netbook, tablet, phone, or the Arduino Web IDE in general. You will be responsible for troubleshooting any driver or Arduino Web IDE issues.
- As of the writing of this tutorial (updated 12-14-2018), the most recent and stable release of the Arduino IDE is version 1.8.5. We recommend using that version of the Arduino IDE; you can download the [previous releases here](#).
- On Windows 10, we do **NOT** recommend installing the Arduino IDE from the app store. You may run into issues because the OS will automatically update to the most recent release of the Arduino IDE, which may have unknown bugs (like the compiler errors in versions 1.8.6

and 1.8.7).

- Raspberry Pi users with Raspbian installed should use the **Linux ARM** download. We do not recommend using the command line installation. It will install the oldest release of Arduino, which is useless when it comes to installing new boards definitions or libraries.
- For additional troubleshooting tips, here is a [troubleshooting guide from Arduino](#).

Windows

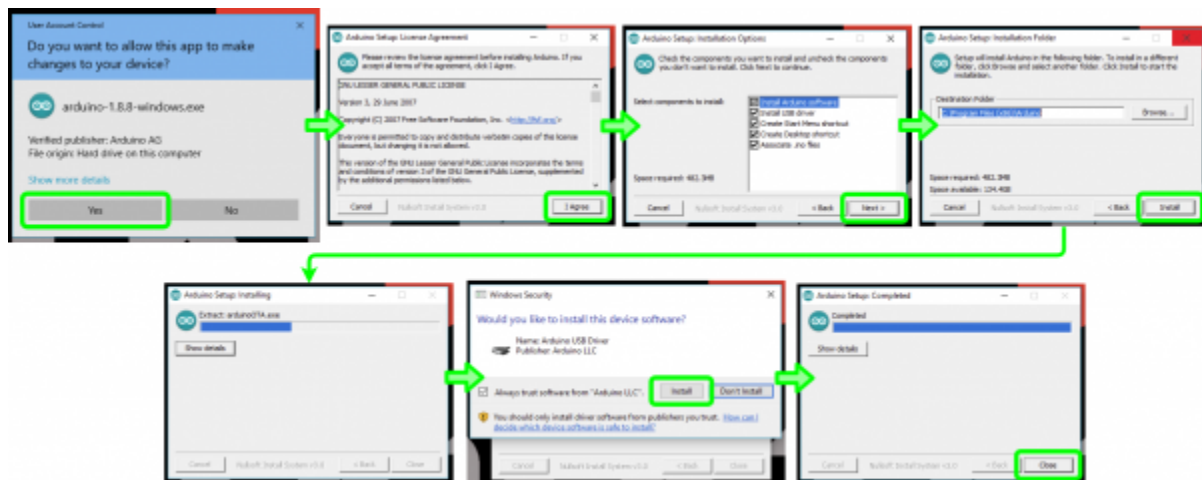
This page will show you how to install and test the Arduino software with a Windows operating system (Windows 10, Windows 7, Vista, and XP).

Go to the Arduino [download page](#) and download the latest version of the Arduino software for Windows if you have not already.

[CLICK FOR ARDUINO IDE DOWNLOAD PAGE](#)

Installer

The Windows version of Arduino is offered in two options: an installer or a zip file. The **installer** is the easier of the two options, just download that, and run the executable file to begin the installation.



Windows install steps. Click the image to get a bigger view.

When you're prompted to install a driver during installation, select "Install". This will install drivers for Arduino specific boards (like the Uno, Nano, etc.) that you may use in the future.

ZIP

If you choose to download the **zip file** version of Arduino, you'll need to extract the files yourself. Don't forget which folder you extract the files into! You will need to run the executable Arduino file in the folder to start the Arduino IDE.

When the download is finished, unzip it and open up the Arduino folder to confirm that yes, there are indeed some files and subfolders inside. The file structure is important so don't be moving any files around unless you really know what you're doing.

Note: On Windows 10, there is an option to install Arduino through their app store. we do not recommend installing the Arduino IDE from the app store. You may run into issues because the OS will automatically update to the most recent release of the Arduino IDE, which may have unknown bugs.

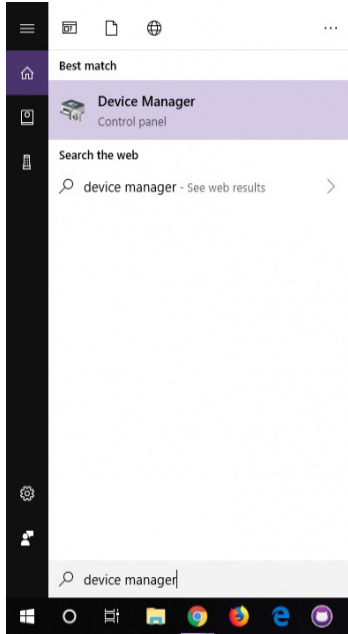
Connecting Your Arduino

Power up your Arduino by connecting your Arduino board to your computer with a USB cable (or FTDI cable if you're using an Arduino Pro). You should see an LED labeled 'ON' light up. ([this diagram](#) shows the placement of the power LED on the UNO).

Drivers for Arduino Uno on Windows

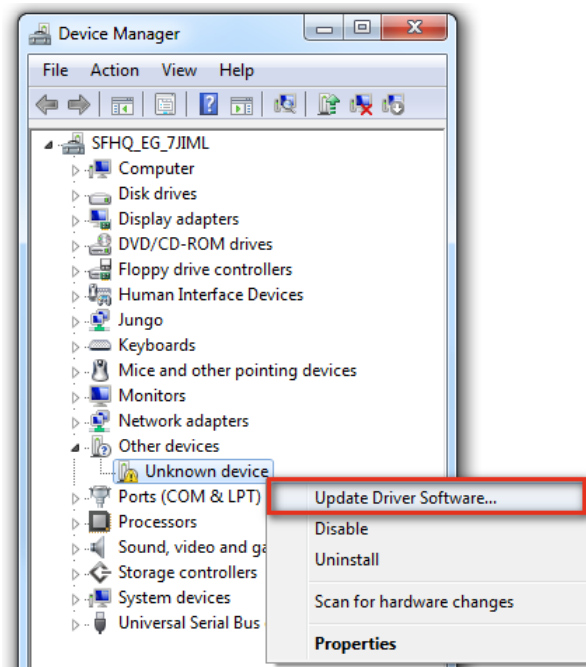
To install the drivers for the Arduino Uno, you will need to plug in your board to your computer's USB port. Once the board is connected, you will need to wait for Windows to begin its driver installation process. After a few moments, the process will probably fail, despite its best efforts.

Open up a search, type in **Device Manager**, and hit **ENTER**.

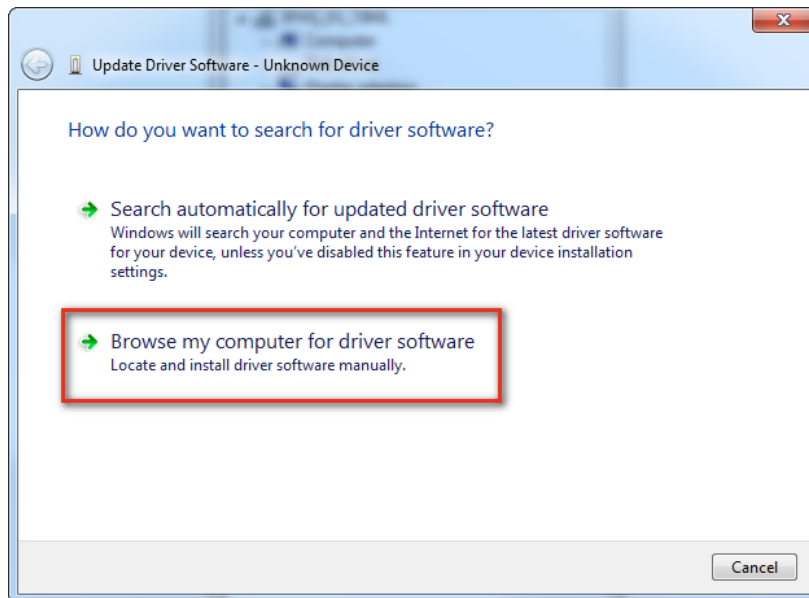


Note: Searching is the easiest method to open the Device Manager. However, there is more than one method of opening the device manager. The longer method is to click on the **Start Menu > Windows System > System and Security > System > Device Manager**.

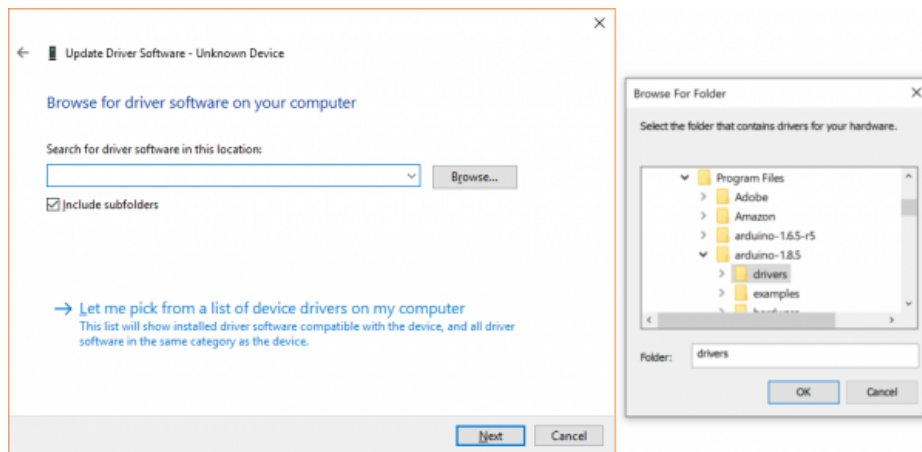
Look under the Ports (COM & LPT) tree. You should see an open port named "**Arduino UNO (COMxx)**". If there is no COM & LPT section, look under "**Other devices**" for "**Unknown Device**". Right click on the "**Arduino UNO (COMxx)**" or "**Unknown device**" port and choose the "**Update Driver Software**" option.



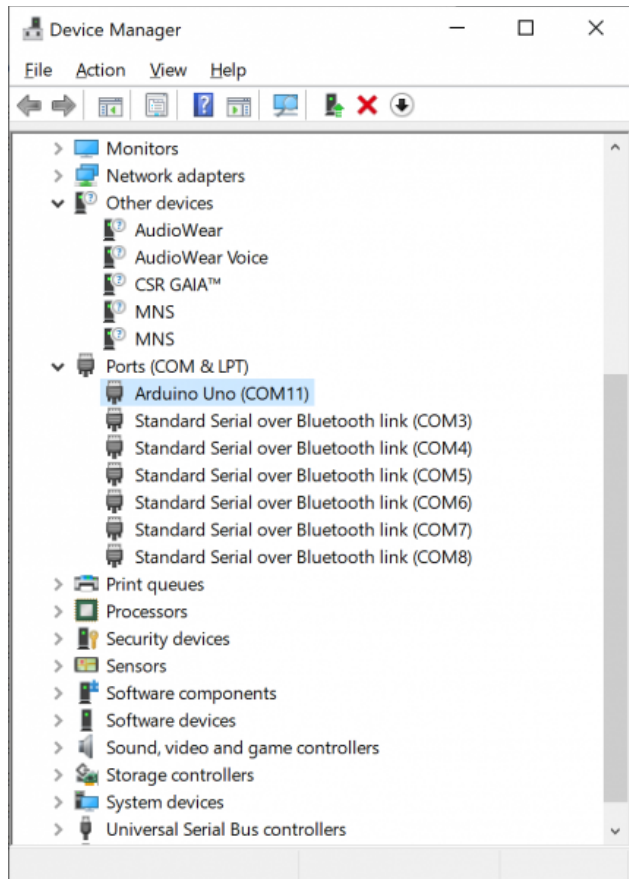
Next, choose the "**Browse my computer for Driver software**" option



Finally, navigate to the Arduino IDE folder. This should be where you unzipped the Arduino IDE (e.g. it should be similar to the following path with a different version number: *C:\Program Files\arduino-1.8.5\drivers*). Depending on what version of Windows you have, you may be able to select the Uno's driver file, named "**Arduino.inf**", located in the "**Drivers**" folder (not the "FTDI USB Drivers" sub-directory). If you cannot see the *.inf file, it is probably just hidden. You can select the "**drivers**" folder with the "**include subfolders**" option selected instead.



Windows will finish up the driver installation from there! Your computer will enumerate with a COM port. You may see a COM port number depending on what is currently saved in your computer. Try to remember what the number is when uploading. If not, you can always navigate back to the device manager and power cycle the Arduino to determine what number your Arduino enumerated on.



For earlier versions of the Arduino boards (e.g. Arduino Duemilanove, Nano, or Diecimila) check out [this page](#) for specific directions.

ARDUINO.CC - GUIDE: WINDOWS

Drivers for RedBoard on Windows

Depending on your board, there may be different USB-to-serial converters on your board. If you are using a RedBoard you will need to go to the [How to Install FTDI Drivers](#) tutorial. If you are using the or RedBoard Qwiic, you will need to go to the [How to Install CH340 Drivers](#) tutorial. Make sure to look closely at your board and its respective hookup guide to determine the USB-to-serial converter that is on board.

How to Install FTDI Drivers

JUNE 4, 2013

How to install drivers for the FTDI Basic on Windows, Mac OS X, and Linux.

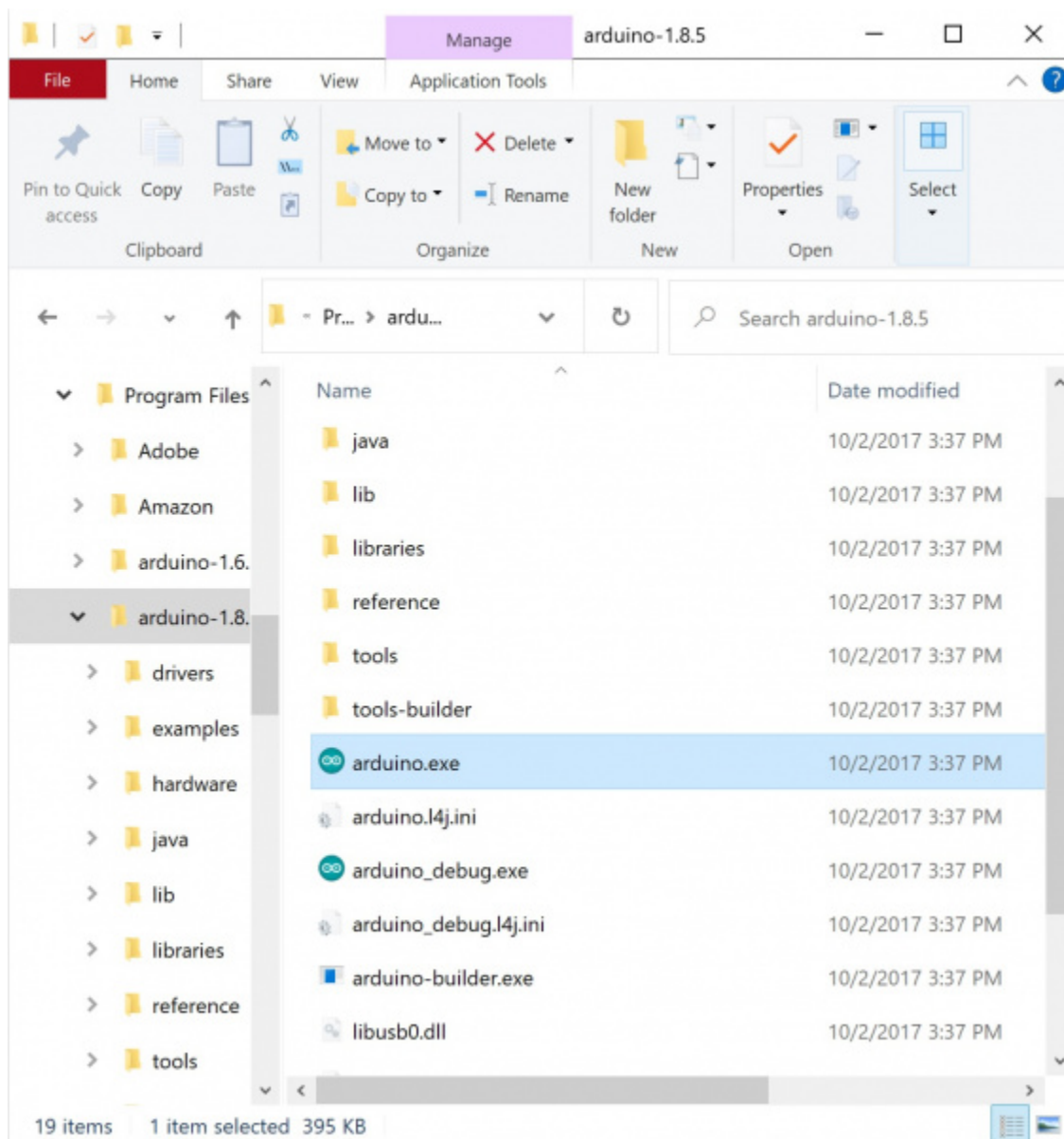
How to Install CH340 Drivers

AUGUST 6, 2019

How to install CH340 drivers (if you need them) on Windows, Mac OS X, and Linux.

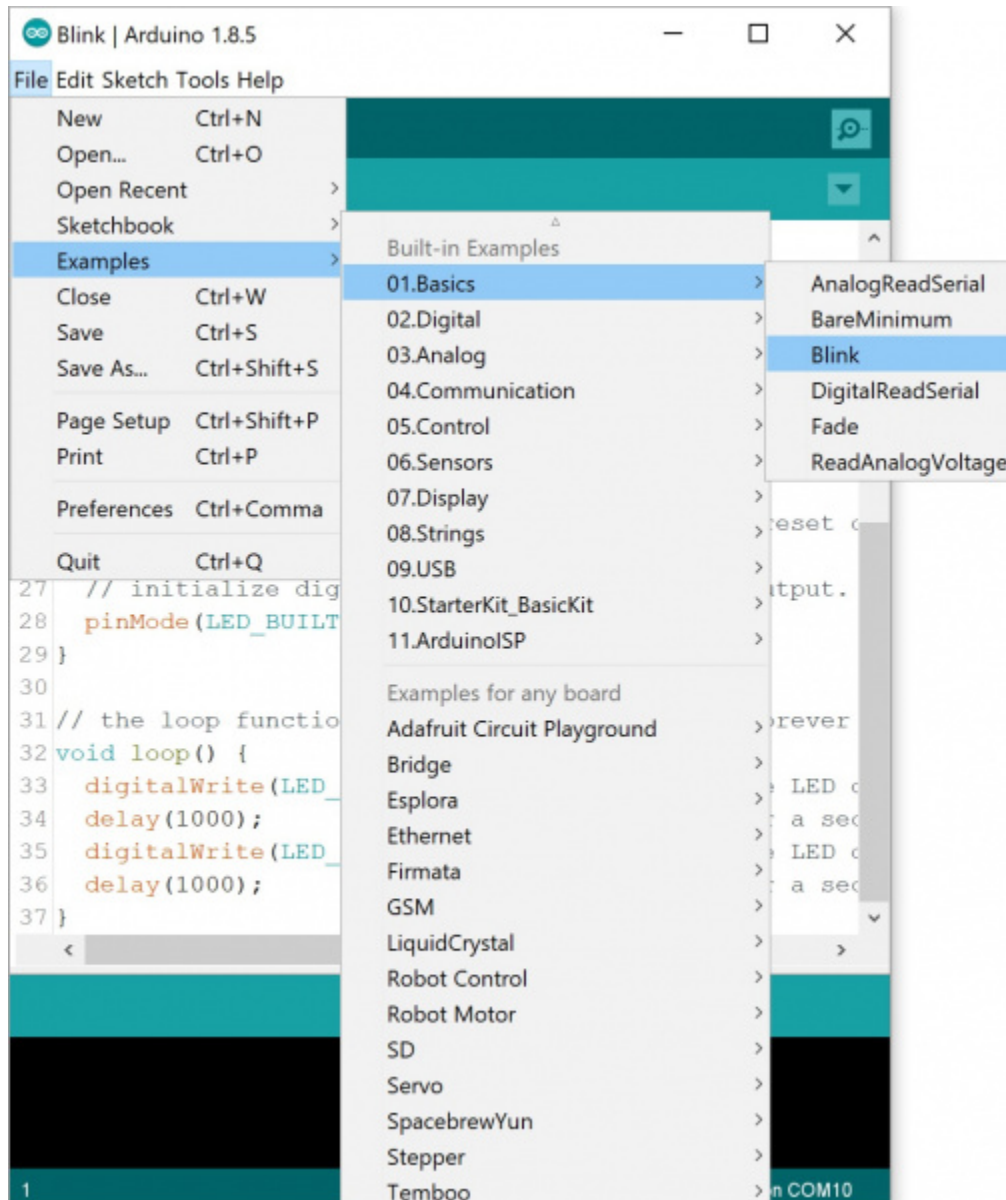
Blink!

After following the appropriate steps for your software install, we are now ready to test your first program with your Arduino board! Launch the Arduino application. If you disconnected your board, plug it back in.

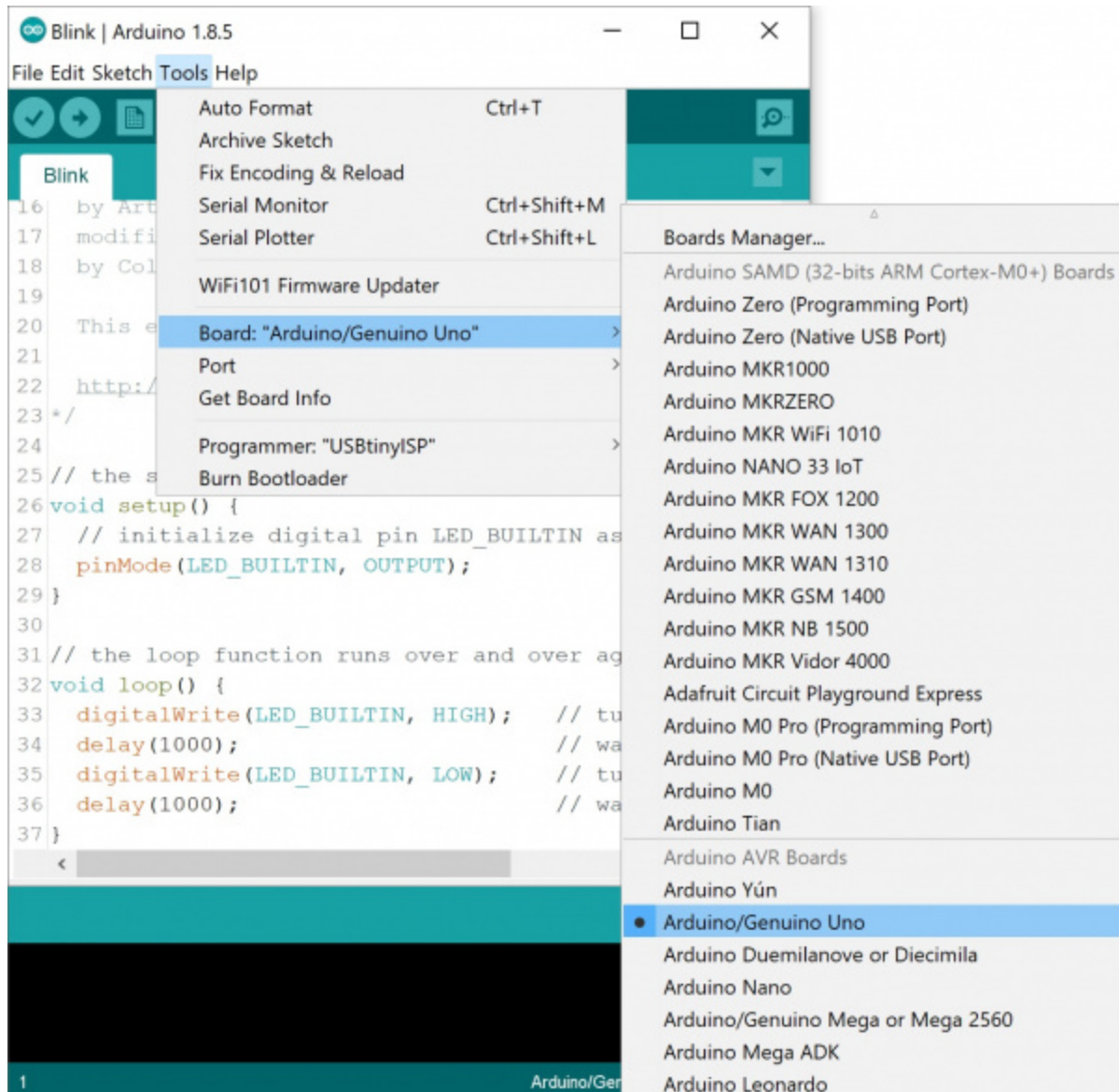


Note: Depending on your method of installing the Arduino IDE, the application may be on your desktop or the program folder.

Open the Blink example sketch by going to: **File > Examples > 01.Basics > Blink.**

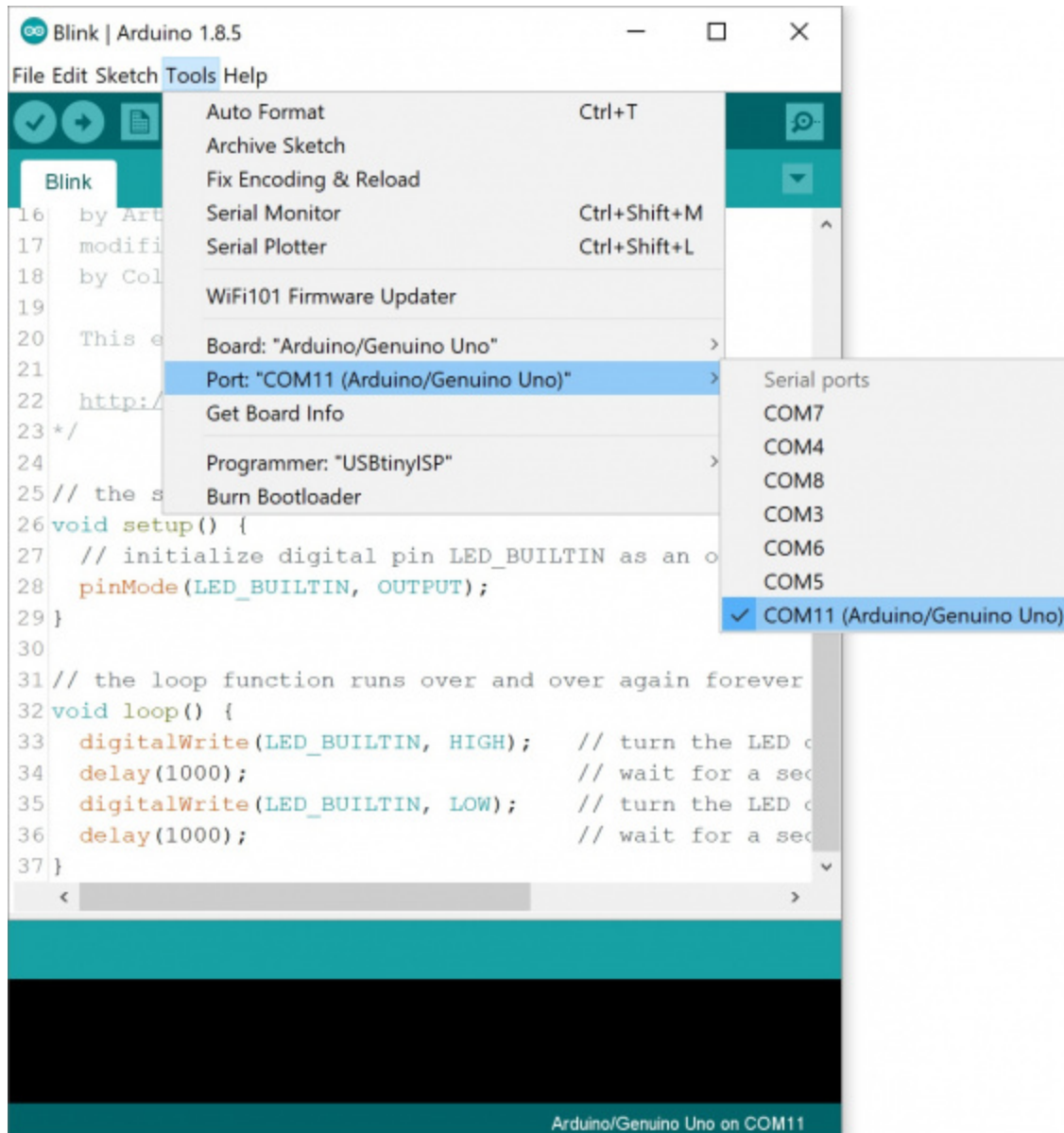


Select the type of Arduino board you're using: **Tools > Board > Arduino Uno.**



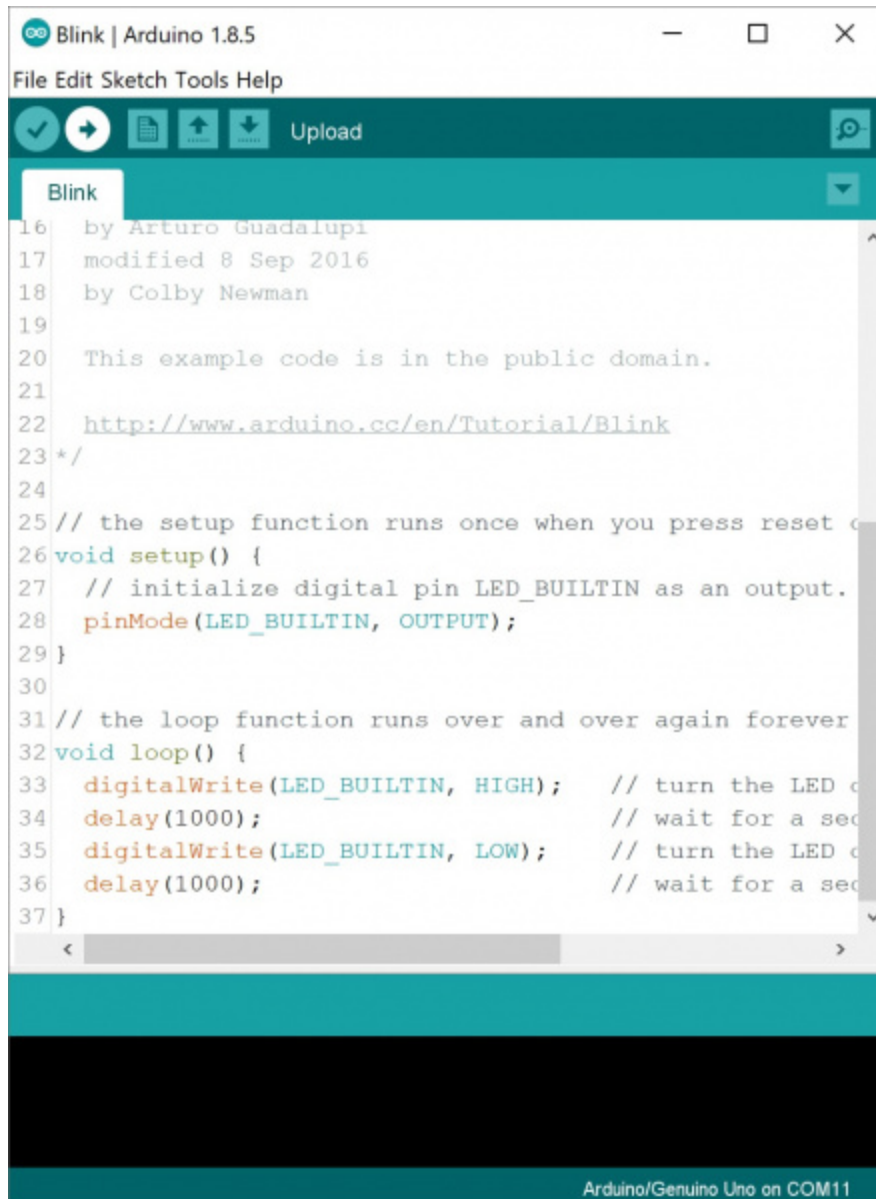
Note: As you move to other architectures, you may need to select a different board definition depending on your development board. For the Arduino Uno R3 and RedBoard development boards with ATmega328P, you can simply select **Arduino Uno**. Certain Arduino IDE versions may have you select **Arduino/ Genuino Uno**.

Select the serial/COM port that your Arduino is attached to: **Tools > Port > COMxx**. In this case it was **COM11**.

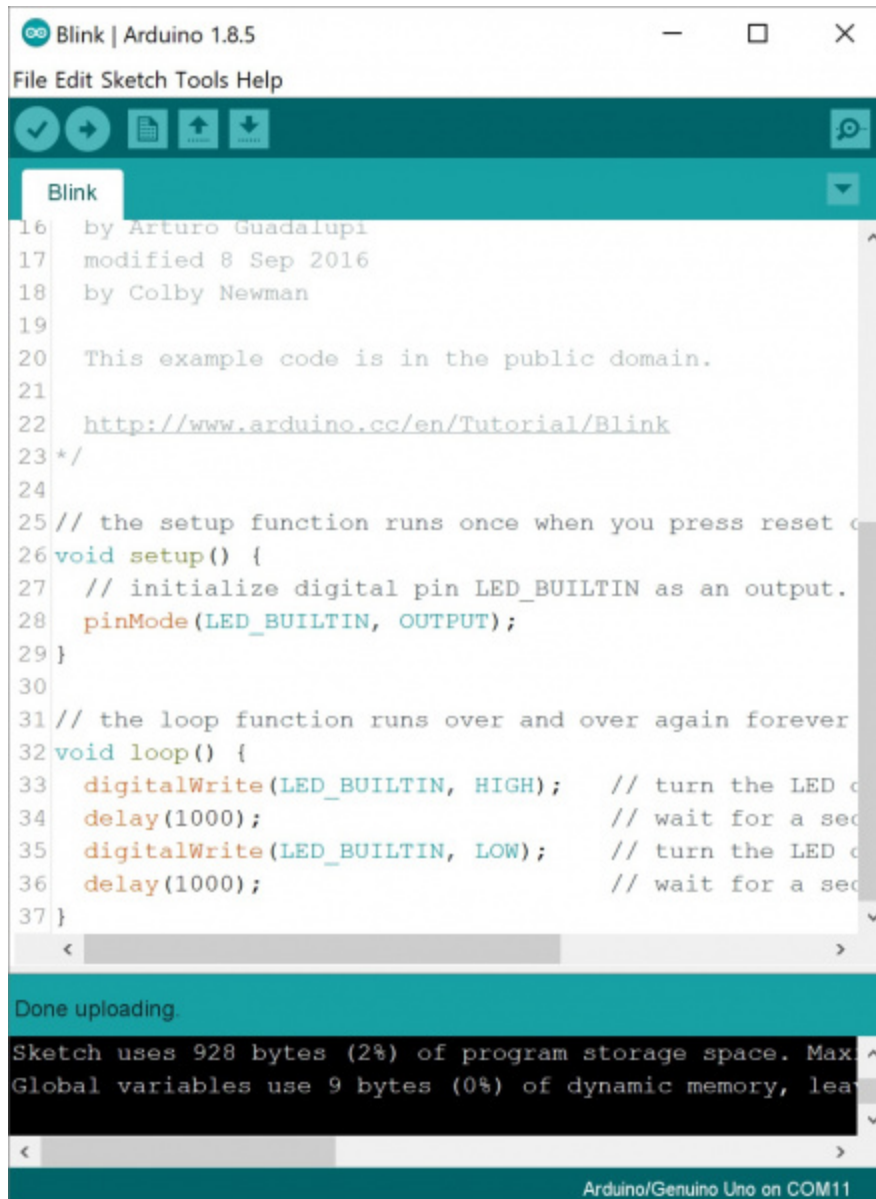


Note: If you're not sure which serial device is your Arduino, take a look at the available ports, then unplug your Arduino and look again. The one that disappeared is your Arduino.

With your Arduino board connected, and the Blink sketch open, press the **"Upload"** button.



After a second, you should see some LEDs flashing on your Arduino, followed by the message **"Done Uploading"** in the status bar of the Blink sketch.



```
16 by Arturo Guadalupi
17 modified 8 Sep 2016
18 by Colby Newman
19
20 This example code is in the public domain.
21
22 http://www.arduino.cc/en/Tutorial/Blink
23 */
24
25 // the setup function runs once when you press reset
26 void setup() {
27   // initialize digital pin LED_BUILTIN as an output.
28   pinMode(LED_BUILTIN, OUTPUT);
29 }
30
31 // the loop function runs over and over again forever
32 void loop() {
33   digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
34   delay(1000);                      // wait for a second
35   digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW
36   delay(1000);                      // wait for a second
37 }
```

Done uploading.

Sketch uses 928 bytes (2%) of program storage space. Max. Global variables use 9 bytes (0%) of dynamic memory, leaving 491 bytes free.

Arduino/Genuino Uno on COM11

If everything worked, the onboard LED on your Arduino should now be blinking! You just programmed your first Arduino!

Note: Depending on the architecture and development board, the built-in LED may be defined on a different pin. You may need to adjust `LED_BUILTIN` or pin 13 to a different value before uploading.

UBIDOTS EXPLORER:

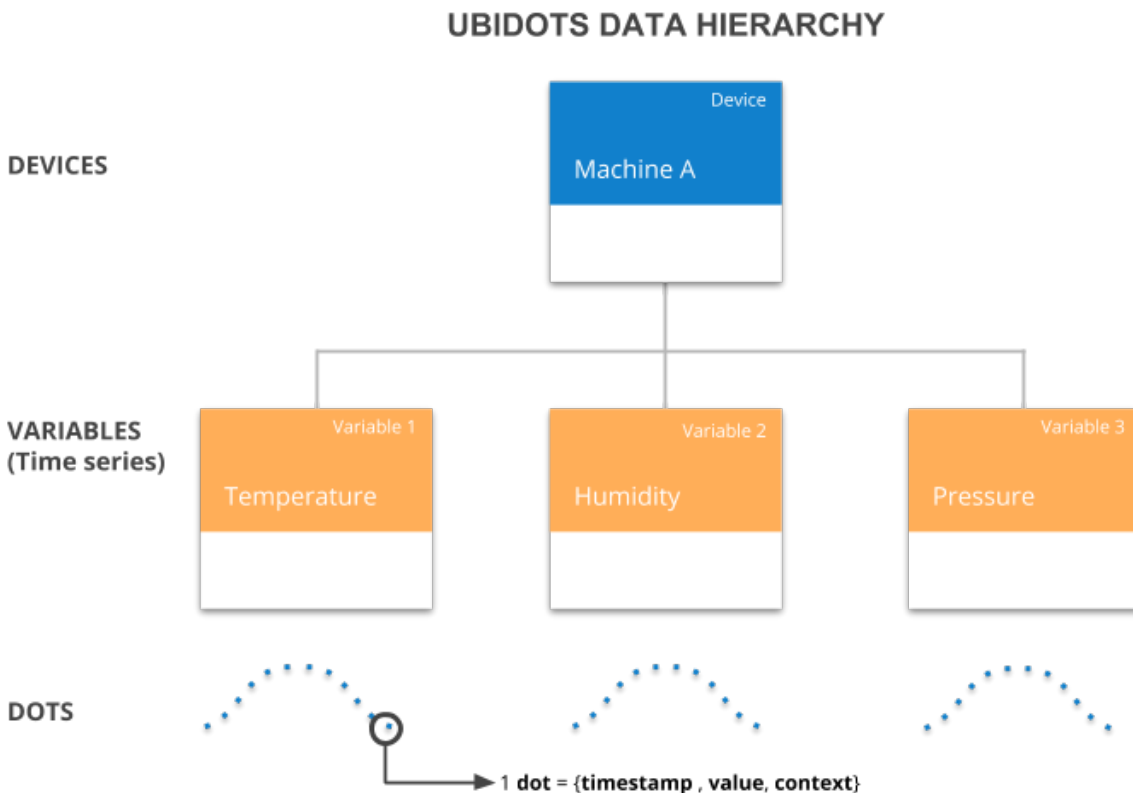
Introduction

Welcome to the Ubidots REST API hardware docs. Here you would reference how to send or retrieve data from your hardware devices using our supported communication protocols: **HTTP**, **MQTT** and **TCP/UDP**.

The purpose of this section is to help you understand what happens backstage when communicating with Ubidots, so you can replicate this in your firmware. For this reason, we avoid the use of examples using libraries. However, if you'd like to use one of our libraries (Python, C, and more), check out our [compatible devices](#) and [API clients](#) sections.

How Ubidots Works?

Every time a device updates a sensor value in a variable, a data-point or "**dot**" is created. Ubidots stores dots that come from your devices inside variables, and these stored dots have corresponding **timestamps**:



Each dot contains these items:

Item	Description	Mandatory
value	A numerical value. Ubidots accepts up to 16 floating-point length numbers.	YES
timestamp	Unix Epoch time, in milliseconds. If not specified, then our servers will assign one upon reception.	NO
context	An arbitrary collection of key-value pairs. Mostly used to store the latitude and longitude coordinates of GPS devices.	NO

Values

A numerical value. Ubidots accepts up to 16 floating-point length numbers.

```
{"value" : 34.87654974}
```

Timestamps

A **timestamp**, as best described [here](#), is a way to track time as a running total of seconds. This count starts at the Unix Epoch on January 1st, 1970 at UTC. Therefore, the unix time stamp is merely the number of seconds between a particular date and the Unix Epoch. Please keep in mind that when you send data to Ubidots, you must set the timestamp in milliseconds; also, if you retrieve a dot's timestamp, it will be in milliseconds.

```
"timestamp" : 1537453824000
```

The above timestamp corresponds to Thursday, September 20, 2018 2:30:24 PM.

PRO-TIP:

A useful tool to convert between Unix timestamps and human-readable dates is [Epoch Converter](#).

Context

Numerical values are not the only data type supported; you can also store *string* or *char* data types inside what we call **context**. The **context** is a key-value object that allows you to store not only numerical but also string values. An example use of the **context** could be:

```
"context" : {"status" : "on", "weather" : "sunny"}
```

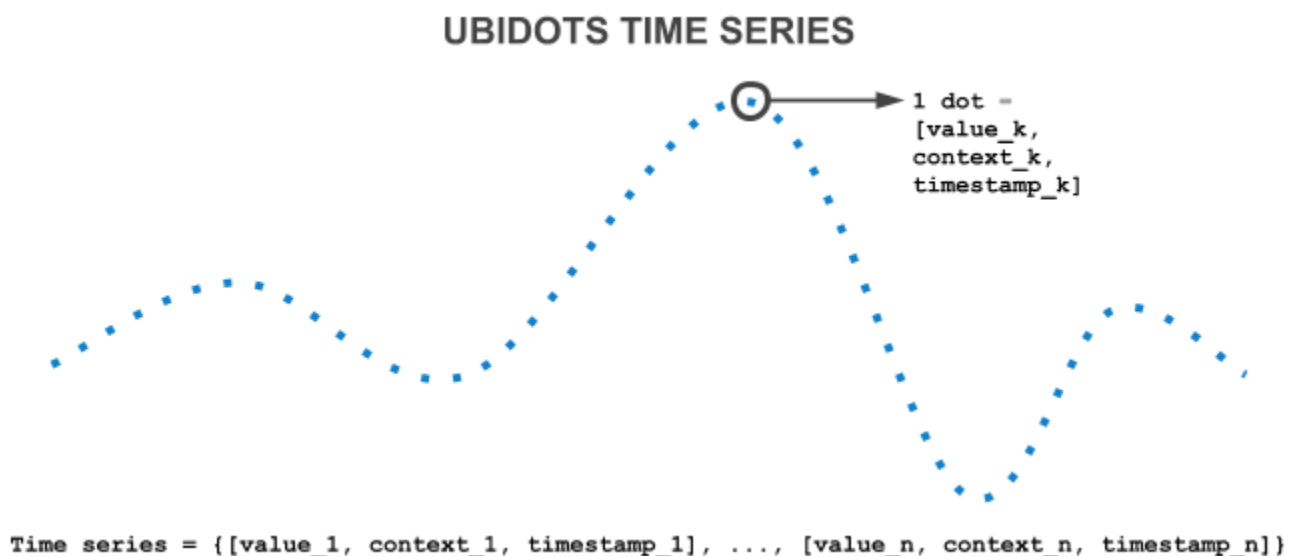
A **context** is commonly used to store the latitude and longitude coordinates of your device for GPS/tracking application use cases. All Ubidots maps uses the *lat* and *lng* keys from a dot's **context** to extract the coordinates of your device, in that way you just need to send a single dot with the coordinates values in the variable context to plot a map instead of sending separately both latitude and longitude in two different variables. Below you can find a typical **context** with coordinates values:

```
"context" : {"lat":-6.2, "lng":75.4, "weather" : "sunny"}
```

Please note that you can mix both string and numerical values in the **context**. If your application is for geo-localization purposes, make sure that the coordinates are set in decimal degrees.

Time Series

Based on the above, we can illustrate an Ubidots time series like this:



Ubidots is an agnostic platform, this means that it does not really care what hardware device you are using, as long as you're able to interact with us through at least one of these protocols:

- [HTTP](#)
- [MQTT](#)
- [TCP/UDP](#)

HTTP

HTTP is the main internet protocol for information exchange in the Internet. It also supports data encryption, which is usually called *secure HTTP* or *HTTPS*. When you use a web browser (MS Edge, Firefox, Google Chrome, etc) you are sending and receiving packages

using *HTTP* or *HTTPS*. Because of this, it was not surprising to see *HTTP*s become a popular option for data exchange within IoT.

When a developer is to implement *HTTP* communication, then he should look for a *RESTful Application Programming Interface*, or *REST API*, which exposes all the endpoints required to interact with a third party application (like Ubidots). In this section, you can reference our *REST API* to understand how to send and retrieve data from our servers.

Note: Our goal with this section is to help you understand what's happening behind when communicating with Ubidots, enabling you to replicate it within your own firmware. Because of this, we avoid using our custom libraries in all the examples.

HTTP requests

The following methods are specified within the *HTTP* standard:

HTTP Method	Description
GET	Used to retrieve information
POST	Used to create a new element
PATCH	Used to update existing elements
DELETE	Used to delete existing elements

HTTP is a **request/response** protocol, this means that every request that you make is answered by the server. This response includes a number (response code) and a body. For example, when you make a request to retrieve a file on a webpage "(e.g. *"Get me the file 'websiteside.html'"*)", you build a *GET* request. If the request is correct, the server will typically return a 200 response code, along with the file requested (body).

Under the same logic, when a device sends a *GET* request to the Ubidots' server (i.e. requesting the last value of a variable), then the server sends back a response with the status of the request (response code), and a body, which would be the a value with its context and timestamp.

An *HTTP* request also needs the parameters below:

- **Host:** Specifies the server you will be making *HTTP* requests to.
- **Path:** This is typically the remaining portion of the URL that specifies the resource you want to consume, be it a variable or a device. For example, if an API endpoint is: `industrial.api.ubidots.com/api/v1.6/devices/my-device` then the path would be `/api/v1.6/devices/my-device`

- **Headers:** Define the operating parameters of the *HTTP* request such as authentication, Content-Type, Content-Length, etc.
- **Body/payload:** In the case of POST and PATCH requests, this is the data sent by your device to the server. GET requests typically do not have a body because they are meant to request data, not to send data.

Ubidots accepts data as *JavaScript Object Notation* or *JSON*. *JSON* is a typical *HTTP* data type, it is a collection of name/value pairs. In various programming languages, this is treated as an object, record, struct, dictionary, hash table, keyed list, or associative array. It is also human readable and language independent. An example of a *JSON* data type that Ubidots accepts can be referenced below:

```
{"temperature": {"value":10, "timestamp": 1534881387000, "context": {"machine": "1st floor"}}}
```

A typical *HTTP* request to Ubidots should be set as below:

```
POST {PATH} HTTP/1.1<CR><LN>
Host: {HOST}<CR><LN>
User-Agent: {USER_AGENT}<CR><LN>
X-Auth-Token: {TOKEN}<CR><LN>
Content-Type: application/json<CR><LN>
Content-Length: {PAYLOAD_LENGTH}<CR><LN><CR><LN>
{PAYLOAD}
<CR><LN>
```

Where:

- {PATH}: Path to the resource to consume
Example: /api/v1.6/variables/ to get user's variables information.
- {HOST}: Host URL.
Example: industrial.api.ubidots.com
- {USER_AGENT}: An optional string used to identify the type of client, be it by application type, operating system, software vendor or software version of the requesting user agent.
Examples: ESP8266/1.0, Particle/1.2
- {TOKEN}: Unique key that authorizes your device to ingest data inside your Ubidots account.
- {PAYLOAD_LENGTH}: The number of characters of your payload.
Example: The payload {"temperature": 20} will have a content-length of 19.

- **{PAYLOAD}**: Data to send.
Example: {"temperature": 20}

PRO TIP:

The easy way of handle *HTTP* and *HTTPs* requests is by using **cURL**, a command line tool. To learn how to install cURL on Windows, MacOSX or Linux please refer to this [guide](#).

API URLs

API access can be over *HTTP* or secure HTTP, *HTTPs*, using the following endpoints based on your Ubidots Account type.

Security Note: We strongly advise to use *HTTPs* to make sure your data travels encrypted, avoiding the exposure of your API token and/or sensor data.

HTTP

Ubidots Account	Endpoint	Port
Educational	http://things.ubidots.com	80
Industrial	http://industrial.api.ubidots.com	80

HTTPs

Ubidots Account	Endpoint	Port
Educational	https://things.ubidots.com	443
Industrial	https://industrial.api.ubidots.com	443

Ubidots supports SSL v1.1, TLS v1.2 and v1.3. You can download our root certificates in different formats:

- [PEM file](#): A certificate chain with two root certificates from our certificate authorities (CAs).
- [DER file](#): Same as the PEM file, with an alternative encoding.
- [CRT file](#): Same as the PEM file, with a different extension. Often referred to as .crt, .cert or .cer.

NOTE:

To avoid future inconveniences, we advise to **use both certificates** in your device's auth routines. While the second is the current valid certificate, we will use the first after 2021.

Fixed IPs

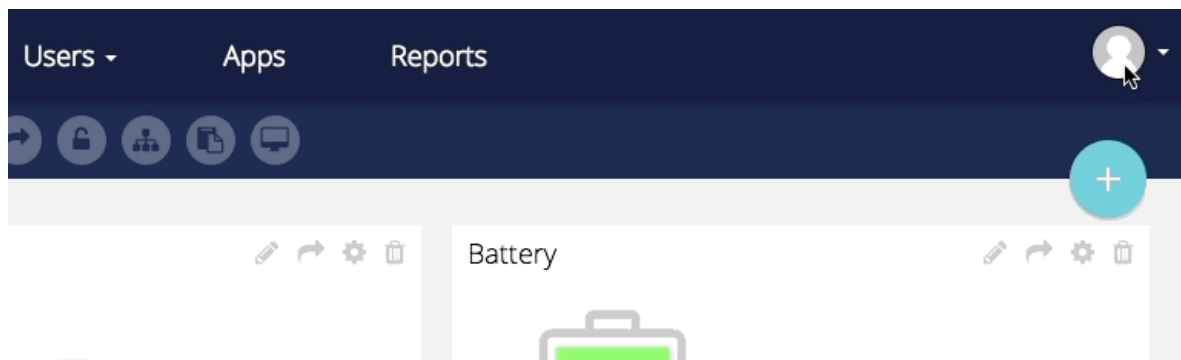
Ubidots also supports fixed IPs address for data ingestion, but we highly advise our users to implement their firmware routines using the dns endpoints instead of fixed IPs, as these last ones may change in the future.

Ubidots Account	ipv4	ipv6	Port
Educational	50.23.124.68	2607:f0d0:2101:39::2	80/443
Industrial	169.55.61.243	2607:f0d0:3:b::2	80/443

HTTP Authentication

Every request requires a *token*. A *token* is an unique key that authorizes your device to ingest data inside your Ubidots account.

The easiest way to get yours is clicking on "**API Credentials**" under your profile tab:



Also, 6 hour valid *tokens* can be generated using your account *API-KEY*. If you desire to learn how to do it, please refer the [Ubidots REST Software API reference](#).

There are two ways to send a *token* in a request:

1. Send the *token* as a parameter in the endpoint, setting it with the `?token=expression`. This is the easiest one to implement but the most insecure method to send your authentication credentials.
2. Send the *token* as a request header. For this, use the `X-Auth-Token` header.

Security Note: While sending your *token* as a URL parameter can be quite comfortable in some devices, we only advise to do this for prototyping stages. For production stages, we strongly advise sending the token using the `X-Auth-Token` header.

GET

<https://industrial.api.ubidots.com/api/>

Send data

Our REST API allows you send data to the platform in two different ways:

- Send data to a **device**
- Send data to a **variable**

If you're looking for more advanced API capabilities, such as editing a device name, or listing all of your devices, please refer to our Ubidots Software REST API Reference.

Send data to a device

Request structure:

```
POST /api/v1.6/devices/{DEVICE_LABEL} HTTP/1.1<CR><LN>
Host: {Host}<CR><LN>
User-Agent: {USER_AGENT}<CR><LN>
X-Auth-Token: {TOKEN}<CR><LN>
Content-Type: application/json<CR><LN>
Content-Length: {PAYLOAD_LENGTH}<CR><LN><CR><LN>
{PAYLOAD}
<CR><LN>
```

Expected Response:

```
HTTP/1.1 200 OK<CR><LN>
Server: nginx<CR><LN>
Date: Tue, 04 Sep 2018 22:35:06 GMT<CR><LN>
Content-Type: application/json<CR><LN>
Transfer-Encoding: chunked<CR><LN>
Vary: Cookie<CR><LN>
Allow: GET, POST, HEAD, OPTIONS<CR><LN><CR><LN>
{PAYLOAD_LENGTH_IN_HEXADECIMAL}<CR><LN>
{"{VARIABLE_LABEL}": [{"status_code": 201}]}<CR><LN>
0<CR><LN>
```

The easiest way to **send values to Ubidots** is by specifying your Device label in the request *path* and making a POST request to it.

Replace the {DEVICE_LABEL} with the unique label of the desired device to send data to. If the device does not exist in your Ubidots account, it will be automatically created. Device labels can contain any alphanumeric character, without blank spaces, i.e *weather-station*, *my-home*, *esp8266*.

To set the variables values, just create a *JSON* data payload as below

N° Variables	JSON structure
One	<code>{"variable-label": VALUE}</code>
Multiple	<code>{"variable-label-1": {"value": VALUE, "timestamp": TIMESTAMP, "context": {"key": VALUE}}, ... , "variable-label-n": {"value": VALUE, "timestamp": TIMESTAMP, "context": {"key": VALUE}}}</code>

timestamp and context parameters are optional.

NOTE 1: By default, the Ubidots backend will take the latitude and longitude values inside any variable labeled as **"gps"**, **"location"**, **"position"** or **"coordinates"** automatically to get your device coordinates in all of the map widgets. Alternatively, you change the default location variable.

NOTE 2: All timestamps must be in milliseconds. If you set a timestamp for a variable value, it will be stored with the date represented by it. You can easily convert dates to timestamps values.

Implementation details:

In the 0th review we chose and confirmed the title by our guide.

In the 1st review get the literature survey and what is the proposed system.

In the 2nd review the components and connection are given as per the block diagram.

Conclusion:

References:

- [1] Mr. Godson Michael D'silva, Mr. Anoop Kunjumonz Achariah, Mr. Lukose Roy Pannapara and Ms. Jessica John Joseph "Smart Ticketing System for Railways in Smart Cities using Software as a Service Architecture" International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (ISMAC 2017)
- [2] Md. Faisal Mehdi Hassan, *Golam Tangim, *Md. Kafiul Islam, *Md. Rezwanul Haque Khandokar, *** Arif UI Alam "RFID-based Ticketing for Public Transport System" 978-1-4244-5540- 9/10/\$26.00 ©2016 IEEE
- [3] Kirti Dhiman¹, Er. CK Raina² "IoT Based Ticket Checking System" Vol. 6, Issue 3, March 2017
- [4] Vijayashanthi. R, Radha. N, Jaya Shree. M, Sindhuja. V "Fingerprint Authentication using Raspberry Pi based On IoT"
- [5] Dhvani. K. Shah, Dr. Vinayak A. Bharadi, V. J. Kaul, Sameer Amrutia "End-to-end Encryption based Biometric SaaS" International Journal of Pure and Applied Mathematics Special Issue
- [6] Syamala, M., Nalini, N.J., Ragupathy, R., Magalur, L.P., "Random forest classifier approach for blurred images", (2017) International Journal of Pure and Applied Mathematics, 116 (6 Special Issue), pp. 67-72.
- [7] Utkam, S.G., Parasa, R., Kondapaneni, A., Tulabandula, P.R.K., Kuppalu, D.R., "A secured symmetric key encryption technique using images as secret keys", (2017) International Journal of Pure and Applied Mathematics, 116 (6 Special Issue), pp. 149-153.
- [8] Veeraiah, T., Mahamkali, A., Kishore, B.N.P., Rao, A.N., Rao, S.K.M., "Image denoising by profundity map method using diagram base change and group scanty", (2017) Mathematics, 115 (8 Special Issue), pp. 143-149.
- [9] Veeraiah, T., Rao, S.K.M., "Classification of fuzzy rule based order-statistic filters for color images", (2017) International Journal of Pure and Applied Mathematics, 115 (8 Special Issue), pp. 189-194.

